

1. CppTips	5
1.1 BewareOfStringPromotion	5
1.2 BewareStdStringLiterals	5
1.3 NeverUseStaticLocalVariables	6
1.4 NoUsingNamespacelnHeaders	6
1.5 PrivateLocking	7
1.6 ReturnStdStringByValue	7
1.7 ScopedLocking	7
1.8 SharedPtr	7
1.9 ValgrindBadSuppressions	8
2. Index	8
2.1 0.6 Feature Matrix	9
2.1.1 0.6 Feature Descriptions	11
2.1.2 0.6 Interoperability Matrix	16
2.2 AMQP0-9-DesignNotes	17
2.3 AMQP (Advanced Message Queueing Protocol)	17
2.4 AMQP Brokers	18
2.4.1 AMQP Messaging Broker (implemented in C++)	18
2.4.2 AMQP Messaging Broker (implemented in Java)	18
2.5 AMQP Messaging Clients	19
2.5.1 AMQP .NET Messaging Client	19
2.5.2 AMQP C++ Messaging Client	19
2.5.2.1 Client configuration	19
2.5.3 AMQP Java JMS Messaging Client	19
2.5.4 AMQP Python Messaging Client	20
2.5.5 AMQP Ruby Messaging Client	20
2.6 AMQP Test Suites for Apache Qpid	20
2.7 C++ vs. Python API Discrepancies	20
2.8 ClusteringHA	24
2.8.1 AMQP breakdown for clustering	25
2.8.2 Cluster Design Note	27
2.8.3 Cluster Failover Modes	29
2.8.4 ClusteringAndFederation	31
2.8.5 Federation Design Note	32
2.8.6 Java Federation Design Proposal	33
2.8.7 Old Clustering Design Note	34
2.8.8 Persistent Cluster Restart Design Note	37
2.8.9 Reliability Requirements	38
2.9 Declarative System Testing	39
2.10 Developer Pages	40
2.10.1 Java Coding Standards	40
2.10.2 Cpp Client Java Interop Issues	47
2.10.3 Java Broker Design	47
2.10.3.1 Qpid Design - Framing	48
2.10.3.2 Qpid Design - Management	49
2.10.3.3 Qpid Design - Threading	49
2.10.3.4 Qpid Design - Message Acknowledgement	51
2.10.3.5 Java Broker Design - MessageStore	52
2.10.3.5.1 BDBMessageStore (3rd Party)	52
2.10.3.5.2 JDBCStore	52
2.10.3.5.3 MemoryMessageStore	52
2.10.3.6 Restructuring Java Broker and Client Design	52
2.10.3.7 Message API Design	55
2.10.3.8 Java Architecture Overview	58
2.10.3.9 Producer flow control	58
2.10.3.9.1 Java Broker - AMQP0-9 Tactical Producer Flow Control	59
2.10.3.10 Java Broker Refactor (QPID-950)	62
2.10.3.11 Java Broker Modularisation	63
2.10.3.12 Java Broker Configuration Design	63
2.10.3.13 Java Broker Design - Flow to Disk	64
2.10.3.13.1 FtD Code Review Notes	72
2.10.3.14 Java Broker Design - High Level Overview of Refactoring	74
2.10.3.15 Java Broker Design - Message Representation	77
2.10.3.16 Network IO Interface	83
2.10.3.16.1 Network IO Interface discussion points	85
2.10.3.16.2 New common network and protocol interfaces	87
2.10.3.16.3 Port server to new interface	93
2.10.3.17 Java Broker Design - Operational Logging	102
2.10.3.17.1 Existing Logging Analysis	105
2.10.3.17.2 Logging Format Design	106
2.10.3.17.3 Status Update Design	107
2.10.3.18 Qpid Design - Queue Implementation	142
2.10.3.19 Qpid Design - Message Delivery	147
2.10.3.20 Java authorization plugins	153
2.10.3.21 0.6 Broker BasicFlow Synchronisation Design	153
2.10.3.22 Slow Consumer Disconnect	154
2.10.3.23 Topic Configuration Design	157
2.10.4 Qpid Java Client refactoring	158
2.10.5 Distributed Testing	158
2.10.6 Low-Level API Diagram	162

2.10.7 Weekly QPID Developer Meetings	162
2.10.7.1 Qpid Java Meeting Minutes 04-04-2008	162
2.10.7.2 Qpid Java Meeting Minutes 11-04-2008	165
2.10.7.3 Qpid Java Meeting Minutes 28-03-2008	169
2.10.7.4 Qpid Java Meeting Minutes 2008 05 02	170
2.10.7.5 Qpid Java Meeting Minutes 2008-04-18	176
2.10.7.6 Qpid Java Meeting Minutes 2008-05-09	180
2.10.7.7 Qpid Java Meeting Minutes 2008-05-16	191
2.10.7.8 Qpid Java Meeting Minutes 2008-05-23	195
2.10.7.9 Qpid Java Meeting Minutes 2008-05-30	198
2.10.7.10 Qpid Java Meeting Minutes 2008-06-20	202
2.10.7.11 Qpid Java Meeting Minutes 2008-06-27	208
2.10.7.12 Qpid Java Meeting Minutes 2008-07-11	213
2.10.7.13 Qpid Java Meeting Minutes 2008-07-25	214
2.10.7.14 Qpid Java Meeting Minutes 2008-08-01	221
2.10.7.15 Qpid Java Meeting Minutes 2008-08-08	226
2.10.7.16 Qpid Java Meeting Minutes 2008-08-15	230
2.10.7.17 Qpid Java Meeting Minutes 2008-08-22	234
2.10.8 Documentation	237
2.10.8.1 Build Creator	237
2.10.8.2 Cheat Sheet for configuring Exchange Options	242
2.10.8.3 Cheat Sheet for configuring Queue Options	243
2.10.8.3.1 LVQ Example	245
2.10.8.3.2 queue state replication	248
2.10.8.4 Documentation2	250
2.10.8.5 DocumentationB	253
2.10.8.5.1 .NET Client	253
2.10.8.5.2 C++ Broker	253
2.10.8.5.3 C++ Client	254
2.10.8.5.4 Example guide	254
2.10.8.5.5 Java Broker	254
2.10.8.5.6 JMS Client	256
2.10.8.5.7 Python Client	256
2.10.8.5.8 Ruby Client	256
2.10.8.6 Java Broker Analysis Tools	256
2.10.8.7 LVQ	259
2.10.8.8 QMan - Qpid Management bridge	265
2.10.8.8.1 Get me up and running	265
2.10.8.8.2 JMX Interface Specification	269
2.10.8.8.3 QMan Components View	277
2.10.8.8.4 QMan Messages Catalogue	281
2.10.8.8.5 QMan System Overview	281
2.10.8.8.6 QMan User Guide	282
2.10.8.8.7 WS-DM Interface Specification	291
2.10.8.9 Qpid ACLs	333
2.10.8.10 Qpid Interoperability Documentation	333
2.10.8.11 SSL	334
2.10.8.12 Starting a cluster	335
2.10.8.13 Use of Get()	337
2.10.8.14 Using Broker Federation	338
2.10.9 ACL	345
2.10.9.1 FileACL Design	349
2.10.10 Qpid Management Framework	350
2.10.10.1 QMan	354
2.10.10.2 QMF Map Message Protocol	357
2.10.10.3 QMF Protocol	362
2.10.10.4 QMF Python Console Tutorial	373
2.10.10.5 QMFv2 Project Page	380
2.10.10.5.1 QMFv2 APIs	381
2.10.10.5.2 QMFv2 Architecture	403
2.10.11 Broker job queue limits	405
2.10.12 JMX Console Use Cases	405
2.10.13 Current Architecture	405
2.10.14 MessageProducer.send() behaviour	409
2.10.15 Multiple Java Brokers - Use Cases	409
2.10.16 Java Client Test Coverage	410
2.10.17 ACL Design	411
2.10.17.1 andrew acl proposal	411
2.10.17.1.1 Method Considered Harmful	413
2.10.17.1.2 Method Considered Harmful Redux	414
2.10.18 AMQP Distributed Transaction Classes (C++)	417
2.10.19 API Error Conditions	417
2.10.20 Broker Management QMF Coverage	418
2.10.21 Java Client Design	427
2.10.21.1 0.6 Java Client Dispatcher Changes	427
2.10.21.1.1 0.6 Java Client Dispatcher Changes - Details	430
2.10.22 Qpid extensions to AMQP	431
2.10.23 Qpid Java Broker - Guidance for 64Bit VM	433
2.11 Download	434
2.11.1 The AMQP Distributed Transaction Classes (Java)	437

2.11.2 AMQP compatibility	437
2.11.2.1 Queue Replay	439
2.12 Getting Involved	441
2.12.1 GSoC	442
2.12.2 OSVC	443
2.12.3 Qpid Project Etiquette Guide	444
2.13 HermesJMS	446
2.14 Informal M2.1 code review 2008-03-18	450
2.15 Navigation	450
2.15.1 Acknowledgments	450
2.15.2 FAQ	451
2.15.3 License	461
2.15.4 Project Status	462
2.16 People	463
2.16.1 MartinRitchie	464
2.16.2 Robbie Gemmell	464
2.17 Proposal for a new JMS Destination configuration	466
2.17.1 Proposal for a new JMS Destination configuration2	468
2.18 Qpid .Net Documentation	470
2.18.1 .NET User Guide	470
2.18.2 Excel AddIn	482
2.18.3 Qpid .Net How To	483
2.18.3.1 Build .NET Client	483
2.18.3.2 Releasing	484
2.18.3.3 Run tests	484
2.18.3.4 Setup .Net Client on Windows	484
2.18.4 WCF	485
2.19 Qpid 'C++' Documentation	486
2.19.1 CppApiGuide	487
2.19.2 CppBrokerStartPlugins	487
2.19.3 CppEventChannello	487
2.19.4 CppHandlerChains	489
2.19.5 CppStyleGuide	489
2.19.6 Persistent Message Store Module	490
2.19.7 PythonBrokerTest	491
2.20 Qpid Integrations	491
2.21 Qpid Java Documentation	491
2.21.1 3rd Party Libraries	493
2.21.2 3rd Party Tools	494
2.21.2.1 Mule	494
2.21.3 AMQP Error Codes	494
2.21.4 Example Classes	495
2.21.5 Getting Started	496
2.21.5.1 MgmtC++	497
2.21.5.2 RAJB	504
2.21.5.3 RASC	504
2.21.6 Getting Started Guide	507
2.21.7 Java broker log monitoring	509
2.21.8 Java Environment Variables	512
2.21.9 JMS Compliance	512
2.21.10 Management Design notes	513
2.21.10.1 JMX Gateway	524
2.21.10.2 qmf_architecture	524
2.21.11 Management Tools	526
2.21.11.1 JConsole	526
2.21.11.2 MessageStore Tool	526
2.21.11.3 Qpid JMX Management Console	527
2.21.11.3.1 Configuring Management Users	528
2.21.11.3.2 Configuring Qpid JMX Management Console	528
2.21.11.3.3 Qpid JMX Management Console FAQ	531
2.21.11.3.4 Qpid JMX Management Console User Guide	531
2.21.11.3.5 Qpid Management Features	542
2.21.12 Multiple AMQP Version Support	543
2.21.12.1 AMQPVersion.1	546
2.21.13 Qpid Java FAQ	549
2.21.14 Qpid Java How To	557
2.21.14.1 Add New Users	557
2.21.14.2 Configure ACLs	559
2.21.14.2.1 Java XML ACLs	559
2.21.14.3 Configure Broker and Client Heartbeating	564
2.21.14.4 Configure Java Qpid to use a SSL connection	564
2.21.14.5 Configure Log4j CompositeRolling Appender	565
2.21.14.6 Configure Operational Status Logging	566
2.21.14.7 Configure the Broker via config.xml	570
2.21.14.7.1 M2.1 - config.xml	570
2.21.14.7.2 M2 - config.xml	573
2.21.14.8 Configure the Virtual Hosts via virtualhosts.xml	576
2.21.14.9 Debug using log4j	578
2.21.14.10 Firewall Configuration	578
2.21.14.11 How to Tune M3 Java Broker Performance	580

2.21.14.12 How to Use JNDI	581
2.21.14.12.1 Using Qpid with other JNDI Providers	583
2.21.14.13 Interact with a JMX MBean	584
2.21.14.14 Qpid Java Build How To	585
2.21.14.14.1 Building	588
2.21.14.15 Split configuration files	604
2.21.14.16 Tune Broker and Client Memory Usage	605
2.21.14.17 Use Last Value Queues (LVQ)	606
2.21.14.18 Use Priority Queues	607
2.21.14.19 Use Producer Flow Control	608
2.21.15 Qpid Java Run Scripts	610
2.21.16 Qpid Troubleshooting Guide	611
2.21.17 Release Plans	612
2.21.18 roadmap	612
2.21.18.1 looking to pitch in	613
2.21.19 Sustained Tests	615
2.21.20 System Properties	616
2.21.21 URL Formats	618
2.21.21.1 0.10 Connection URL Format	618
2.21.21.2 BindingURLFormat	619
2.21.21.3 Connection URL Format	620
2.21.21.4 Url Format Proposal	621
2.21.21.4.1 Qpid Java Broker Management CLI	624
2.22 Qpid Meetup at ApacheCon 2009	629
2.23 Qpid Release Page	629
2.23.1 0.6 Release	629
2.23.2 M1 Release	632
2.23.2.1 M1 Release Check list	632
2.23.3 M2 Release	632
2.23.4 M4 Release Process Notes	633
2.23.5 Qpid Release Notes	633
2.23.5.1 Qpid Java M1 Release Notes	633
2.23.6 QpidReleaseProcess	634
2.23.7 RC Multi-Platform Testing	637
2.24 Qpid Ruby Documentation	637
2.25 Qpid Testing	637
2.25.1 Interop Testing Specification	637
2.25.2 Java Unit Tests with InVM Broker	649
2.25.3 Performance, Reliability and Scaling	650
2.25.3.1 Latency	652
2.25.3.2 Reliability	653
2.25.3.3 Throughput	653
2.25.4 Qpid JMX Management Console Testing Guide	653
2.25.4.1 Qpid Management Console Testing (Old UI)	666
2.25.5 Testing Design - Java Broker CPU GC Monitoring	671
2.26 Source Repository	674
2.26.1 Mailing Lists	674
2.27 Useful Links	675

CppTips

Some advice on makefiles and build system

- [SingleMakefile]

This is a collection of coding guidelines, some specific to Qpid, some just good practice in C++.

- PrivateLocking
- ScopedLocking
- SharedPtr
- BewareStdStringLiterals
- NeverUseStaticLocalVariables
- BewareOfStringPromotion
- ReturnStdStringByValue
- NoUsingNamespaceInHeaders
- ValgrindBadSuppressions

BewareOfStringPromotion

`std::string` is a useful tool for simplifying memory management of strings and avoiding unnecessary copies by reference counting. However there is one common gotcha where it *causes* unnecessary copies. Consider:

```
void f(const std::string& s) { cout << s << endl };

void g() {
    for (int i = 0; i < 1000; ++i) { f("hello"); }
}
```

This actually allocates, copies and deletes 1000 heap buffers with the string "hello"! The problem here is that "hello" is *not* an instance of `std::string`. It is a `char[5]` that must be converted to a temporary `std::string` using the appropriate constructor. However `std::string` always wants to manage its own memory, so the constructor allocates a new buffer and copies the string. Once `f()` returns and we go round the loop again the temporary is deleted along with its buffer.

Here's a better solution:

```
void f(const std::string& s) { cout << s << endl };
namespace { const std::string hello("hello"); }
void g() {
    for (int i = 0; i < 1000; ++i) { f(hello); }
}
```

This time we have a constant `std::string` that is created once at start up and destroyed once at shut-down. The anonymous namespace makes the constant private to this .cpp file so we won't have name clashes. (It's similar to using the `static` keyword on a global declaration in C, but anonymous namespaces are the preferred way to do it in modern C++)

BewareStdStringLiterals

The short story: in C++ code using `std::string` never use string literals except to initialize static-scoped `std::string` constants. (And by the way: [NeverUseStaticLocalVariables](#))

The long story: `std::string` is all about avoiding copies. Reference counting and copy-on-write serve to maximise the sharing of a single heap-allocated char array while maintaining memory safety. When used consistently in a program it works rather nicely.

However, when mixed with classic C-style string literals `std::string` can actually *cause* needless heap-allocated copies. Consider these innocent looking constructs:

```

void f(const std::string& s);
void g(const std::string& s = "hello");
std::string h() { return "foo"; }

void copy_surprise {
    std::string x = "x"; // 1
f("y"); // 2
g(); // 3
x = h(); //4
while (x != "end") { ... } // 4
}

```

Lines 1-4 all cause creation and destruction of an implicit temporary `std::string` to hold the literal value. Line 5 does this for every execution of the while loop. That's a new/memcpy/delete each time. The heap is a heavily used resource, in tight inner loops in multi-threaded code this can be a severe contention bottleneck that cripples scalability.

Use static class `std::string` constants or file-private constants instead. You can make global declarations file-private by using a nameless namespace (this is preferred over the use of the `static` keyword.)

```

namespace {
    const std::string end("end");
}
void f() { std::string x; while (x != end) {...} }

```

And once again [NeverUseStaticLocalVariables](#)

NeverUseStaticLocalVariables

Never do this:

```

void f() {
    static int x = 10;
}

```

Static on a local variable means the compiler is supposed initialize it the first time the function is entered, but it holds its value on subsequent calls. It's sometimes used for local counters, or in the "Myers Singleton" approach to singletons.

The problem is that the C++ standard does not require compilers to make this initialization thread safe, and almost none do. So in a multi-threaded program if there are concurrent first calls to `f` there will be a disaster. Using this for singletons is particularly prone to multi-threaded collisions.

So use the less elegant but safer options: make the variable a class member for member functions or a file-private global for non-member functions.

NoUsingNamespacelnHeaders

Don't use the `{using namespace ...}` or `using ...` constructs in a header file. It might save you some typing but it also forces the namespaces you use onto every `.cpp` file that directly or indirectly includes your headers. That could create name clashes with names in some other namespace that the author of the `.cpp` file wants to use. Use fully qualified names, painful as it might be.

There is one exception. It's sometimes handy to "import" names from one namespace to another. For example suppose some C++ compiler doesn't provide the `std::tr1::auto_ptr` template, which is used in `qpid`. [Boost](#) does provide a compatible `boost::auto_ptr` but it's in the `boost` namespace and `qpid` expects it in `std::tr1`. No problem, we create our own `tr1/memory` header file:

```

#include <boost/memory>
namespace std {
    namespace tr1 {
        using boost::auto_ptr;
    }
}

```

This makes the `boost` template available in the standard namespace. (Actually you don't need to do this yourself, `boost` provides a set of adapter headers for all the `tr1` stuff.)

PrivateLocking

The only way to write thread safe code without losing your mind is to keep your synchronisation simple and small. You cannot test for thread safety. Really you can't. If synchronization is complicated or spread out it's pretty much impossible to know by inspection whether it's correct.

A key technique is to encapsulate synchronization *inside* thread-safe classes. Every public member function should protect *itself* from concurrent access by using private locks or other synchronization objects. You can verify the synchronization of just that class in isolation. It's much easier to build complicated thread-safe code from simple pieces that you know to be individually thread-safe.

It's very dangerous to provide public access to locks because now to establish thread safety for a class you have to inspect *every potential* use of that class. Not to mention every change to or addition of code using the class. Did I mention losing your mind?

ReturnStdStringByValue

Don't do this:

```
std::string& f();
const std::string& g(); // Not much better
```

Instead do this:

```
std::string f();
std::string g();
```

`std::string` is designed expressly to allow you to treat strings as simple pass-by-value types, like `int`. It's efficient to return by value rather than reference and it avoids core dumps if the real string hidden away in `f` gets deleted before the reference. In particular it allows `f()` to compute once-off values and forget about them, e.g.:

```
std::string hello(const std::string& name) { return "hello " + name; }
```

With the "&" style return this would be an immediate disaster as the returned reference is invalid before the caller even gets it! NB. The last example contains another error! See [BewareOfStringPromotion](#).

ScopedLocking

Always use scoped lockers to lock mutexes and the like. Don't do this:

```
lock.acquire();
do_stuff(); // DANGER: lock never released if exception thrown here.
lock.release();
```

Instead use a "scoped locker". This is simply a class that does the "acquire" in its constructor and the "release" in its destructor:

```
Locker locker(lock);
do_stuff();
```

Not only does this save a bit of typing, it guarantees that the lock will be released even if an exception is thrown, because C++ guarantees to call destructors of all local variables on exit from a scope. This also protects you forgetting to release the lock at every exit point from a function with multiple exit points - the compiler takes care of it for you. This technique applies more generally to any situation where you have to acquire and release resources. `std::auto_ptr` is a similar tool for memory management.

SharedPtr

`std::tr1::shared_ptr` is an almost-standard smart pointer template that provides unintrusive reference-counting semantics for any class. It almost makes memory management too easy for a C++ programmer.

It's available in g++ and some other compilers by default. There are several open source implementations if we ever port to a compiler that doesn't have it.

The golden rule: if class Foo has shared ownership then never ever write `Foo*`. Anywhere. Ever. Use `shared_ptr` in all function signatures and variables, use `std::tr1::dynamic_pointer_cast` and friends for casting.

Qpid will use it for all classes with shared ownership semantics, enforced by private constructors and static factory functions. We'll also adopt the convention to typedef `shared_ptr` within the class for convenience. E.g.

```
class Foo {
    Foo() { ... }

public:
    typedef std::tr1::shared_ptr<Foo> shared_ptr;
    static shared_ptr create() { return new Foo(); }
    // .. a create for each constructor.
}

Foo::shared_ptr p = Foo::create(); // etc...
```

There's a good article at http://www.boost.org/libs/shared_ptr/sp_techniques.html.

ValgrindBadSuppressions

(Observed with valgrind 3.2.1, fixed in 3.2.3)

Valgrind 3.2.1 sometimes produces suppressions that don't work, like this:

```
{
    <insert a suppression name here>
    Memcheck:Free
    fun:_vgrZU_libcZdsoZa_free
    fun:main
}
```

The identifying characteristic is that they begin with `fun:_vg<something>`

<http://article.gmane.org/gmane.comp.debugging.valgrind/5939> explains what these symbols are.

The workaround is to replace `fun:_vg*_lib*<something>` with `fun:<something>`, where something should be a valid C or C++ mangled symbol.

The following test program demonstrates the problem

```
#include <stdlib.h>
int main(int argc,char**argv) {
    char*p = malloc(10);
    free(p);
    free(p);
}
```

Index

Apache Qpid: Open Source AMQP Messaging

Enterprise Messaging systems let programs communicate by exchanging messages, much as people communicate by exchanging email. Unlike email, enterprise messaging systems provide guaranteed delivery, speed, security, and freedom from spam. Until recently, there was no open standard for Enterprise Messaging systems, so programmers either wrote their own, or used expensive proprietary systems.

AMQP Advanced Message Queuing Protocol is the first open standard for Enterprise Messaging. It is designed to support messaging for just about any distributed or business application. Routing can be configured flexibly, easily supporting common messaging paradigms like point-to-point, fanout, publish-subscribe, and request-response.

Apache Qpid implements the latest AMQP specification, providing transaction management, queuing, distribution, security, management, clustering, federation and heterogeneous multi-platform support and a lot more. And Apache Qpid is extremely fast. Apache Qpid [aims to be 100% AMQP Compliant](#).

AMQP Messaging Brokers

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. See [Download](#) to see which messaging clients work with each broker.

AMQP Client APIs: C++, Java, JMS, Ruby, Python, and C#

Qpid provides AMQP Client APIs for the following languages:

- C++
- Java, fully conformant with JMS 1.1
- C# .NET, 0-10 using WCF
- Ruby
- Python

Operating Systems and Platforms:

The Qpid C++ broker runs on the following operating systems:

- Linux systems
- Windows
- Solaris (coming soon)

The Qpid Java broker runs on:

- Any Java platform

Qpid clients can be run on the following operating systems and platforms:

- Java:
 - any platform, production proven on Windows, Linux, Solaris
- C++:
 - Linux
 - Windows
 - Solaris (coming soon)
- C#
 - .NET

Getting Started

- Download Qpid here: [download page](#)
- Follow these instructions to get started fast: [Getting Started](#)
- The Qpid Management Framework: [QMF](#)
- Read the Documentation: [Documentation](#)
- If you need help, [mail the lists](#)

Getting Help

If you have a question about any aspect of Qpid or need help getting up and running please send an email to [one of our mailing lists](#).

Getting Involved

We welcome contributions to Qpid. [Mail us](#) on one of our lists if you want to contribute to the project, have questions on using it or just want to get our thoughts on a topic...

Roadmap

For details on releases, a summary of what is in each release can be found here [roadmap](#)

0.6 Feature Matrix

- 1. Related Pages
- 2. Table Key
- 3. Broker Features
- 4. Client Features



Incomplete

This table is a work-in-progress and should be considered neither complete nor correct at this point.



Join In

I have started a thread about this table on the dev mailing list - "0.6 Feature Matrix". Let us know what you think!



Interoperability Not Implied

This table does not imply interoperability across broker implementations. For instance, if the Java and C++ brokers both support clustering, the ability to use clustering from a Java client to a C++ broker is not implied here. Interoperability is covered in a separate 0.6 Interoperability Matrix page.

1. Related Pages

[0.6 Feature Descriptions](#)
[0.6 Interoperability Matrix](#)

2. Table Key

Y : Supported
N : Not Supported
P : Planned
P_{X,Y} : Planned for release X.Y
? : Unknown: may/should work, but not tested
- : Not applicable

3. Broker Features

	C++		Java
	Linux	Windows	*
Protocols			
AMQP 0-8	N	N	Y
AMQP 0-9	N	N	Y
AMQP 0-9-1	N	N	Y
AMQP 0-10	Y	Y	Y
AMQP 1-0	P	P	P
Producer Flow Control	Y	Y	Y⁴
Transactions	Y	Y	Y
Distributed Transactions	Y	Y³	
SSL	Y	P	
RDMA	Y	N	N
Broker Features			
Access Control List (ACL)	Y	P	Y
Clustering	Y	N	N
Federation	Y	Y	P
Management Exchange	Y	Y	Y²
QMF Agent	Y	Y	Y²
JMX Management Console	N	N	Y
QMan	N	N	Y
Selectors	N	N	Y
Replication	Y	Y	
Watchdog	Y		

XML Exchange	Y		
Last Value Queue	Y	Y	P
Priority Queue	P	P	Y
SASL Security	Y	Y	Y
BDB Store Module	N	N	Y
SQL Database Store Module	N	Y	Y
Async Store Module	Y	N	N
Durable Exchanges	Y ¹	Y ¹	Y ¹
Durable Queues	Y ¹	Y ¹	Y ¹
Durable Bindings	Y ¹	Y ¹	Y ¹
Queue Sizing Policies	Y		
Flow-to-disk	Y ¹		N
External Tools			
qpid-config	Y	Y	
qpid-tool	Y	Y	
qpid-cluster	Y	N	
qpid-route	Y	Y	
qpid-stat	Y	Y	

Notes

1. When a store module is loaded
2. via Qman
3. Not persistent at this time on SQL Database store
4. On 0-8, 0-9 and 0-9-1 only at this time

4. Client Features

	C++		JMS	Java	Python	Ruby	WCF
	Linux	Windows	*	*	*	*	Windows
Protocols							
AMQP 0-8	N	N	Y		?	?	N
AMQP 0-9	N	N	Y		?	?	N
AMQP 0-10	Y	Y	Y		Y	Y	Y
AMQP 1-0	P	P	P		P	P	P
Client Features							
New Messaging API	Y	Y			Y	P	
New QMF API							
Priority Delivery							

0.6 Feature Descriptions

- 1. Related Pages
- 2. Protocol Features
 - 2.1. AMQP
 - 2.2. Producer Flow Control
 - 2.3. Transactions
 - 2.4. Distributed Transactions
 - 2.5. SSL
 - 2.6. RDMA
- 3. Broker Features
 - 3.1. Access Control Lists (ACL)
 - 3.2. Clustering
 - 3.3. Federation
 - 3.4. QMF Management Exchange
 - 3.5. QMF Agent
 - 3.6. JMX Management Console
 - 3.7. QMan
 - 3.8. Selectors
 - 3.9. Replication
 - 3.10. Watchdog
 - 3.11. XML Exchange
 - 3.12. Last Value Queue (LVQ)
 - 3.13. Priority Queue
 - 3.14. SASL Security
 - 3.15. BDB Store Module

- 3.16. SQL Database Store Module
- 3.17. Async Store Module
- 3.18. Durable Exchanges
- 3.19. Durable Queues
- 3.20. Durable Bindings
- 3.21. Queue Sizing Policies
- 3.22. Flow-to-disk
- 4. Client Features
 - 4.1. New Messaging API
 - 4.2. New QMF API
 - 4.3. Priority Delivery
- 5. External Tools
 - 5.1. qpid-config
 - 5.2. qpid-tool
 - 5.3. qpid-cluster
 - 5.4. qpid-route
 - 5.5. qpid-stat



Incomplete

This page is a work-in-progress and should be considered neither complete nor correct at this point.



Join In

I have started a thread about this table on the dev mailing list - "0.6 Feature Matrix". Let us know what you think!

1. Related Pages

[0.6 Feature Matrix](#)
[0.6 Interoperability Matrix](#)

2. Protocol Features

2.1. AMQP

Advanced Message Queuing Protocol (AMQP) is an Open Messaging Middleware standard upon which Qpid's wire protocol is based. The standard is maintained by the [AMQP Working Group](#). Currently the following versions of the protocol exist:

- 0-8, released June 2006. This is the first working version of AMQP.
- 0-9, released December 2006. This version improves reliability aspects of the protocol.
- 0-9-1, released November 2008 (after 0-10).
- 0-10, released February 2008.
- 1-0, **not yet released**, is slated to be the first stable release of AMQP. Version 1-0 is currently in final draft (PR2).

2.2. Producer Flow Control

The broker will throttle (reduce) the rate at which clients can publish messages if the broker starts to run low on resources or if queue size policies dictate.

For a detailed discussion, see the following pages:

- Producer flow control
- Java Broker - AMQP0-9 Tactical Producer Flow Control
- QPID-942.

2.3. Transactions

Local one-phase commit (1PC) transactions ensure atomicity over a number of otherwise disconnected actions on the broker (such as publishing or consuming a number of messages in a group). For local transactions, the broker creates an internal transaction ID and uses it to track the state of the transaction. The client must either commit or abort the transaction to close it.

2.4. Distributed Transactions

Distributed two-phase commit (2PC) transactions ensure atomicity over a number of otherwise disconnected actions on a distributed system (which involve two or more brokers and/or clients). This operation is usually coordinated by an external transaction monitor which creates transaction IDs and controls the state of the transaction.

For a detailed discussion, see the following resources:
[The AMQP Distributed Transaction Classes \(Java\)](#)

2.5. SSL

SSL allows IP communications between the broker(s) and client(s) to be encrypted.

For a detailed discussion, see the following resources:

[SSL](#)

Configure Java Qpid to use a SSL connection.

[JMX SSL Configuration](#)

[FAQ](#)

2.6. RDMA

Remote Direct Memory Access (RDMA) permits high-throughput, low-latency communications between broker(s) and client(s). Among its features is zero-copy networking in which the network hardware copies networked data directly to the application memory space without the use of operating system buffers. RDMA implementations include Infiniband and iWarp (which uses RDMA over TCP).

3. Broker Features

3.1. Access Control Lists (ACL)

Security mechanism by which users may be granted permissions to perform the various operations on a broker from a client.

For a detailed discussion, see the following resources:

[Qpid Design - Access Control Lists](#)

[Qpid ACLs](#)

[ACL](#) (v2 ACL format specification, used on both Java and C++ brokers)

[Java XML ACLs](#) (**deprecated**: v1 XML-based ACL format specification used on M4 release Java broker)

3.2. Clustering

Module which allows several brokers to form an active-active or high availability (HA) cluster, primarily for reliability.

For a detailed discussion, see the following resources:

[Starting a cluster](#)

[Persistent Cluster Restart Design Note](#)

[Old Clustering Design Note](#)

3.3. Federation

Mechanism by which brokers can be connected primarily for the purpose of sharing load and providing broker-broker connectivity.

For a detailed discussion, see the following resources:

[Using Broker Federation](#)

[Federation Design Note](#)

3.4. QMF Management Exchange

QMF is a general-purpose management bus built using Qpid. Qpid itself may be managed using this facility. The Management Exchange is the component loaded by the broker to enable management functionality on a broker.

For a detailed discussion, see the following resources:

[Qpid Management Framework](#)

[Qpid Management Features](#)

3.5. QMF Agent

The QMF agent is the component which is embedded into the managed entities, and provides QMF awareness to that entity.

For a detailed discussion, see the following resource:

[Qpid Management Framework](#)

3.6. JMX Management Console

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

For a detailed discussion, see the following resources:

[Qpid JMX Management Console](#)

[JConsole](#)

[Qpid JMX Management Console User Guide](#)

[Qpid JMX Management Console Testing Guide](#)

3.7. QMan

QMan is a management bridge for Qpid. It exposes the broker's QMF management interfaces using Java Management Extensions (JMX) and / or OASIS Web Services Distributed Management (WSDM).

For a detailed discussion, see the following resource:
[QMan - Qpid Management bridge](#)

3.8. Selectors

The ability to filter the messages browsed or consumed from a queue. The filter is limited to message header properties.

3.9. Replication

Asynchronous replication of queue state through the use of events on a secondary broker.

For a detailed discussion, see the following resource:
[queue state replication](#)

3.10. Watchdog

The watchdog plug-in will kill the qpid broker process if it becomes stuck for longer than a configured interval.

3.11. XML Exchange

A plug-in exchange which can open messages and run xquery against its XML content in order to determine routing to the appropriate queue.

3.12. Last Value Queue (LVQ)

A queue in which the content is maintained as key-value pairs. Publishing to a LVQ updates the value against its key; consuming a message for a particular key allows the last value to be read. The key/value pair may or may not be consumed, depending on options.

For a detailed discussion, see the following resources:
[LVQ](#)
[LVQ Example](#)

3.13. Priority Queue

Queues in which the delivery order is determined primarily by the priority of the message, and secondarily by the order of arrival.

For a detailed discussion, see the following resources:
[Use Priority Queues](#)
[Qpid Design - Queue Implementation](#) - see section on Priority Queues

3.14. SASL Security

Simple Authentication and Security Layer - an industry standard framework for authentication, and implemented in Qpid.

For a detailed discussion, see the following resources:
[Qpid Design - Authentication](#)
[Qpid Interoperability Documentation](#) - Authentication mechanism interoperability

3.15. BDB Store Module

An implementation of a persistence store using Oracle Berkeley Database (BDB) which provides persistence to exchanges and queues and their configurations, and to the messages on these queues. Exchanges, queues and messages must be set to be persistent before they will be persisted. In addition, only persistent queues may store persistent messages.

For a detailed discussion, see the following resource:
[MessageStore Tool](#)

3.16. SQL Database Store Module

An implementation of a persistence store using a QSQL database which provides persistence to exchanges and queues and their configurations, and to the messages on these queues. Exchanges, queues and messages must be set to be persistent before they will be persisted. In addition, only persistent queues may store persistent messages.

3.17. Async Store Module

A Linux-only implementation of a persistence store using a combination of BDB (for exchange and queue configuration) and a custom-written

asynchronous store (for message content and transactions). This store is capable of writing messages to disk at high rates through the use of DMA. Exchanges, queues and messages must be set to be persistent before they will be persisted. In addition, only persistent queues may store persistent messages.

3.18. Durable Exchanges

Exchanges and their configuration are persisted so that they do not need to be recreated on recovery or on startup of a previously running broker where they were present. The exchange must be set to be persistent and there must be a store module loaded for this persistence to be active.

3.19. Durable Queues

Queues and their configuration are persisted so that they do not need to be recreated on recovery or on startup of a previously running broker where they were present. The queue must be set to be persistent and there must be a store module loaded for this persistence to be active. Note also that only persistent queues can store persistent messages and recover them at recovery/startup.

3.20. Durable Bindings

Bindings and their configuration are persisted so that they do not need to be recreated on recovery or on startup of a previously running broker where they were present. The exchange and the queue being bound must be set to be persistent and there must be a store module loaded for this persistence to be active.

3.21. Queue Sizing Policies

The content of queues may be limited by number and/or cumulative message size. When these limits are exceeded, the queue may manage the situation by (among others) refusing to accept new messages, throttling message production, or flowing the messages to disk (see [Flow-to-disk](#) below).

3.22. Flow-to-disk

Flow-to-disk is one of the mechanisms for handling queue size policy violations. This mechanism allows all messages which exceed a queue size policy to be written to disk (whether persistent or not), and the message content is released from memory. To consume the message, however, the message must first be read from the store to restore its content to the queue.

For a detailed discussion, see the following resources:

[Java Broker Design - Flow to Disk](#)
[FtD Code Review Notes](#)

4. Client Features

4.1. New Messaging API

A new consistent set of client messaging APIs which do not require an in-depth knowledge of AMQP, but focus instead on generic messaging tasks such as sending and receiving messages.

4.2. New QMF API

Built on top of the [New Messaging API](#), this new QMF API simplifies the use of QMF, and uses a work-queue based event model.

For a detailed discussion, see the following resource:
[QMFv2 API Proposal](#)

4.3. Priority Delivery

A client can change the priority model and/or level used by the broker to deliver messages (see [3.12. Priority Queue](#) above).

5. External Tools

5.1. qpid-config

A command-line tool to create, delete and configure queues, exchanges and bindings on a broker.

For a detailed discussion, see the following resource:
[Management Tools Overview](#)

5.2. qpid-tool

A telnet type tool to access QMF data, view QMF management schemas, issue commands and QMF resources.

For a detailed discussion, see the following resource:
[Management Tools Overview](#)

5.3. qpid-cluster

For a detailed discussion, see the following resource:
[Management Tools Overview](#)

5.4. qpid-route

A command-line tool to configure broker federation routes. This tool is used to establish a broker federation.

For a detailed discussion, see the following resource:
[Management Tools Overview](#)

5.5. qpid-stat

A command-line tool which shows information on brokers, connections, exchanges and queues.

For a detailed discussion, see the following resource:
[Management Tools Overview](#)

0.6 Interoperability Matrix

- [1. Related Pages](#)
- [2. AMQP Interoperability](#)
- [3. Feature Interoperability](#)



Incomplete

This table is a work-in-progress and should be considered neither complete nor correct at this point.



Join In

I have started a thread about this table on the dev mailing list - "0.6 Feature Matrix". Let us know what you think!

1. Related Pages

[0.6 Feature Matrix](#)
[0.6 Feature Descriptions](#)

2. AMQP Interoperability

			Brokers		
			C++	Java	
			Linux	Windows	*
Clients	C++	Linux	0-10	0-10	0-10
		Windows	0-10	0-10	0-10
	JMS	*	0-10	0-10	0-8 0-9 0-9-1 0-10
		*	0-10	0-10	0-8 0-9 0-9-1 0-10
	Python	*	0-10	0-10	0-8 0-9 0-9-1 0-10
		*	0-10	0-10	0-8 0-9 0-9-1 0-10
	Ruby	*	0-10	0-10	0-8 0-9 0-9-1 0-10
	WCF	Windows	0-10	0-10	0-10

3. Feature Interoperability



Feature Interoperability

We need a matrix which addresses feature interoperability across implementations. For instance, if the Java broker has clustering, will the JMS client support cluster failover on a C++ broker? (and vice versa). What about Python and Ruby?

AMQP0-9-DesignNotes

Design notes for AMQP 0-9 implemenation

- Reserved field in Request frame must be set to 0.
- Request and Response constants were added to amqp.0-9.xml
- Request ID and Response ID must start at 1 for new channels. 0 is reserved for future use, and should not be used in normal interactions between client and server.
- Response Mark must start at 0 for new channels.
- Content class encoding: For inline messages (first byte = 0), a null or empty byte array may be used.
- Content class encoding: For refs, (first byte = 1), an error or exception must be thrown if the byte array is either null or empty. It makes no sense to send a null ref.
- Content class decoding: For inline messages (first byte = 0), it is not possible to discriminate between the null array or empty array case above that encoded it. Decode as an empty byte array, not a null. (open for discussion)
- Content class: It may be possible to set a value for either/or null and empty values in the future - if a use-case can be made for it
- Possible batch-handling modes should be decided upon.
- TODO: Devise a mechanism to allow one-way requests, where no acknowledgements are sent.

AMQP 0-9 Specification Issues

- Errata will be made by adding to an amqp-errata.0-9.xml file rather than by making edits directly to the specification file. These are the advantages:
 - The differences between the current specification and the spec as we use it are readily apparent.
 - Different implementations share the same specification file. Thus errors that may arise as a result of a change required for one implementation (e.g. Java) on others (e.g. C++) are controlled/eliminated.
- Two constants are missing and need to be inserted as an erratum:

```
<constant name = "frame-request"      value = "9"  />
<constant name = "frame-response"    value = "10" />
```

- The Basic field `Basic.type` (a shortstr) was omitted from `Message.transfer`. However, after some discussion it was resolved that since this field serves JMS messaging only, that it should be handled as a custom property rather than creating an XML erratum to insert it. The property name is "`JMSXType`".
- The Basic field `Basic.mandatory` (a bit) was originally omitted from `Message.transfer` because its functionality would have been handled by the availability of dead-letter queues. However, they did not make it into the AMQP 0-9 specification. Thus, `Basic.mandatory` has been temporarily added as the last field in `Message.transfer` until dead-letter queues become a reality in the specification.

AMQP (Advanced Message Queueing Protocol)

What is AMQP?

AMQP Advanced Message Queuing Protocol is an open standard designed to support reliable, high-performance messaging over the Internet. AMQP can be used for any distributed or business application, and supports common messaging paradigms like point-to-point, fanout, publish-subscribe, and request-response.

Apache Qpid implements AMQP, including transaction management, queuing, clustering, federation, security, management and multi-platform support.

Apache Qpid implements the latest AMQP specification, providing transaction management, queuing, distribution, security, management, clustering, federation and heterogeneous multi-platform support and a lot more.

Apache Qpid is highly optimized, and aims to be 100% AMQP Compliant.

Download the AMQP Specifications

AMQP version 0-10

- AMQP 0-10 Specification (PDF)
- AMQP 0-10 Protocol Definition XML
- AMQP 0-10 Protocol Definition DTD

AMQP version 0-9-1

- AMQP 0-9-1 Specification (PDF)
- AMQP 0-9-1 Protocol Documentation (PDF)
- AMQP 0-9-1 Protocol Definitions (XML)

AMQP version 0-9

- AMQP 0-9 Specification (PDF)
- AMQP 0-9 Protocol Documentation (PDF)
- AMQP 0-9 Protocol Definitions (XML)

AMQP version 0-8

- AMQP 0-8 Specification (PDF)
- AMQP 0-8 Protocol Documentation (PDF)
- AMQP 0-8 Protocol Definitions (XML)

AMQP Brokers

- AMQP Messaging Broker (implemented in C++)
- AMQP Messaging Broker (implemented in Java)

AMQP Messaging Broker (implemented in C++)

Running the AMQP Messaging Broker

- Running an AMQP 0-10 C++ broker
- Configuring Queue Options
- Configuring Exchange Options
- Using Broker Federation
- How to use SSL
- Understanding Last Value Queues (LVQ)
- Queue State Replication
- Getting Started
- Starting a cluster
- Understanding Access Control Lists

Management

- Managing the C++ Broker
- QMan - Qpid Management bridge
- Qpid Management Framework
- Qpid Management Framework (QMF) Protocol
- Manage anything with Qpid - QMF Python Console Tutorial

AMQP Messaging Broker (implemented in Java)

General User Guides

- Feature Guide
- FAQ
- Getting Started Guide
- Broker Environment Variables
- Troubleshooting Guide

How Tos

- Add New Users
- Configure ACLs
- Configure Java Qpid to use a SSL connection.
- Configure Log4j CompositeRolling Appender
- Configure the Broker via config.xml
- Configure the Virtual Hosts via virtualhosts.xml
- Debug using log4j
- How to Tune M3 Java Broker Performance
- Qpid Java Build How To
- Use Priority Queues

Management Tools

- Qpid JMX Management Console
- MessageStore Tool
- Qpid Java Broker Management CLI
- Management Design notes

AMQP Messaging Clients

AMQP Java JMS Messaging Client
AMQP C++ Messaging Client
AMQP .NET Messaging Client
AMQP Python Messaging Client
AMQP Ruby Messaging Client

AMQP .NET Messaging Client

Currently the .NET code base provides two client libraries that are compatible respectively with AMQP 0.8 and 0.10. The 0.8 client is located in qpid\dotnet and the 0.10 client in: qpid\dotnet\client-010

You will need an AMQP broker to fully use those client libraries. Use M4 or later C++ broker for AMQP 0.10 or Java broker for AMQP 0.8/0.9.

User Guides

- [.NET client user guide](#)
- [.NET client Excel plug-in](#)
- [The WCF interface for the .NET client](#)

Examples

- [.NET AMQP Messaging Client Examples](#)

Developer Guidelines

- [Qpid Developer Documentation](#)
- [\[Coding Standards\]](#)
- [How Tos](#)
 - [Build .NET Client](#)
 - [Releasing](#)
 - [Run tests](#)
 - [Setup .Net Client on Windows](#)

AMQP C++ Messaging Client

User Guides

- [C++ Client API \(AMQP 0-10\)](#)
- [Client configuration](#)

Examples

- [Examples](#)
- [Running the C++ Examples](#)

Client configuration

There are several environment variables that affect the Qpid library in qpid client programs.
They are similar to configuration options for the qpid broker.

Loadable Modules

By default a qpid client loads modules from a default directory, the exact location depends on your system.
`qpidd --help` will show you the default directory for broker modules. If the broker directory is `<path>/daemon`, then the client directory is `<path>/client`.

The following environment variables modify how modules are loaded:

`QPID_MODULE_DIR`: Load modules in this directory instead of the default directory.
`QPID_LOAD_MODULE`: Load this additional module.
`QPID_NO_MODULE`: Don't load modules from the default directory.

Logging

The client recognizes the same logging options as the broker as environment variables. `qpidd --help` will show you the logging options available.

AMQP Java JMS Messaging Client

The Java Client supported by Qpid implements the JMS 1.1 specification.

General User Guides

- [AMQP Java JMS Client Feature Guide]
- [FAQ](#)
- Java JMS 1.1 Specification
- System Properties
- Connection URL Format - The format used to describe a connection.
- BindingURLFormat - The format used for creating bindings within and to a broker.
- [How to Use JNDI to configure the AMQP Java JMS Client]
- [Using the AMQP Java JMS Client with RT Java]
- [AMQP Java JMS Client Tuning Guide]

AMQP Java JMS Examples

- [AMQP Java JMS Examples](#)
- [Script for Running the Examples](#)
- [README for the above script](#)

AMQP Python Messaging Client

User Guides

- [Python Client API Guide](#)

Examples

- [AMQP Python Client Examples](#)
- [Running the AMQP Python Client Examples](#)

Test Framework

- [Python Test Framework](#)

AMQP Ruby Messaging Client

The Ruby Messaging Client currently has little documentation and few examples.

Examples

*[AMQP Ruby Messaging Client Examples](#)

AMQP Test Suites for Apache Qpid

C++ vs. Python API Discrepancies

This page compares C++ and Python code samples from our examples, looking for arbitrary discrepancies in the API. I suggest that we add proposals to fix these discrepancies inline for each example.

Opening and closing connections and sessions

C++

```

Connection connection;
try {
    connection.open(host, port);
    Session session = connection.newSession();
    ...
    connection.close();
    return 0;
} catch(const std::exception& error) {
    std::cout << error.what() << std::endl;
}
return 1;

```

Python

```

host="127.0.0.1"
port=5672
user="guest"
password="guest"

socket = connect(host, port)
connection = Connection (sock=socket, username=user, password=password)
connection.start()
session = connection.session(str(uuid4()))
...
session.close(timeout=10)

```

Discrepancies

- Python uses a socket object, not needed in the C++ API. I suggest that Python follow C++ here.
- Python requires the user to provide a UUID instead of creating one for him. I suggest that Python provide the UUID automagically, as does C++.
- Our Python examples end by closing the session, our C++ examples end by closing the connection. If both APIs allow both approaches, we should fix the examples; if not, we should make the APIs consistent.

Subscribing to a Queue

C++

```

LocalQueue local_queue;
SubscriptionManager subscriptions(session);
subscriptions.subscribe(local_queue, string("message_queue"));
subscriptions.run();

```

Python

```

local_queue_name = "local_queue"
queue = session.incoming(local_queue_name)
queue.message_subscribe(queue="message_queue", destination=local_queue_name)
queue.start()

```

Discrepancies

- C++ local queues are created as standalone objects, then subscribed. Python retrieves a local queue from the server using the `session.incoming()` method.
- Python starts delivery using the local queue's `start()` method. C++ begins delivery using the `SubscriptionManger.run()` or `Session.run()` method.
- Naming: Python calls the method "message_subscribe", which corresponds to the AMQP name, but it's odd naming: it does not subscribe to a message, nor does it subscribe a message to anything else.

Setting Delivery Properties, Message Properties

C++

```
message.getDeliveryProperties().setRoutingKey("routing_key");
```

Python

```
delivery_props = session.delivery_properties(routing_key="routing_key")
```

Discrepancies

- Python requires the programmer to create the delivery property object for a new message using a session object. That seems strange - it would be better to create it using the message, since it pertains to a message's delivery properties.
- This same discrepancy affects message properties

SubscriptionManager

Discrepancies

- Python does not have a SubscriptionManager. This would be very useful once Python supports async mode.

Message Listeners

C++

```
// A message listener:  
  
class Listener : public MessageListener{  
private:  
    SubscriptionManager& subscriptions;  
public:  
    Listener(SubscriptionManager& subscriptions);  
    virtual void received(Message& message);  
};  
  
void Listener::received(Message& message) {  
    std::cout << "Message: " << message.getData() << std::endl;  
    if (endCondition(message)) {  
        subscriptions.cancel(message.getDestination());  
    }  
}  
  
// Using a message listener with a subscription manager:  
  
SubscriptionManager subscriptions(session);  
  
Listener listener(subscriptions);  
subscriptions.subscribe(listener, "message_queue");  
subscriptions.run();
```

Python

```

----- Message Receive Handler -----
class Receiver:
    def __init__(self):
        self.finalReceived = False

    def isFinal(self):
        return self.finalReceived

    def Handler(self, message):
        content = message.body
        session.message_accept(RangedSet(message.id))
        print content
        if content == "That's all, folks!":
            self.finalReceived = True

    # Call message_subscribe() to tell the broker to deliver messages
    # from the AMQP queue to this local client queue. The broker will
    # start delivering messages as soon as message_subscribe() is called.

    session.message_subscribe(queue="message_queue", destination=local_queue_name)
    queue.start()

    # Register a message listener with the queue

    receiver = Receiver()
    queue.listen(receiver.Handler)

    while not receiver.isFinal():
        sleep(1)

```

Discrepancies

- In Python, a message handler is registered with a local queue, and the local queue is subscribed to the remote queue. In C++, a message listener is subscribed to using a Session or a Subscription Manager. I prefer the C++ approach here.

Synchronous / Asynchronous Modes

Discrepancies

- C++ supports both synchronous and asynchronous modes. Python supports only synchronous mode. Python should support both.

Getting and Setting Message Contents

C++

```

// getData()

std::cout << "Response: " << message.getData() << std::endl;

// setData()

message.setData("That's all, folks!");

// appendData()

message.appendData(" ... let's add a bit more ...");

```

Python

```

message = queue.get(timeout=10)
content = message.body

```

Discrepancies

- C++ calls this "message data" and accesses it via methods, Python calls it the message body and provides direct access. They are completely different

Other issues

- Both languages should support streaming data into and out of messages
- The C++ interface has consistently confused people who work with binary data

Getting and Setting Application Headers

C++

```
message.getHeaders().getString("control");

message.getHeaders().setString("control", "continue");
```

Python

```
message_properties = message.get("message_properties")
message_properties.application_headers["control"] = "continue"
```

Discrepancies

- Different data model. C++ has a message, which has headers. Python has a message, which has message properties, including application headers.
- Python supports via a dictionary, which is very nice. Can I do this in C++?

Clustering/HA

Definitions

There are two very different reasons to cluster:

- Reliability/Fault Tolerance
 - Cluster members replicate state.
 - If one member fails, clients can fail-over to another.
- Scalability/throughput/load balancing:
 - Distribute large work load across multiple brokers for higher throughput.

It's important not to confuse the two goals. Note that a reliable cluster will be LESS scalable and performant than even a single broker - replication is extra work on top of normal processing. There is also:

- Federation (a set of distributed exchanges and queues, separately managed and wired together)

It's not clear where to draw the line between federation and clustering for scalability.

Reliability clustering is orthogonal to scalability clustering/federation, which means they can be combined. Just replace the individual brokers in your federation or scalability cluster with reliable broker clusters and you have a reliable and scalable system.

Requirements/Use Cases

- Reliability Requirements
- ClusteringAndFederation

Design notes

- Cluster Design Note - Cluster for reliability. A reliable broker cluster can participate as a single broker in federation or throughput clusters.
- Persistent Cluster Restart Design Note - re-starting a cluster with persistent members.

- Cluster Failover Modes - how a cluster and its clients deal with failures.
- AMQP breakdown for clustering - Analysis of AMQP 0-10 commands and their effect on replicated state in a cluster.
- Federation Design Note - Discussion of what has been done to date in C++
- Java Federation Design Proposal - Discussion of what could be implemented in the Java Broker.

Related reading

- AMQP specification, chapter 3 "Sessions" and session class documentation in chapter 9.
<http://jira.amqp.org/confluence/download/attachments/720900/amqp.0-10.pdf?version=1>
- openais.org docs on Closed Process Group (CPG), cpg man pages in openais install.

AMQP breakdown for clustering

AmqpBreakdown

AmqpBreakdown

Breakdown of AMQP 0-10 commands and controls for clustering.

Definitions

Replica: broker member of the cluster.

Connected replica: replica directly connected to the client.

Connected session: Attached session on the connected replica.

Shadow session: Backup of active session state on non-connected replica.

Detached session: Session not attached to any replica, awaiting timeout.

Replicate: Multicast to cluster, defer completing some action till cluster responds.

Inferred: Change on the connected replica that can be inferred by other replicas because it is a deterministic response to some replicated change (e.g. responding to a query command)

Types of state.

- shared state
 - wiring.
 - queue contents.
- session state.
 - command ids.
 - subscriptions.

Cluster qualities of service:

- shared state only: replicate shared state but does not support failover.
- failover: Replicate shared state and sessions state, supports failover.

We examine the impact on cluster state of each AMQP control & command, and the implications for what needs replication.

Connection controls

None of the connection controls MUST be replicated, they do not affect state.

Connection creation/destruction MAY be replicated if using shadow connections to organize sessions.

Session controls

Performance note: Session controls other than completed are not sent on a per-message basis so are not critical path.

Failover requires that all replicas know:

- state of command IDs to correlate completions.
- outgoing commands in doubt for replay.

This means:

- all outgoing commands must be inferred or replicated
- all incoming commands are replicated OR additional command-id info is replicated.

Receive

attach,detach: MUST replicate. Replicas must know all sessions & attachment.

request-timeout: MUST replicate - replicas should respect timeout. See other events below.

command-point, gap, expected: MUST replicate for consistent command numbering.

confirmed: Can be ignored as per spec.

flush: No need to replicate, only attached replica need respond.

known-completed: MUST replicate so replicas can avoid wrap-around. on their unknown-complete set.

completed: Replicas need client completions to bound their replay list. They do not need immediate replication since completions will be re-sent as part of failover. They MUST known of completions before sending a known-complete to client since the client will no longer notify completion of known-complete commands.

So we can do one of

- repliacate all completions (performance risk)
- replicate when sending known-complete (risk memory growth in replay lists)
- combination: replicate sending known-completed and replicate completions when replay list is large.

Send

attached, detached, timeout, command-point, expected, confirmed, flush: No effect on replicated state.

known-completed: See receive completed above.

gap: Not used.

completed: Only need replication for *persistent commands*, i.e. commands that change persistent shared state.

Persistent commands and the async store.

Persistent commands guarantee that once completed their effects are stored persistently and will survive total broker shutdown.

For the strongest guarantee it must be persisted on all persistent replicas in the cluster:

- Connected broker receives command, replicates and initiate async store.
- All persistent replicas initiate async store.
- On async completion, replicas mcast an async store confirmation.
- The connected broker waits for all store confirmations before sending completion.

A more performant implementation with a weaker guarantee would send completed when the local async store completes with no async notifications from the cluster. The risk: client receives completed, local disk is destroyed, rest of cluster shuts down before storing, message is lost. Need to determine if that's an acceptable risk in general, or perhaps offer configurable choice.

Detached session timeouts.

Having each replica indepedently destroy timed-out sessions creates a race: a client could resume a session on ne replica concurrently with the timeout expiring and session state being destroyed on other replicas.

To avoid this we choose an arbitrary cluster member to mcast "session timeout" events when sessions time out. This woudl be the primary in active/passive mode or an arbitrarily chosen member (e.g. the oldest member) in active-active mode.

Execution commands

Receive

sync: No need to replicate, no effect on replicated state. **exception**: MUST replicate, causes destruction of session state. **transfer**: MUST replicate before sending completed.

Send

sync, result, exception: inferred.

Message commands

Receive

Definition: an incoming message transfer is *finished* when:

- accept=none: completed sent for incoming transfer.
- accept=explicit: received accept for the message and sent completed for the accept.

Before a message is finished it can be re-queued in the event of a client disconnect.

transfer: MUST replicate, update queue content.

acquire, release, accept, reject: See dequeue management below.

resume: Not implemented.

subscribe, cancel, set-flow-mode, flow, flush, stop: MUST replicate subscription state for replay.

Send

transfer: MUST replicate for shared state & replay. Need not replicate content if it can be inferred. See "Persistent commands and the async store" above and "Dequeue management" note below.

acquire, release, accept, reject: See dequeue management below.

stop:

Dequeue management.

Enqueue is straightforward: incoming transfers are replicated.

Dequeue provides more options:

- active/passive mode: active replica is "owner" and controls order of enqueue dequeue. Information about dequeues can be delayed/compressed/batched.
- active/active mode, no queue owners: all dequeue information must be replicated.
- active/active mode with queue owners: queues have an owner like active/passive but ownership can be transferred.

Active-active no owners: Replicate all incoming message commands. Replicate dequeue decisions that cannot be inferred. In our current IO-driven model this means all dequeues.

Active/passive: Active broker does dequeues. Can avoid/defer replication till of incoming acquire, release, accept, reject and outgoing transfer till messages are *finished* - i.e. till sending completion for accept or for transfer in implicit-accept mode. At this point it may be possible to batch the events.

Note that events must still be sent even if batched: all replicas need to know which messages were removed from which queues, and the order to replay them in the event of fail-over.

Tx commands

select, commit, rollback: MUST replicate. All replicas must commit/rollback consistently.

Dtx commands

select,start,end,commit,forget,get-timeout,prepare,recover,rollback,set-timeout: MUST replicate so all replicas know which commands are within transactions.

All replicas must join the DTX so all will commit/fail under control of DTX manager.

Exchange commands

declare,delete,bind,unbind: MUST replicate, update wiring.

bound,query: MUST replicate replicas can respond in the event of failover.

Queue commands

declare,delete,purge: MUST replicate, update wiring/queue content. **query:** MUST replicate replicas can respond in the event of failover.

File & Stream commands not implemented.

Cluster Design Note

Reliable Broker Cluster

This document describes cluster design and implementation as of 19 June 2009.

Overview

A *Reliable Broker Cluster* or just *cluster* is a group of brokers collaborating to present the illusion of a single broker with multiple addresses. The cluster is *active-active*, that is to say each member broker maintains the full state of the clustered broker. If any member fails, clients can fail-over to any other member.

New members can be added to a cluster while it is running. An established member volunteers to provide a state update to the new member. Both updaters and updatees queue up cluster activity during the update and process it when the update is complete.

The cluster uses the CPG (Closed Process Group) protocol to replicate state. CPG was part of Open AIS package, it is now part of the corosync package. To avoid confusion with AMQP messages we will refer to CPG multicast messages as *events*.

CPG is a *virtual synchrony* protocol. Members multicast events to the group and CPG ensures that each member receives all the events *in the same sequence*. Since all members get an identical sequence of events, they can all update their state consistently. To achieve consistency, events must be processed in the order that CPG presents them. In particular members wait for their own events to be re-delivered by CPG before acting on them.

Implementation Approach

The cluster implementation is highly decoupled from the broker. There's no cluster-specific code in the general broker, just a few hooks that the cluster uses to modify broker behavior.

The basic idea is that the cluster treats the broker as a black box and assumes that provided it is fed identical input, it will produce identical results. The cluster::Connection class intercepts data arriving for broker Connections, and sends that data as a CPG event. As data events are delivered by CPG, they are fed to the original broker::Connection objects. Thus each member sees all the data arriving at all the members in the same sequence, so we get the same set of declares, enqueues, dequeues etc. happening on each member.

This approach replicates *all* broker state: sessions, connections, consumers, wiring etc. Each broker can have both direct connections and *shadow* connections. A shadow connection represents a connection on another broker in the cluster. Members use shadow connections to simulate the actions of other brokers, so that all members arrive at the same state. Output for shadow connections is just discarded, brokers only send data to their directly-connected clients.

This approach assumes that the behavior of the broker is *deterministic*, that it is completely determined by the input data fed to the broker. There are a number of cases where this does not hold and the cluster has to take steps to ensure consistency:

- Allocating messages: the stand-alone broker allocates messages based on the writability of client connections.
- Client connection disconnects.
- Timers: any action triggered by a timer may happen at an unpredictable point with respect to CPG events.

Allocating messages

The cluster allocates messages to consumers using CPG events rather than writability of client connections. A cluster connection that has potentially got data to write sends a *do-output* event to itself, allowing it to dequeue N messages. The messages are not actually dequeued until the do-output event is re-delivered in sequence with other events. The value of N is dynamically estimated in an attempt to match it to the rate of writing messages to directly connected clients. All the other members have a shadow connection which allows them to de-queue the same set of messages as the directly connected member.

Client disconnects

When a client disconnects, the directly-connected broker sends a deliver-close event via CPG. It does not actually destroy the connection till that message is re-delivered. This ensures that the direct connection and all the shadows are destroyed at the same point in the event sequence.

Actions initiated by a timer

The cluster needs to do some extra work at any points where the broker takes action based on a timer (e.g. message expiry, management, producer flow control) See the source code for details of how each is handled.

Error Handling

There are two types of recoverable error

- *Predictable* errors occur in the same way on all brokers as a predictable consequence of cluster events. For example binding a queue to a non-existent exchange.
- *Unpredictable* errors may not occur on all brokers. For example running out of journal space to store a message, or an IO error from the journal.

Unpredictable errors must be handled in such a way that the cluster does not become inconsistent. In a situation where one broker experiences an unpredictable error and the others do not, we want the broker in error to shut down and leave the cluster so its clients can fail over to healthy brokers.

When an error occurs on a cluster member it sends an error-check event to the cluster and stalls processing. If it receives a matching error-check from all other cluster members, it continues. If the error did not occur on some members, those members send an error-check with "no error" status. In this case members that did experience an error shut themselves down as they can no longer consistently update their state. The member that did not have the error continue, clients can fail over to them.

Transactions

Transactions are conversational state, allowing a session to collect changes for the shared state and then apply them all at once or not at all.

For TX transactions each broker creates an identical transaction, they all succeed or fail identically since they're all being fed identical input (see Error Handling above for what happens if a broker doesn't reach the same conclusion.)

DTX transactions are not yet supported by the cluster.

Persistence and Asynchronous Journaling

Each cluster member has an independent store, each recording identical state.

A cluster can be configured so that if the cluster is reduced to a single member (the "last man standing") that member can have transient data queues persisted.

Recovery: after a total cluster shutdown, the state of the new cluster is determined by the store of the *first* broker started. The second and subsequent brokers will get their state from the cluster, not the store.

At time of writing there is a bug that requires the stores of all but the first broker to be deleted manually before starting the cluster.

Limitations of current design

There are several limitations of the current design.

Concurrency: all CPG events are serialized into a single stream and handled by a single thread. This means clustered brokers have limited ability to make use of multiple CPUs. Some of this work is pipelined, so there is some parallelism, but it is limited.

Maintainability: decoupling the cluster code from the broker and assuming the broker behaves deterministically makes it very easy for developers working on the stand-alone broker to unintentionally break the cluster, for example by adding a feature that depends on timers.

Non-replicated state: The current design replicates all state. In some cases however, queues are intended only for directly connected clients, for example management queues, the failover-exchange queues. It would be good to be able to define replicated and non-replicated queues and exchanges in these cases.

Scalability: The current cluster design only addresses reliability. Adding more brokers to a cluster will not increase the cluster's throughput since all brokers are doing all the work. A better approach would move some of the work to be done only by the directly-connected broker, and to allow messages to "bypass" the cluster when both producer and consumer are connected to the same member.

Cluster Failover Modes

Qpid cluster failure modes.

This section describes failure modes and techniques to deal with them, the following section provides configuration details for the techniques mentioned here.

Broker process terminated

E.g. broker killed.

Clients: disconnected immediately, can fail over to another broker in the cluster.

Multicast group: broker is automatically removed from the multicast group.

The broker needs to be manually restarted.

Broker host crash

E.g. power failure, hardware failure.

Clients: may not detect loss of connection until a long TCP timeout is reached. Use heartbeats to reduce the time to detect loss of connection.

Multicast group: broker is automatically removed from the multicast group after the configurable totem token timeout value.

Broker freeze -e .g. kill -STOP

E.g. using kill -STOP.

Clients: disconnected after TCP timeout, use heartbeats to disconnect quicker.

Multicast group: Broker is not automatically held up by all cluster traffic. Use the watchdog plugin to kill a broker that is unresponsive for a configured period of time.

Broker needs to be manually restarted.

Client-broker network failure

Clients: disconnected after TCP timeout, use heartbeats to disconnect quicker.

Broker: clean up client resources (e.g. auto-delete queues) when client disconnect is detected after TCP timeout. Use heartbeats to disconnect quicker.

Broker-broker multicast network failure

A failure in the multicast network creates a "partition" creating two or more sub-clusters that are unable to communicate. This creates

inconsistent state in the sub clusters that cannot be reconciled correctly if they are re-connected, and will result in unpredictable behaviour.

To deal with this situation, you need cman's quorum service. In the event of split-brain only one of the sub clusters will have a "quorum". Brokers in the other sub-clusters will automatically shut down, allowing clients to fail over to a broker in the quorum.

Alternatively to avoid partitions entirely you can use the openais/corosync Redundant Ring Protocol which uses two physically separate networks for cluster communication. This enables the multicast group to survive the loss of either of the networks (but not both.)

Brokers that shut down need to be manually restarted.

Broker-broker update network failure.

New brokers joining the cluster receive an initial state snapshot from an established member of the cluster via TCP. A network failure at this point will cause the joining broker to exit.

Broker must be manually restarted.

Note as of qpid 0.6 the update connections are made using the same URL that clients uses to connect, its not possible to restrict broker-broker update connections to a different network from client connections.

Client crash

Broker: client resources such as auto-delete queues are reclaimed immediately.

Client host crash

Broker: client resources such as auto-delete queues are reclaimed after the TCP time-out.
To have resources reclaimed more quickly use heartbeats.

Configuration

Separate client/multicast networks

For best performance use a separate network for clients and the multicast group. If possible the multicast group network should be

openais.conf/corosync.conf

totem.token: timeout in milliseconds until host crash or network disconnect is detected by the multicast group. Defaults to 1000ms.

Redundant ring protocol (RRP), uses two physically separate networks for cluster communication. To use RRP, you must choose a replication mode for your environment. RRP has 3 modes:
Modes

active: Active replication can offer slightly lower latency in faulty network environments, however it can reduce throughput.
passive: Passive replication can nearly double the speed from transmit to delivery, but also carries the potential for the protocol to become bound to a single CPU.
none: Disables redundant ring.

To enable RRP make the following changes to corosync.conf (for RHEL6) or openais.conf (for RHEL5):

1. In the totem section, add rrp_mode=active or rrp_mode=passive
2. Add a second interface section with a different bindnetaddr for your second network.

qpid configuration options

cluster-url: specify addresses that clients will use to connect. Can be used to ensure clients connect on a different network from the multicast network.

Note a future release will provide cluster-update-url to allow updates to be restricted to a different network from client connections.

watchdog plugin

The watchdog plug-in will kill the qpid broker process if it becomes stuck for longer than a configured interval.

If the watchdog plugin is loaded and the --watchdog-interval=N option is set then the broker starts a watchdog process and signals it every N/2 seconds.

The watchdog process runs a very simple program that starts a timer for N seconds, and resets the timer to N seconds whenever it is signalled by the broker. If the timer ever reaches 0 the watchdog kills the broker process (with kill -9) and exits.

This is useful in a cluster setting because in some instances (e.g. while resolving an error) it's possible for a stuck process to hang other cluster members that are waiting for it to send a message. Using the watchdog, the stuck process is terminated and removed from the cluster allowing other members to continue and clients of the stuck process to fail over to other members.

cman configuration

Note: when using cman, do not start the openais/corosync service. It will be started automatically by the cman service.

Only basic cman configuration (cluster.conf) is required. Other cluster suite services (GFS, DLM, fencing etc.) do not need to be configured.

Enabling heartbeats

In C++ clients, heartbeat is disabled by default. You can enable heartbeat by specifying a heartbeat interval (in seconds) for the connection:

```
ConnectionSettings settings;
settings.heartbeat = 1;
FailoverManager fmgr(settings);
```

In a JMS client, heartbeat is set using the idle_timeout property of the connection URL. For instance, the following line from a JNDI properties file sets the heartbeat time out to 3 seconds:

```
pconnectionfactory.qpidConnectionFactory = amqp://guest:guest@clientid/test?brokerlist='tcp://localhost:5672',idle_timeout=3
```

Heartbeats are enabled in both directions, the connection can be closed at either end if the heartbeat interval is missed.

References

Cman configuration: See chapters 3 & 5 of
http://www.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/5.4/html/Cluster_Administration/index.html

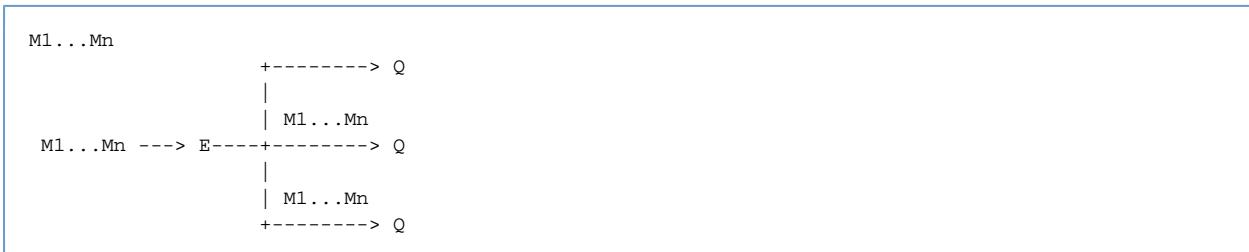
ClusteringAndFederation

Clustering And Federation

Each diagram below depicts a distributed network of exchanges and queues. The following notation is used in all diagrams:

- M: message
- E: exchange
- Q: queue

Multicast



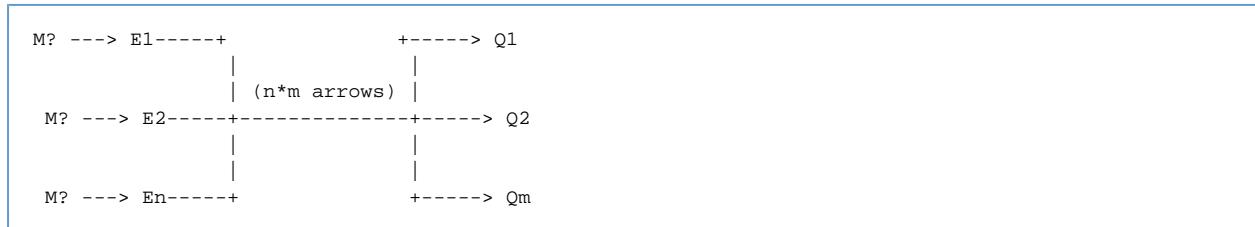
would be ideal between E and Q, or even directly between E and consumers.

Load Balancing



No ordering is guaranteed across different queues. A naive implementation could just be an exchange doing round-robin routing or any algorithm of choice. A more complicated exchange could have flow control between each queue and the exchange.

Multiple Exchanges



Both the Load Balancing and Multicast scenarios can be extended by adding multiple exchange nodes wired into the same (or an overlapping) set of queues. One virtual mega exchange (with relaxed ordering semantics) could be created by segmenting client connections between exchanges. This could be done using a number of strategies, e.g. round-robin dns, name mangling, redirects.

The topologies described above could in theory be used in a variety of scenarios ranging from an isolated high speed subnet with identically configured nodes to a loosely coupled WAN with separately administered nodes. In fact a single network could include exchanges bound to local queues, remote queues available on an isolated high speed subnet, and remote destinations (exchange or queue) available over WAN/internet. In the last case the exchange may be required to queue messages routed to the remote destination if the WAN/internet link is down.

In the terminology I've been using, a cluster is a set of machines sharing the same software and configuration, and generally connected via an isolated high speed subnet. A federation on the other hand consists of distinctly configured machines individually wired together. Both clustering and federation **could** share a common protocol for message delivery. This could possibly even be used for multicast if it were a simple stateless store-and-forward protocol. (Note the "store" in "store-and-forward" can mean both store on disk and store in memory.)

With this model the key distinction between a cluster and a federation is that all the nodes in a cluster are managed as a single unit, e.g. one place to start/stop/add/remove/etc. Because of this the nodes in a cluster have to pass control messages to each other distinct from the general message traffic. These control messages need to be isolated from the general message traffic (e.g. on their own subnet). This could be done using JGroups and OpenAIS for Java and C++ respectively.

This document doesn't directly address fault tolerance, but it is assumed that any node/broker that contains state can be configured to have a passive counterpart that supports two methodologies for failover. Broker swapout based on virtual IP, or client reconnect to a backup IP.

Federation Design Note

Design Note

The information below is quite stale. The outstanding issues listed have been resolved as of M3.

Information needed for this design note:

- Mapping of federation features to source files
- Full description of the dynamic binding protocol and its associated algorithms
- A discussion of how changes to AMQP could improve the protocol (the main thing needed is an arguments map in the unbind method)

Old Content

Formatted mail from Gordon...

Regarding federation, what we have now in the c++ broker is really inter-broker routing.

Links between brokers can be setup to transfer messages from one to another.

In the current terminology a 'link' is a connection between two brokers. Such a link is setup using the management system, by asking one broker to establish a connection to another broker given the host and port.

Once a link is established, a 'bridge' can be created. A bridge is essentially a subscription for messages between two brokers, requesting the transfer of messages from a source to a destination. The 'source' for a bridge can logically be either an exchange or a queue; the destination is an exchange on the receiving broker.

The current implementation of bridges relies on the symmetry of the message.transfer command in the 0-10 AMQP specification. A bridge is created by issuing a subscribe request to one broker using the exchange name to which the messages should be delivered as the 'destination' argument. So once the subscription is setup the bridge appears to be a standard consumer to the source broker and messages routed from that broker appear as standard publications at the source broker.

If the logical source of the bridge was an exchange rather than a queue, an exclusive queue is created for the bridge and bound with the relevant binding details (currently only a binding key is supported, but that's easy to extend).

Bridges can be established to support different types of message flow. A common case is where you have two or more brokers over which you want to offer a 'federated exchange'. I.e. you want messages published to that exchange on one broker to be routed through the equivalent exchanges on all the brokers in the federation, allowing queues bound locally at those brokers to receive such messages. This common case is supported by the qpid-route tool.

There are currently a few outstanding issues needing to be resolved.

- One is preventing messages from looping in configurations where there are circularities in the defined routes (such as those described for the 'federated exchange'). I plan to address that next week. The solution I have in mind is to have the exclusive queues used for bridging from exchanges append an identifier to a custom property ('x-qpid-route' or whatever) in each message that passes through them. It will then be possible to specify a list of exclusions when establishing a bridge and messages where the route property contains any of the excluded identifiers will be silently dropped. I'm not entirely delighted with that approach, but it will have to do in the short term I think.
- Another is ensuring that links are re-established when lost (e.g. due to network failures or brokers being taken down) and that the details of configured bridges survive restart. These will also be addressed quite soon I hope.

This is obviously just the beginning of full federation capabilities. There are many ways it can be made more sophisticated and I for one would be interested in debating ideas, use cases and directions.

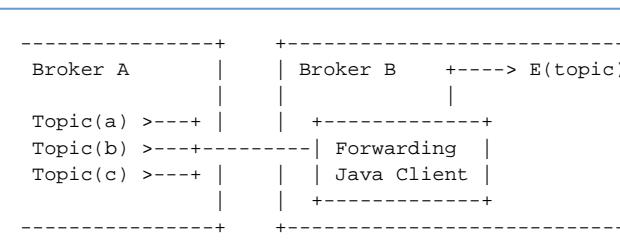
Java Federation Design Proposal

Message Federation Design Proposal

The following proposal is only to address two types of message federation. This is aimed to be implemented in the Java broker before we fully upgrade it to support AMQP 0-10.

The two types of message federation that we will consider. Topic Fanout and Remote Queues.

Topic Fanout



Broker B has a process where it subscribes to various topics on Broker A forwarding the messages on to the

Specified Exchange on B.

New Dynamic Topic Exchange

An additional topic exchange type can be added so that any *bind* request is added to the existing set of subscriptions. This would remove the need to explicitly configure any forwarding and so ensure that any new clients that joined would receive messages on the topic they requested.

Embedded Client

While the client could be a much simpler client than the existing Java Client. By utilising the existing java client reduce the replication of functionality. If the performance overhead of using the full client is shown to be too high then we can revisit this situation.

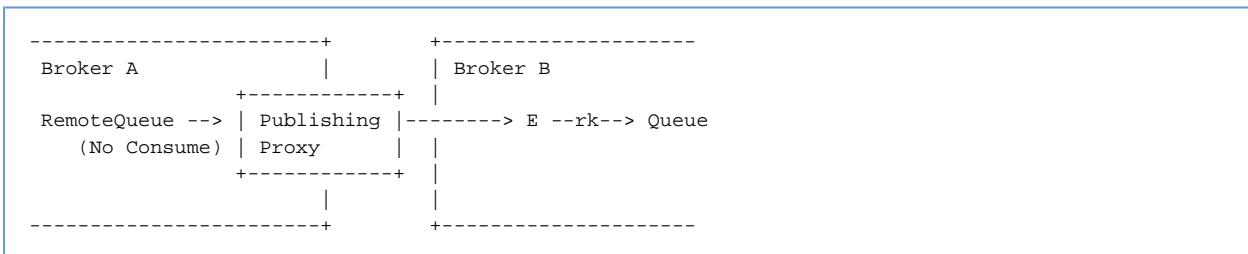
There are several features that the Forwarding Client should perform:

- 'Topic Reduction', 'weather.europe.scotland.*' and 'weather.europe.#' can be reduced to 'weather.europe.#'
- Handle reconnection to Broker A.
- Loop detection

Loop Detection

If a tag identifying the broker is added to each message that is forwarded then the client can tell if a message has already been forwarded and so silently drop the looped message. This detection could be made more proactive by having the Client subscriptions use a selector to specifically exclude all messages with the Broker B's ID.

Remote Queues



The Remote Queue case has a Publishing Proxy that will push all the messages that arrive on the 'RemoteQueue' to the specified Exchange (E) and RoutingKey (rk).

The 'RemoteQueue' should potentially prevent any client subscriptions from directly consuming from the Queue on Broker A. Attempts to consume from this queue could utilise the redirect method to send the requesting client to Broker B.

Publishing Proxy

All messages should be sent via transaction to maintain the reliability of the transfer.
Just as with the Forwarding Client this proxy must be capable of reconnecting in the event of connection failure.

Old Clustering Design Note

Overview

The following is a proposal for the design of a clustering solution to increase the scalability of the Qpid AMQP broker by allowing multiple broker processes to collaborate to provide services to application clients connected to any one of these processes. By spreading the connections across different processes more clients can be supported.

Terms & Definitions

A cluster consists of any number of brokers in a fixed order. Each broker in the cluster has a unique name. All brokers in the cluster know the set of brokers in their cluster and agree on the ordering of those brokers. The first broker in the cluster is the leader; all brokers agree on the current leader. The mechanisms for achieving and maintaining this structure will be described below.

Each client is connected to one broker in the cluster, to whom it sends its requests in the same way it would were it connected to a broker that was not part of a cluster. Clients that implement the AMQP specification should be able to work with a clustered broker unaltered. However, when in a cluster, a broker will need to alter its response to certain client requests. An objective of the design is to minimise the scope and complexity of this altered behaviour.

Brokers in a cluster all connect to each other. Though one socket between any two brokers would suffice, it is simpler to implement if we assume that each broker will be a client of each other broker. Thus there will in fact be two connections between any two members, one in each 'direction'. This way we can reuse as much of a non-clustered brokers behaviour as possible.

A broker will need to distinguish between sessions with an application client and sessions where the 'client' of the socket in a session is actually another broker.

Outline of Approach for Clustering

Stated simply, the cluster will:

- replicate exchanges by broadcasting the original Exchange.Declare and Exchange.Delete messages to all members of the cluster.
- replicate queues by broadcasting the original Queue.Declare and Queue.Delete messages to all members of the cluster
- replicate bindings by broadcasting Queue.Bind messages to all members of the cluster
- relay messages from a copy of an exchange in one broker to the equivalent exchange in another broker where necessary to ensure that consumers on any broker in the cluster receive messages that are published to any broker in the cluster

Private queues exist in real form in only one broker; the broker that receives the original declaration from a client. All the other brokers in the cluster set up a proxy for this queue. This proxy can be bound to exchanges as a normal queue can, but whenever a message is routed to it, that message is simply relayed on to the broker in which the real queue exists. However, though multiple queue proxies may exist with the same target member, if these are bound such that all of them match a given message, only one copy of that message should be relayed to the target member.

Copies of shared queues will exist at each broker in the cluster. These are bound to exchanges as usual and can have consumers registered with them. In addition to consumers from the application clients, these shared queue copies track the number of consumer for that queue that are held on other brokers. They use this information to fairly distribute messages between all consumers.

The clustering in general involves propagation of certain methods received by one broker in the cluster from a client to all the other cluster members. Specifically those methods concerned with the setup of routing information are propagated allowing all members of the cluster to play their part in the routing of messages from and to clients distributed across the cluster.

In particular the cluster will propagate all Exchange.Declare, Exchange.Delete, Queue.Declare, Queue.Delete and Queue.Bind messages. It will also propagate Basic.Consume and Basic.Cancel messages that refer to shared queues.

The propagation can be carried out synchronously or asynchronously with respect to the original client request. In other words the broker that receives one of these messages from a client will send an equivalent message to the other brokers and can then wait until it receives responses from these brokers before it sends the confirmation message back to the client. Alternatively it could return a response to the client immediately. A hybrid approach could also be used. In general the originating broker waits for n responses, where $0 < n <$ number of members in the cluster. The value of n to be used will be set through policies to achieve the required latency v. consistency trade offs for a particular situation.

Cluster Management

As mentioned above the cluster is defined to be an agreed set of member brokers in an agreed order. This helps reasoning about consistency. The 'first' member of the group acts as the leader and issues authoritative statements on who is in or out of the cluster. All brokers in the cluster store the last received membership announcement from which they can infer the current leader.

Ordering is maintained by requiring that new members join through the leader. A prospective new member can connect to any other member in the cluster, but these other members should pass on the join request to the leader.

Once connected to a member of the group the new member issues a join request, to which the leader responds by sending a new membership announcement to all members including the new member. It will also initiate the replay messages required to replicate cluster state to the new member; the other cluster members also participate in this message replay. Once it has processed all the replayed messages and is therefore up to date with respect to cluster state, the new member can start accepting client connections.

State is transferred through (a) Exchange.Declare methods for all exchanges, (b) Queue.Declare messages for all queues, (c) Queue.Bind requests for all queue bindings in all exchanges and (d) Basic.Consume requests for all consumers of shared queues at each node. The leader is responsible for replicating all exchanges, shared queues and their bindings. Other members are responsible for replicating private queues hosted by them and the bindings for these queues as well as consumer counts for shared queues. The replay of messages from the leader must be processed before those from other cluster members (as e.g. bindings for private queues require that the exchanges have already been declared). The completion of the required replay of messages from a broker is signaled by a Cluster.Synch message. Messages received after this are 'live' messages received through the receiving broker being treated as a normal member.

Failure of a broker may be detected by any other broker in the cluster in the course of trying to communicate with that broker. Failures are handled by sending a suspect message to the leader of the cluster, who verifies the suspected broker is down and issues a new announcement of membership, with the failed broker removed if the failure is verified. In addition to discovery of failure during normal communication, each broker member is responsible for periodically pinging the 'previous' broker (i.e. the broker that occurs just before itself in the ordered membership list). The leader will assume responsibility for pinging the last member to join the group.

The leader may itself fail. This may be detected by the next broker in the list, in which case that broker responds by assuming leadership and sending an announcement of the new membership list with the old leader removed. It may also be detected by other brokers. As they cannot send a suspect warning to the leader, they send it to the broker next to the leader.

Message Handling Changes and Protocol Extensions

To incorporate clustering while reusing the same communication channel for intra-cluster communications and extension to the protocol is proposed. It is not necessary for clients to know about this extension so it has no impact on the compliance of the broker and can be treated as a proprietary extension for Qpid. The extension consists of a new class of messages, Cluster, which has the following methods:

Cluster.Join

Sent by a new member to the leader of the cluster to initiate the joining process. On receiving a join the leader will try to establish its own connection back to the new member. It will then send a membership announcement and various messages to ensure the new member has the required state built up.

Cluster.Membership

Sent by the leader of the cluster whenever there is a change in the membership of the cluster either through a new broker joining or through a broker leaving or failing. All brokers should store the membership information sent. If they are waiting for responses from a member that is no longer part of the cluster they can handle the fact that that broker has failed. If it contains a member to whom they have not connected they can connect (or reconnect).

Cluster.Leave

Sent to the leader by a broker that is leaving the cluster in an orderly fashion. The leader responds by sending a new membership announcement.

Cluster.Suspect

Sent by brokers in the cluster to the leader of the cluster to inform the leader that they suspect another member has failed. The leader will attempt to verify the failure and then issue a new Cluster.Membership message excluding the suspected broker if it has failed leaving it in if it seems to be responding.

Cluster.Synch

Sent to complete a batch of message replayed to a new member to allow it to build up the correct state.

Cluster.Ping

Sent between brokers in a cluster to give or request a heart beat and to exchange information about loading. A ping has a flag that indicates whether it expects a response or not. On receiving a ping a broker updates its local view of the load on that server and if required sends its own ping in response.

In addition to this new class, the handling of the following is also altered. The handling of each message may depend on whether it is received from an application client or from another broker.

Connection.Open

A broker needs to detect whether the open request is from an application client or another broker in the cluster. It will use the capabilities field to do this; brokers acting as clients on other brokers require the 'cluster-peer' capability.

If a broker receives a Connection.Open from an application client (i.e. if the cluster-peer capability is not required) it may issue a Connection.Redirect if it feels its loading is greater than the loading of other members in the cluster.

Exchange.Declare

On receiving this message a broker propagates it to all other brokers in the cluster, possibly waiting for responses before responding with an Exchange.Declare-Ok.

Queue.Declare

On receiving this message a broker propagates it to all other brokers in the cluster, possibly waiting for responses before responding with a Queue.Declare-Ok.

Queue.Bind

Again, this is replicated to all other brokers, possibly waiting for responses before sending back a Queue.Bind-Ok to the client.

Queue.Delete

On receiving this message a broker propagates it to all other brokers in the cluster, optionally waiting for responses before responding to the client.

Basic.Consume

If the consume request is for a private queue, no alteration to the processing is required. However, if it is for a shared queue then the broker must additionally replicate the message to all other brokers.

Basic.Cancel

If the cancel request is for a subscription to a private queue, no alteration to the processing is required. However, if it is for a shared queue then the broker must additionally replicate the message to all other brokers.

Basic.Publish

The handling of Basic.Publish only differs from the non-clustered case where (a) it ends up in a shared queue or (b) it ends up in a 'proxy' for a private queue that is hosted within another member of the cluster.

When the published message ends up in a shared queue, the broker must be aware of whether the message was published to it by another broker or by an application client. Messages that come from other brokers are dispatched to the local brokers own application client subscribers. Messages that come from application clients are either dispatched to the next application client or relayed to another broker. A round-robin scheme applies here where each subscriber, whether a 'real' subscriber or a consumer in a relay link to another broker, gets its 'turn'.

In other words the allocation of a message to a consumer on a shared queue happens at the first broker to receive the publish request from

the application. All brokers signal their local consumer count by propagating the Basic.Consume (and Basic.Cancel) messages they receive from clients so each broker has a local view of the cluster wide distribution of consumers which can be used to achieve a fair distribution of messages received by that broker.

As each broker can receive messages from the application, strict round-robin delivery is not guaranteed, but in general a fair distribution will result. Brokers should remember the next consumer to receive messages from the application and also the next consumer to receive messages from the cluster.

A local broker's view of consumer distribution is updated asynchronously with respect to message publishing and dispatch. This means that the view might be stale with regard to the remote consumer counts when the next consumer for a message is determined. It is therefore possible that one broker directs a message to a broker that it thinks has a consumer, but when that message arrives at the remote broker the consumer has disconnected. How this is handled should be controlled through different policies: pass it on to another broker, possibly with the redelivered flag set (particularly if it goes back to the broker it came from), discard the message or hold on to it for a finite period of time and deliver it to any application consumer that subscribes in that time.

The situation just described is essentially the same situation as in a non-clustered case where a consumer disconnects after a message has been sent to it, but before it has processed that message. Where acknowledgements aren't used the message will be lost, where acknowledgements or transactions are used the message should be redelivered, possibly out of sequence. Of course in the clustered case there is a wider window in which this scenario can arise.

Where the messages is delivered to a proxied private queue, that message is merely relayed on to the relevant broker. However, It is important that where more than one proxied queue to the same target broker are bound to the same exchange, the message only be relayed once. The broker handling the Basic.Publish must therefore track the relaying of the message to its peers.

Failure Analysis

As mentioned above, the primary objective of this phase of the clustering design is to enable the scaling of a system by adding extra broker processes that cooperate to serve a larger number of clients than could be handle by one broker.

Though fault tolerance is not a specific objective yet, the cluster must allow for the failure of brokers without bringing the whole system to a halt.

The current design (and implementation) only handles process failures entirely satisfactorily. Network failures* result in the exclusion of brokers from the cluster and will behave reasonably only where the view of reachability is consistent across the cluster. Network partitions between the cluster nodes will result in independent clusters being formed and there is currently no provision for merging these once the partition heals.

- failures here means anything that causes a tcp stream to fail; a relatively straightforward improvement would be to buffered unacknowledged requests that have been broadcast allowing attempts to re-establish a tcp connection on failure and replaying the messages (assuming idempotent messages)

The group abstraction described above does not provide virtual synchrony. When a broker fails while performing a broadcast to the group, the result will not be uniform across the other members. Where synchronous propagation is used, the client will be aware of this state as it will not have received the response from the broker and will reissue the request on failing over to another broker. (The current failover as implemented in the Qpid client will actually recreate all state required by the client).

Persistent Cluster Restart Design Note

Persistent cluster, user perspective.

A persistent cluster is one where all members have a persistent store. A cluster must have all transient or all persistent members, mixed clusters are not allowed.

cluster-size option

`cluster-size N` Wait for at least N initial members before completing cluster initialization and serving clients.

Use this option in a persistent cluster so all brokers in a persistent cluster can exchange the status of their persistent store and do consistency checks before serving clients.

Clean and dirty shut-down.

Each store is an independent replica of the cluster's state. If a broker crashes while there are other brokers running, its store is marked "dirty" because it will be out-of-date with regard to the rest of the cluster.

If the broker is re-started to re-join the a running cluster it will discard the dirty store and get an update from an active cluster member to re-synchronize its state.

If the entire cluster is shut down by an administrator using the `qpid-cluster -k` command, then all brokers will shut down at exactly the same point with the same state in their stores. In this case the stores are marked "clean".

If the cluster is reduced to a single broker, and that broker is shut down, its store is marked clean since it is the the only broker and therefore has the authoritative store.

When the cluster is restarted, brokers with clean stores will recover from their store, brokers with dirty stores will get an update from a clean broker.

Consistency checks

Two UUIDs are saved with each broker's store: cluster-id and shutdown-id. These are used during startup to detect a mistaken attempt to use mis-matched stores.

The cluster-id identifies the persistent cluster. It remains the same if the cluster is shut down and restarted. It ensures no accidental mixing of stores belonging to different clusters.

The shutdown-id identifies a particular clean shut-down event. It ensures that all clean stores were shut down at the same point.

If there is any mis-match in these IDs, all members of the cluster will log a message and exit.

Manual recovery

In the unlikely event that all brokers in a cluster crash so close together that it's impossible to determine which was the last one to shut down, all their stores will be dirty.

In this case manual intervention is required to identify which store to recover from.

TODO: describe manual intervention: two parts. First identify which is the best store to start from. Second mark the store as clean by writing a UUID to the shutdown ID in the data directory.

Design details

Persistent restart scenarios:

- first run of persistent cluster, all members have empty stores.
- persistent member crashes and re-starts - re-joins running cluster
- automatic restart after orderly shutdown of persistent cluster
- manual recovery after total cluster failure of persistent cluster

Other requirements:

- cluster initialization: wait for N initial members before going active.
- enforce consistency of broker options that need to be identical across cluster

Persistent cluster

Store states on broker start-up:

- empty: not used before.
- clean: has state, was shut down by admin. Has initial and shutdown-ids
- dirty: has state, not shut down by admin. Has cluster-id.

cluster-id is stored on the first run of a persistent cluster. Used to ensure members are part of the same cluster.

shutdown-id is stored at administrative shut-down of the cluster. Used to ensure clean stores are from the same shut-down event.

Initialization

1. Wait for N initial members
2. Verify options are consistent for all members or abort.
3. Verify valid store states or abort (see below)
4. Members with empty/dirty stores get update from clean member.

All empty is a valid store state: all members record the same cluster-id and go active.

If any are non empty then

- at least one store must be clean
- all clean stores must have same shutdown-id.
- all clean and dirty stores must have same cluster-id.

All clean members restore from stores. All empty members set the cluster-id from the cluster. All dirty/empty members get an update from a clean member.

Joining

If the new member has a non-empty store, the cluster-id must match the cluster. The new member gets an update from the cluster.

Manual Recovery

TDB: how to identify the best store?

Reliability Requirements

Reliability Requirements

Fail-over (session state)

A cluster member informs its clients of backup candidates for each session. It can update the list periodically.

After an unexpected disconnect the client can connect to one of the candidates and resume its session transparently. All session state is preserved including:

- Open references
- Active consumers
- Commands-in-flight
- Open transactions (question: Is there any value in fail-over that aborts TX and/or DTX transactions?)

Sessions *do not* survive

- multiple failures that include the current node and all back-up nodes for that session.
- shutdown/restart of the cluster.

Cluster Restart (durable resources)

The AMQP entities that survive a restart are those defined by AMQP to survive broker restart. AMQP defines *durable* exchanges and queues and *persistent* messages. Some further definitions:

- *durable* message: persistent messages on a durable queue.
- *durable* enqueue: act of enqueueing a persistent message on a durable queue.
- *durable* binding: binding between durable exchange and durable queue.

The following are preserved if the entire cluster shuts down/crashes and is re-started:

- *Durable* wiring: durable exchanges, queues and bindings.
- *Durable* messages
- *Prepared* DTX transactions

The following do not survive a restart:

- Session state
- Non-durable wiring
- TX transactions are aborted.
- Unprepared DTX transactions are aborted.
- Non-durable effects of prepared DTX transactions are lost.

Restarting DTX Transactions

On restart, prepared DTX transactions may commit or rollback. In either case the outcome is *as if* the transaction had committed or rolled back just *before* the restart: All durable transaction effects survive the restart, all non-durable effects are lost.

In particular

- On **commit**: *non durable* messages enqueued in the transaction are *lost*, as if they had been enqueued before the restart and were lost in the restart.
- On **rollback**: *non durable* messages dequeued in the transaction are *lost*, as if they had been put back on the queue before restart and then lost in the restart.

Declarative System Testing

The Java and C++ have fairly extensive system tests (the Java has less extensive unit tests, I'm not sure about the C++ coverage). The .Net has significantly less, but they tend to transliterations of the Java tests. There's also a bunch of interop tests which are reimplemented in each of the languages, but has patchy coverage (Java and C++ implement most, the .Net implements less, I don't think Python implements any). While attempting to automate running these, it was pointed out that the obvious model for the interop tests is that the coordinator sends a test case to the clients describing what they should do rather than a "run test 1" message.

Proposal:

Implement a generic system for turning test definitions into test code.

Outline:

Given a document like this:

```
<test>
<create type="queue" name="queue" exchange="amq.direct" routing-key="queue">
<create type="consumer" name="consumer" destination="queue">
<create type="producer" name="producer" destination="queue">
<send number="10" exchange="amq.direct" routing-key="queue" size="1024" producer="producer">
<receive number="10" consumer="consumer">
</test>
```

the test would send 10 messages through the broker and read them back.

This would allow for easy sharing of test cases throughout the clients and would mean that the interop test co-ordinator could send an xml document to the clients to allow for easy extension of the interop tests without having clients lagging behind.

Problems:

Well, writing such a beast shouldn't be too difficult, although a couple of questions immediately raise their heads with regard to onMessage vs receive senders and threading that would need to be addressed. Also the test grammar would need to be defined and agreed quite closely for this to really work.

The other question is whether the test runners should interpret the XML directly, which would allow for consistency of code between the test suite and the interop testing, or if they should generate FooUnit test cases which would allow for easier debugging but a longer build cycle.

Comments and implementations gratefully received.

Developer Pages

Developer Pages

- [Java Coding Standards](#)
- [Cpp Client Java Interop Issues](#)
- [Java Broker Design](#)
- [Qpid Java Client refactoring](#)
- [Distributed Testing](#)
- [Low-Level API Diagram](#)
- [Weekly QPID Developer Meetings](#)
- [Documentation](#)
- [ACL](#)
- [Qpid Management Framework](#)
- [Broker job queue limits](#)
- [JMX Console Use Cases](#)
- [Current Architecture](#)
- [MessageProducer.send\(\) behaviour](#)
- [Multiple Java Brokers - Use Cases](#)
- [Java Client Test Coverage](#)
- [ACL Design](#)
- [AMQP Distributed Transaction Classes \(C++\)](#)
- [API Error Conditions](#)
- [Broker Management QMF Coverage](#)
- [Java Client Design](#)
- [Qpid extensions to AMQP](#)
- [Qpid Java Broker - Guidance for 64Bit VM](#)

Process Notes

[Release Process](#)

Testing

See [Qpid Testing](#)

- [Qpid JMX Management Console Testing Guide](#)
- [Interop Testing Specification](#) - Common test cases to ensure all clients and brokers interop.
- [Performance, Reliability and Scaling](#) - Details of the test cases and telemetry available
- [Java Client Test Coverage](#)

Design Notes

- [Management Design notes](#) - The layered AMQP management protocol for mgmt tools (currently in the M3 C++ broker)
- [ClusteringHA](#) - Federation, HA, and Clustering design notes
- [Queue Replay](#) - Adding replay to queues.
- [The AMQP Distributed Transaction Classes \(Java\)](#) - The+AMQP+Distributed+Transaction+Classes
- [AMQP Distributed Transaction Classes \(C++\)](#) - Distributed Transaction handling in the C++ broker
- [ACL](#) - design page
- [ACL Design](#) - design page
- [QMF](#) - The Qpid Management Framework

Java Coding Standards

This page documents the standard adopted for Java code in the Qpid project. All committers are expected to follow these standards; checkstyle or similar is used to check compliance.

Executive Summary

The main things for layout purposes in the standard are:

- Indent using four spaces. **No tabs**.
- braces always go on new lines, e.g.

```
if (x == 5)
{
    System.out.println("Hello");
}
```

rather than

```
if (x == 5) {
    System.out.println("Hello");
}
```

- Always add braces, e.g.

```
if (x == 5)
{
    System.out.println("Hello");
}
```

rather than

```
if (x == 5}
    System.out.println("Hello");
```

- Fields prefixed with underscores, e.g. `_messageCount`
- Spaces after keywords but no spaces either before or after parentheses in method calls, e.g.

```
if (x == 5)
```

rather than

```
if(x==5)
```

but

```
foo.bar(4, 5)
```

rather than

```
foo.bar( 4, 5 )
```

Details

Introduction

This document describes two types of coding standard:

1. **Mandatory** standards must be followed at all times.
2. **Recommended** standards should in general be followed but in particular cases may be omitted where the programmer feels that there is a good reason to do so.

Code that does not adhere to mandatory standards will not pass the automated checks (or a code review if the guideline is not stylistic).

Source files

This section defines the general rules associated with the contents of a Java source file and the order in which each part should be presented. No rules on programming style, naming conventions or indentation are given here.

1. Java source files must have a ".java" suffix (this will be enforced by the compiler) [mandatory].
2. The basename of a Java source file must be the same as the public class defined therein (this will be enforced by the compiler) [mandatory].
3. Only one class should be defined per source file (except for inner classes and one-shot uses where the non-public class cannot conceivably be used outside of its context) [mandatory].
4. Source files should not exceed 1500 lines [recommended].
5. No line in a source file should exceed 120 characters [mandatory].
6. The sections of a source file should be presented in the following order [mandatory]:
 - File information comment (see rule 7 below).
 - Package name (see rules 1 to 3 in the section 2.1 above and rule 8 below).
 - Imports (see rules 9 to 10 below).
 - Other class definitions.
 - Public class definition.
1. Do not use automatically expanded log or revision number provided by your source code management system unless it provides a facility to avoid "false conflicts" when doing merges due simply to revision number changes (which happens, for example, with cvs when branches are used). [mandatory]
Rationale: classes that are not explicitly put in a package are placed in the unnamed package by the compiler. Therefore as the classes from many developers will be being placed in the same package the likelihood of a name clash is greatly increased.
2. Every class that is to be released must be a member of a package [mandatory].
3. All class imports from the same package should be grouped together. A single blank line should separate imports from different packages [recommended].
4. Use javadoc tags and use HTML mark-up to enhance the readability of the output files [mandatory].

Java Elements

This section gives advice on coding the various elements of the Java programming language.

Class definitions

This section gives guidelines for class and interface definitions in Java. The term class in this section is used more broadly to mean class and interface:

1. Class names should start with a capital letter with every subsequent word capitalised, for example: DataProcessor [mandatory].
2. The name of exception classes should end in the word exception, for example: UnknownMungeException [mandatory].
3. Class names should in general not be overloaded. For example, defining a class "com.foo.bar.String" should be avoided as there is already a class "java.lang.String" [recommended].
Rationale: adhering to this rule reduces the likelihood of confusion and means that the use of fully qualified class names should not be required.
4. The definition of the primary class (i.e. the class with the same name as the java file) should start in column 0 of the source file. Inner class definitions should be indented 4 spaces more than their enclosing class [mandatory].
5. Declare a class as final only if specialisation will never be required and improved performance is essential. With modern JVMs there in fact may be no performance advantage. Warning: use of final limits code reuse [mandatory].
6. For all but simplest classes the following methods should have useful definitions [recommended]:

```
public boolean equals(Object obj)
public int hashCode()
public String toString()
```

7. The order of presentation of the sections in a class should be [mandatory]:

- Variables
- Methods

Variables

This section gives guidelines for class and instance variable definitions in Java. In this section if a rule uses the term variable rather than instance variable or class variable, then the rule applies to both types of variable.

1. The order of presentation of variables in a class definition should be [recommended]:
 - private, protected, public: static final variables (aka constant class variables).
 - private, protected, public: static variables (aka class variables).
 - private, protected, public: final variables (aka constant instance variables).
 - private, protected, public: variables (aka instance variables).

It should be noted that as javadoc will automatically order variables in a consistent manner, rigid adherence to this rule is not necessary.
1. Variable modifiers should be presented in the following order: static, final, transient, volatile [mandatory].
2. The names of static final variables should be upper case with subsequent words prefixed with an underscore [mandatory]. For example:

```
public static final int NOT_FOUND = -1;
```

3. When a subclass refers to a static final variable defined in a parent class, access should be qualified by specifying the defining class name [mandatory]. For example: use ParentClass.MAX rather than MAX.
4. The names of variables (other than static final) should start with a lower case letter. Any words that are contained in the rest of the variable name should be capitalised [mandatory]. For example:

```
String name;  
String[] childrensNames;
```

5. Class and instance variables must be prefixed with an underscore (_) [mandatory].
6. Variables must not be named using the so-called Hungarian notation [mandatory]. For example:

```
int nCount = 4; // not allowed
```

7. Only one variable may be defined per line [mandatory].
8. Variable declarations should be indented 4 spaces more than their enclosing class [mandatory].
9. All variables should be preceded by a javadoc comment that specifies what the variable is for, where it is used and so forth. The comment should be of the following form and be indented to the same level as the variable it refers to [mandatory]
10. Never declare instance variables as public unless the class is effectively a "struct" [mandatory].
11. Never give a variable the same name as a variable in a superclass [mandatory].
12. Ensure that all non-private class variables have sensible values even if no instances have been created (use static initialisers if necessary, i.e. "static { ... }") [mandatory].

Rationale: prevents other objects accessing fields with undefined/unexpected values.

Methods

This section gives guidelines for class and instance method definitions in Java. In this section if a rule uses the term method rather than instance method or class method, then the rule applies to both types of method.

13. Constructors and finalize methods should follow immediately after the variable declarations [mandatory].
14. Do not call non-final methods from constructors. This can lead to unexpected results when the class is subclassed. If you must call non-final methods from constructors, document this in the constructor's javadoc [mandatory]. Note that private implies final.
15. Methods that are associated with the same area of functionality should be physically close to one another [recommended].
16. After grouping by functionality, methods should be presented in the following order [recommended]:

- private, protected, public: static methods.
- private, protected, public: instance methods.

It should be noted that as javadoc will automatically order methods in a consistent manner, rigid adherence to this rule is not necessary.

1. Method modifiers should be presented in the following order: abstract, static, final., synchronized [mandatory]
2. When a synchronized method is overloaded, it should be explicitly synchronized in the subclass [recommended].
3. Method names should start with a lower case letter with all subsequent words being capitalised [mandatory]. For example:

```
protected int resize(int newSize)  
protected void addContentsTo(Container destinationContainer)
```

4. Methods which get and set values should be named as follows [mandatory]:

```
Type getVariableName()  
void setVariableName(Type newValue)
```

Exceptions should be used to report any failure to get or set a value. The "@param" description should detail any assumptions made by the implementation, for example: "Specifying a null value will cause an error to be reported".

5. Method definitions should be indented 4 spaces more than their enclosing class [mandatory].
6. All methods should be preceded by a javadoc comment specifying what the method is for, detailing all arguments, returns and possible exceptions. This comment should be of the following form and be indented to the same level as the method it refers to [mandatory].
7. The braces associated with a method should be on a line on their own and be indented to the same level as the method [mandatory]. For example:

```

public void munge()
{
    int i;
    // method definition omitted...
}

```

8. The body of a method should be indented 4 columns further than the opening and closing braces associated with it [mandatory]. See the above rule for an example.
9. When declaring and calling methods there should be no white space before or after the parenthesis [mandatory].
10. In argument lists there should be no white space before a comma, and only a single space (or newline) after it [mandatory]. For example:

```

public void munge(int depth, String name)
{
    if (depth > 0)
    {
        munge(depth - 1, name);
    }
    // do something
}

```

11. Wherever reasonable define a default constructor (i.e. one that takes no arguments) so that Class.newInstance() may be used [recommended]. If an instance which was created by default construction could be used until further initialisation has been performed, then all unserviceable requests should cause a runtime exception to be thrown.
12. The method public static void main() should not be used for test purposes. Instead a test/demo program should be supplied separately. [mandatory].
13. Public access methods (i.e. methods that get and set attributes) should only be supplied when required [mandatory].
14. If an instance method has no natural return value, declare it as void rather than using the "return this;" convention [mandatory].
15. Ensure that non-private static methods behave sensibly if no instances of the defining class have been created [mandatory].

Expressions

This section defines the rules to be used for Java expressions:

16. Unary operators should not be separated from their operand by white space [mandatory].
17. Embedded ++ or – operators should only be used when it improves code clarity [recommended]. This is rare.
18. Extra parenthesis should be used in expressions to improve their clarity [recommended].
19. The logical expression operand of the "?:" (ternary) operator must be enclosed in parenthesis. If the other operands are also expressions then they should also be enclosed in parenthesis [mandatory]. For example:

```

biggest = (a > b) ? a : b;
complex = (a + b > 100) ? (100 * c) : (10 * d);

```

20. Nested "?:" (ternary) operators can be confusing and should be avoided [mandatory].
21. Use of the binary "," operator (the comma operator) should be avoided [mandatory]. Putting all the work of a for loop on a single line is not a sign of great wisdom and talent.
22. If an expression is too long for a line (i.e. extends beyond column 119) then it should be split after the lowest precedence operator near the break [mandatory]. For example:

```

if ((state == NEED_TO_REPLY) ||
    (state == REPLY_ACK_TIMEOUT))
{
    // (re)send the reply and enter state WAITING_FOR_REPLY_ACK
}

```

Furthermore if an expression requires to be split more than once, then the split should occur at the same logical level if possible.

23. All binary and ternary operators (exception for ".") should be separated from their operands by a space [mandatory].

Statements

Simple Statements

This section defines the general rules for simple Java statements:

24. There must only be one statement per line [mandatory].
25. In general local variables should be named in a similar manner to instance variables [recommended].

26. More than one temporary variable may be declared on a single line provided no initialisers are used [mandatory]. For example:

```
int j, k = 10, l; // Incorrect!
int j, l;          // Correct
int k = 10;
```

27. A null body for a while, for, if, etc. should be documented so that it is clearly intentional [mandatory].

28. Keywords that are followed by a parenthesised expression (such as while, if, etc) should be separated from the open bracket by a single space [mandatory]. For example:

```
if (a > b)
{
    munge();
}
```

29. In method calls, there should be no spaces before or after the parentheses [mandatory]. For example:

```
munge (a, 10); // Incorrect!
munge(a, 10); // Correct.
```

Compound Statements

This section defines the general rules associated with compound statements in Java:

30. The body of a compound statement should be indented by 4 spaces more than the enclosing braces [mandatory]. See the following rule for an example.
31. The braces associated with a compound statement should be on their own line and be indented to the same level as the surrounding code [mandatory]. For example:

```
if ((length >= LEN_BOX) && (width >= WID_BOX))
{
    int i;
    // Statements omitted...
}
```

32. If the opening and closing braces associated with a compound statement are further than 20 lines apart then the closing brace should be annotated as follows [mandatory]:

```
for (int j = 0; j < SIZE; j++)
{
    lotsOfCode();
} // end for
```

33. All statements associated with an if or if-else statement should be made compound by the use of braces [mandatory]. For example:

```
if (a > b)
{
    statement();
}
else
{
    statement1();
    statement2();
}
```

34. The case labels in a switch statement should be on their own line and indented by a further 4 spaces. The statements associated with the label should be indented by 4 columns more than the label and not be enclosed in a compound statement. [mandatory]. For example:

```

switch (tState)
{
    case NOT_RUNNING:
        start();
        break;

    case RUNNING:
    default:
        monitor();
        break;
}

```

35. In switch statements - the statements associated with all cases should terminate with a statement which explicitly determines the flow of control, for example break [recommended].
36. In switch statements - fall through should be avoided wherever possible, however if it is unavoidable it must be commented with "/* FALLTHROUGH */" [mandatory].
37. In switch statements - a default case must be present and should always be the last case [mandatory].

General

This section gives general rules to be followed when programming in Java:

38. When comparing objects for equivalence use the method equals() and not the == operator. The only exceptions to this are static final objects that are being used as constants and interned Strings [mandatory].
39. In general labelled break and continue statements should be avoided [recommended]. This is due to the complex flow of control, especially when used with try/finally blocks.
40. Unless some aspect of an algorithm relies on it, then loops count forward [mandatory]. For example:

```

for (int j = 0; j < size; j++)
{
    // Do something interesting
}

```

41. Use local variables in loops [recommended]. For example:

```

ArrayList clone = (ArrayList)listeners.clone();
final int size = clone.size();
for (int j = 0; j < size; j++)
{
    System.out.println(clone.elementAt(j));
}

```

42. Anonymous inner classes should define no instance variables and be limited to three single line methods. Inner classes that declare instance variables or have more complex methods should be named [mandatory].
43. Use final local variables where possible to help avoid errors in code [recommended]. For example:

```

public void foo()
{
    final int x = dataSource.getCount();
    // do things with x
    ...
}

```

Exceptions

This section gives general guidance on the use of exceptions when programming in Java.

44. try/catch blocks should be laid out like any other compound statement [mandatory]. For example:

```

try
{
    String str = someStrings[specifiedIndex];
}
catch (IndexOutOfBoundsException ex)
{
    // The user specified an incorrect index, better take
    // some remedial action.
}

```

4. When an exception is caught but ignored then a comment should be supplied explaining the rationale [mandatory]. For example:

```

try
{
    propertySet.setProperty("thingy", new Integer(10));
}
catch (UnknownPropertyException ignore)
{
    // This exception will never occur as "thingy" definitely exists
}

```

45. All exceptions that are likely to be thrown by a method should be documented, except if they are runtime exceptions (note: the compiler will not enforce catch blocks for runtimes even if they are mentioned in the throws clause) [mandatory]. For example:

```

/* Comment snippet:
 * @exception IllegalArgumentException Thrown if values is null or
 *      any of the integers it contains is null.
 */
private Integer sum(Integer[] values) throws IllegalArgumentException

```

Cpp Client Java Interop Issues

Issues affecting C++ client/Java broker/Java client interop build 6th Feb:

Open Issues

- <http://issues.apache.org/jira/browse/QPID-243> Inconsistent use of paths in #includes – this does not affect interop, will do after 0-9 branch merge
- <http://issues.apache.org/jira/browse/QPID-350> Broker infinite loop on restart with immediate messages
- Non-Apache JIRA - Problem with BDBStore queue recreation at startup - details:

```

Restarting our application, I sometimes get a "Protocol Error" exception and the following
message appearing in the broker log. This is even after restarting the broker when there
are absolutely no connections.
RECV: Frame[channel=1; ChannelClose: replyCode=405; replyText=Cannot declare queue, as
exclusive queue with same name declared on another connection [error code 405]; classId=50;
methodId=10]

I deleted the $QPID_WORK/<virtual-host>-store directory and I was able to restart.

Also, in the application code, the queue is created as shared (single param constructor) so
not sure why the broker thinks that it is exclusive?

```

Resolved Issues

- <http://issues.apache.org/jira/browse/QPID-353> Amend type of destination type from byte to int
- <http://issues.apache.org/jira/browse/QPID-349> Use the empty string as the default for virtual host name.

Java Broker Design

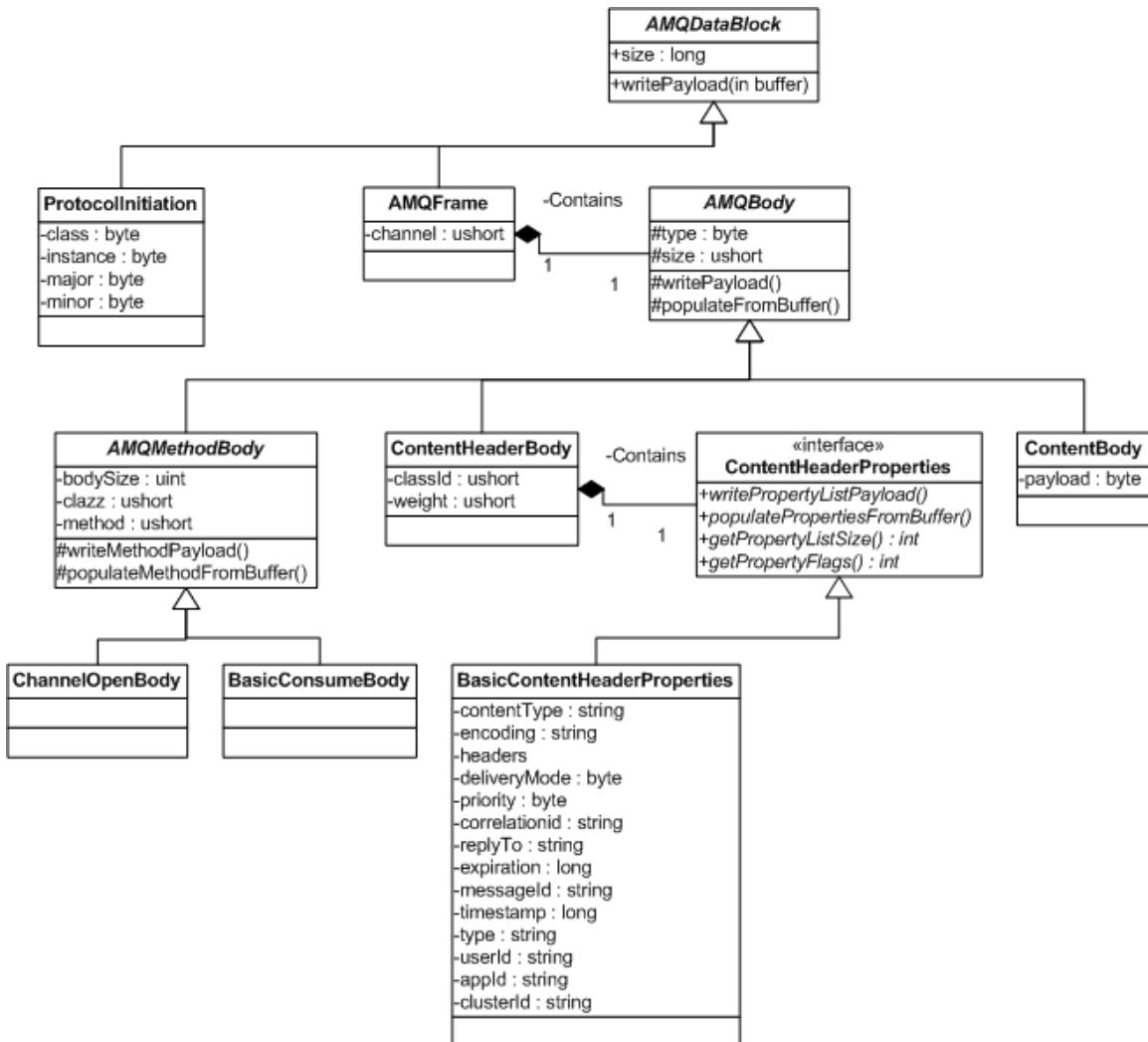
- Qpid Design - Framing
- Qpid Design - Management

- Qpid Design - Threading
- Qpid Design - Message Acknowledgement
- Java Broker Design - MessageStore
- Restructuring Java Broker and Client Design
- Message API Design
- Java Architecture Overview
- Producer flow control
- Java Broker Refactor (QPID-950)
- Java Broker Modularisation
- Java Broker Configuration Design
- Java Broker Design - Flow to Disk
- Java Broker Design - High Level Overview of Refactoring
- Java Broker Design - Message Representation
- Network IO Interface
- Java Broker Design - Operational Logging
- Qpid Design - Queue Implementation
- Qpid Design - Message Delivery
- Java authorization plugins
- 0.6 Broker BasicFlow Synchronisation Design
- Slow Consumer Disconnect
- Topic Configuration Design

Qpid Design - Framing

Frame Classes

The framing definition in the protocol specification maps quite nicely to an object-oriented representation. The class diagram is shown below:



The **AMQDataBlock** at the root of the hierarchy defines a `writePayload` method that subclasses implement in order to be able to transform themselves into bytes. This is called by the encoder, documented below. The decoding (from bytes into objects) is slightly more complex since it involves factories for the instantiation of the correct objects (again documented below).

An **AMQFrame** is the basic unit transmitted over the network, and contains a body which is the real payload. There are numerous method frames, which are subclasses of **AMQMethodBody**. The method body subclasses are all code generated from the protocol specification. The **ContentHeaderBody** can support different types of content properties or metadata (examples being file or stream in addition to *basic* which is

standard JMS-style messaging).

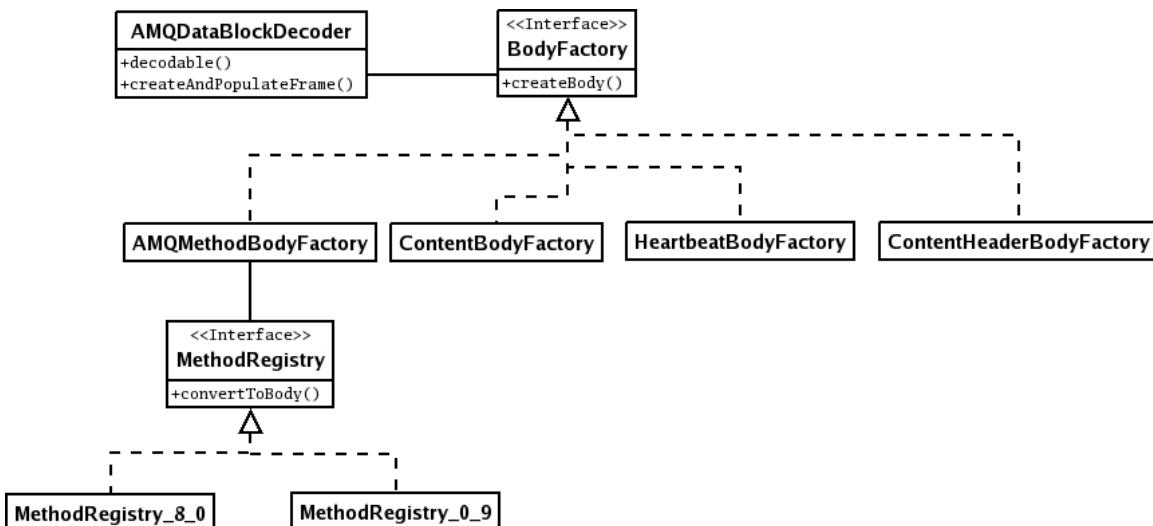
ContentBody is a lightweight wrapper for message data.

Encoding

Encoding is a straightforward process. The `AMQDataBlock` class has only two method: `getSize()` and `writePayloadToBuffer(ByteBuffer)`. The encoder simply needs to ask the data block its size, allocate a buffer of that size, then ask the data block to write itself into the buffer.

Decoding

The classes involved in decoding are illustrated in this UML class diagram:



The `AMQDataBlockDecoder` has only two methods: `decodable()` in which it attempts to read enough information from the supplied buffer to determine whether it has all the data and whether it appears to represent a known data block. If it needs more data, it return false. If the frame appears to be invalid it throws an exception.

The decoder stores the factories for `HeartbeatBody`, `ContentHeaderBody` and `ContentBody` frame types in an array, indexed on type. The `AMQMethodBody` factory is version specific and retrieved from the current session. The decoder constructs an `AMQFrame`, passing in the factory the appropriate factory. The result of that call is either a fully populated frame or an exception being thrown if data is invalid or inconsistent.

The `MethodBodyDecoderRegistry` is generated from the protocol XML. Each method is registered by protocol class and protocol method and when looked up by the `AMQMethodBodyFactory` an instance of the appropriate method body is returned. The generated code for the methods handles the reading and writing of the bytes to and from `ByteBuffers` as well as calculation of the size of the populated method bodies.

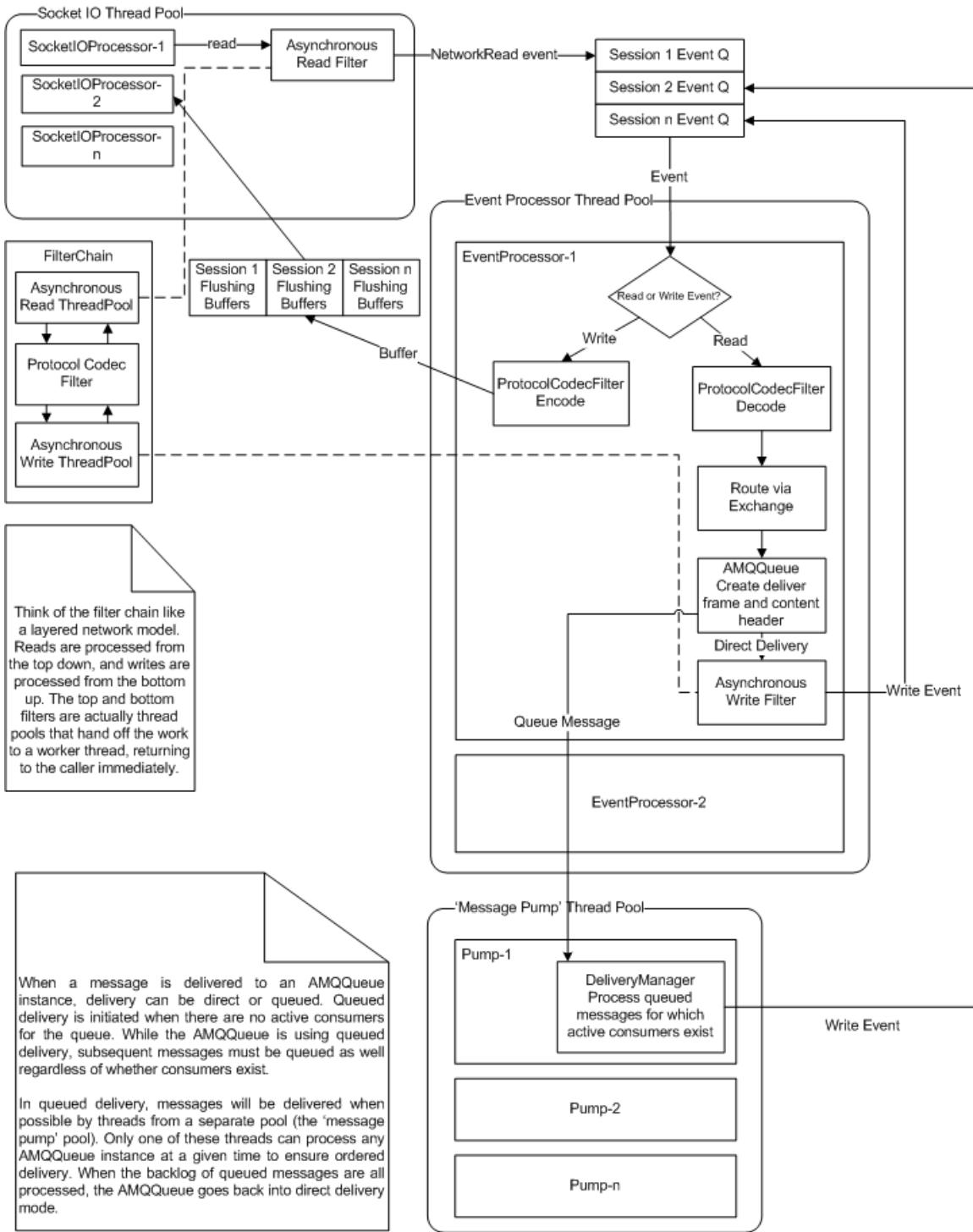
Qpid Design - Management

The broker makes several general JMX MBeans available for functionality such as User Management and Logging Management, as well as MBeans to allow management of each individual Connection, Exchange, and Queue. These are accessible via a JMX ConnectorServer started by the broker if its configuration calls for management to be enabled. This can then be used for management by a compatible JMX client.

For further details on configuring and accessing the management functionality, see [Qpid JMX Management Console](#) or [JConsole](#).

Qpid Design - Threading

The following diagram shows the threading model used in QPID:



Sessions

A session is the encapsulation of a client connection. A session has independent state associated with it.

Event Queues

The thread pool cannot simply be a generic pool that takes arbitrary work to process. Doing so would mean that no guarantees could be made for message ordering.

Each session has a single read event queue and it is populated by the single socket IO processor associated with the session. (Several sessions can be bound to a single IO processor; the standard select/poll mechanism is used to check for activity). Only one worker (event processing) thread can be processing the event queue for a given session but the particular thread can change over time. If more events come in while the queue is being processed, they are added to the queue being processed, and the worker thread only processes up to n events for fairness.

Similarly, for write events there is a separate queue that behaves analogously to the read queue.

One of the main benefits of this approach is that it allows enough parallelism while avoiding excessive context switching. The socket I/O processor reads as much data as it can - it does very little apart from polls and reads (side note: this makes it very straightforward to move to AIO if support is available). Message decoding (i.e. going from raw bytes to objects) and routing occurs in a worker thread but the entire

dispatch process - including encoding but excluding the socket I/O - occurs on a separate worker thread. This means that on a suitable SMP box the following activities can all take place in parallel:

- network reading and writing for a given session
- data decoding and routing
- response encoding

A further improvement would be to allow reading and writing in parallel by splitting that into separate IO processor threads, and this is being investigated (along with AIO).

Message Delivery

Messages delivered to an AMQQueue are delivered directly if possible (i.e. a write request is written to the consumers session by the thread processing the publish request). This reduces context switch or the overhead of adding and removing messages to a queue. However if there are no consumers then the message needs to be queued. In this case delivery will be done by a 'message pump' thread and direct delivery has to be stopped until the backlog of messages is processed in order to ensure that the ordering is not violated.

Qpid Design - Message Acknowledgement

Message Acknowledgements and Delivery Modes

When implementing the JMS client it became apparent that the JMS specification offered a considerable degree of latitude for interpreting the precise semantics of acknowledgement modes and it also did not cover all acknowledgement modes that are of interest.

Here we describe the precise semantics of the JMS acknowledgement modes and the additional modes that the JMS client provides.

In this discussion, "the client" refers to the JMS client implementation and "the user" refers to code that is part of the client application (i.e. code written by the end-user developer).

AUTO_ACKNOWLEDGE (JMS)

In this mode, the client acknowledges each message once it has been received by the user. In the case of an asynchronous message consumer, this means that an acknowledgement is sent once the onMessage method of a message listener has completed without throwing an exception of any sort. For a synchronous consumer, it means when the receive() method has returned the message to the user.

A single BasicAckBody is sent with the delivery tag of the message and the multiple flag set to false, acknowledging that message only.

CLIENT_ACKNOWLEDGE (JMS)

In this mode, the user acknowledges messages manually by calling the acknowledge() method on either the session or the message itself. These both have the same effect.

The JMS does not say how many message a client is allowed to receive before acknowledging. However, it does talk in vague terms about implementations making sure clients don't go too long without acknowledging to avoid resource exhaustion. Qpid uses the [Prefetch](#) value for this - the consumer must ack it's messages before it reaches this limit if it wants to receive any more.

Calling either Session.acknowledge or Message.acknowledge() acknowledges the receipt of all messages up to and including the current one.

DUPS_OK_ACKNOWLEDGE (JMS)

This mode is identical to AUTO_ACKNOWLEDGE from an implementation perspective, however the user application must be prepared to deal with duplicate messages.

PRE_ACKNOWLEDGE (non-JMS)

A mode not covered by the JMS specification is one where the client acknowledges a message before calling the onMessage() or receive() methods. It sends a BasicAcknowledge for each message as the message is passed to onMessage or receive() retrieves it. The semantics are therefore exactly the same as AUTO_ACKNOWLEDGE for receive(), but differ for onMessage() in that the message is acknowledged regardless of whether the method completes successfully or not.

The constant org.apache.qpid.jms.Session.PRE_ACKNOWLEDGE defines this mode.

NO_ACKNOWLEDGE (non-JMS)

Certain data may be time sensitive in the sense that redelivery is pointless - if the client cannot process it at the instant it is sent there is no point in redelivering it.

In this case, acks are redundant. Since TCP means that the server can be sure the client received the message the only problem could be client error.

Setting NO_ACKNOWLEDGE means that the client never sends a BasicAcknowledge and the broker removes the message from the queue as soon as it is sent.

The constant org.apache.qpid.jms.Session.NO_ACKNOWLEDGE defines this mode.

Delivery Modes

For message production, similar considerations apply. JMS defines two delivery modes, PERSISTENT and NON_PERSISTENT which allow

the implementor considerable freedom of implementation.

Unfortunately the JMS specification addresses what are really two separate reliability concerns with a single delivery mode.

The default delivery mode can be set on a producer. This can be overridden on each message sent.

PERSISTENT (JMS)

Persistent is the straightforward option. Messages are sent with the persistent flag set to true which means that they will be committed to stable storage.

NON_PERSISTENT (JMS)

Non persistent gives maximum performance with least guarantees. The persistent flag is set to false in each message which means that if the broker suffers an error it is neither required nor expected to recover those messages.

Java Broker Design - MessageStore

The MessageStore interface allows us to abstract the method by which the message data is stored within the broker.

Currently available implementations

- [BDBMessageStore \(3rd Party\)](#)
- [JDBCStore](#)
- [MemoryMessageStore](#)

BDBMessageStore (3rd Party)

BDBMessageStore

The BDBMessageStore module utilises the BerkelyDB to provide persistence. It is not part of the Apache Qpid distribution. However, it is listed here to provide Apache Qpid users with a complete list of available Storage Mechanisms.

Licensing

This module is available under GPL.

Download

The current build is available here: <http://rhm.et.redhat.com/download/>

JDBCStore

JDBCStore

This is a MessageStore that uses a JDBC connection as a backing store for the broker.

MemoryMessageStore

MemoryMessageStore

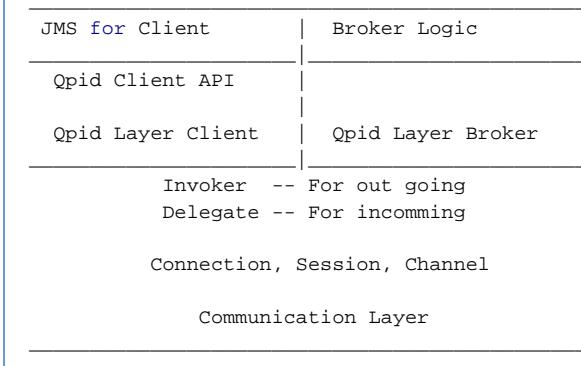
This is the default MemoryStore it performs no persistence between broker restarts.

Restructuring Java Broker and Client Design

Proposed Architecture

The following diagram depicts the architecture for the client and broker.

The idea is to auto generate and share as much code as possible between the broker and the client.



- The communication layer will be common to both client and broker.
- The Qpid Layer contains handlers that are both common to client and broker as well as Client and Server specific classes.
- The 3rd Layer is where the message processing logic will be implemented.
- For Client there will be a thin wrapper called Qpid API to mask around the Invoker and expose methods by class. The client will use the Delegate to handle incoming events.
 - This can be used to implement JMS on top.
 - Or application logic directly on top of the Qpid API.
- The Broker will build its logic around the Invoker and Delegate.

Invoker and Delegate

Invoker and Delegate are generated classes that contain all the methods in the spec.
To call a method you use the Invoker and to handle a method you use the Delegate.

Multi Version Support

To tackle multi version support the following strategy is used.

Struct interfaces are defined with a union of methods from different versions. And version specific struct factories will produce concrete structs for each version. All these classes are auto generated from the spec.

Ex:

```

interface Struct_A
{
    public getX(String s); // AMQP 0-10
    public getX(String s, int i); // AMQP 0-11

    public getY(); // AMQP 0-10
    public getZ(); // AMQP 0-11
}

```

```

Struct_A_v0_10 implements Struct_A
{
    public getX(String s){.... }
    public getY(){.... }
}

```

```

Struct_A_v0_11 implements Struct_A
{
    public getX(String s){.... }
    public getX(String s,int i){.... }
    public getY(){.... }
    public getZ(){.... }
}

```

Client code can still use v0-10 methods with a v0-11 library and compile as the Interface and existing methods have not changed. Changes are handled by adding the new or modified methods. Note this strategy is only envisaged for incremental changes. A major change in spec would need substantial code changes.

Qpid Client API

Is a thin wrapper around the Invoker plus a few convinience methods.
For example lets look at the Session class.

```
Invoker
{
    // All the spec methods
}
```

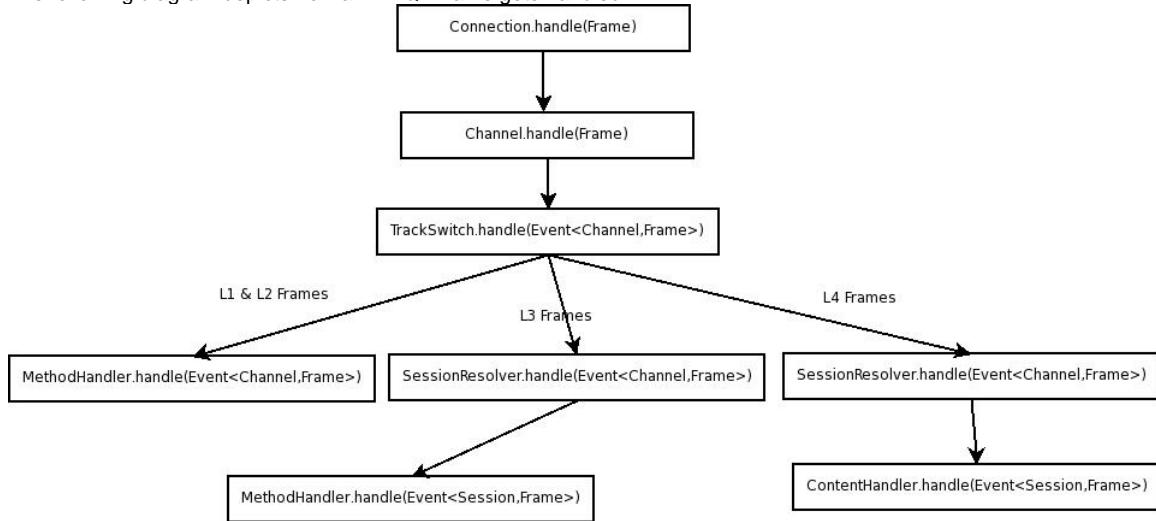
```
CommonSession extends Invoker
{
    // convinience methods for messaging
    header(Header h);
    data(byte[] src);
    endData();
    messageTransfer(String destination,Message msg);
}
```

```
ClientSession extends CommonSession implements Session
{
    // no implementation for 90% of the methods
    // Acts as a Mask for Session 'class' so only session specific methods are visible to the user.
}
```

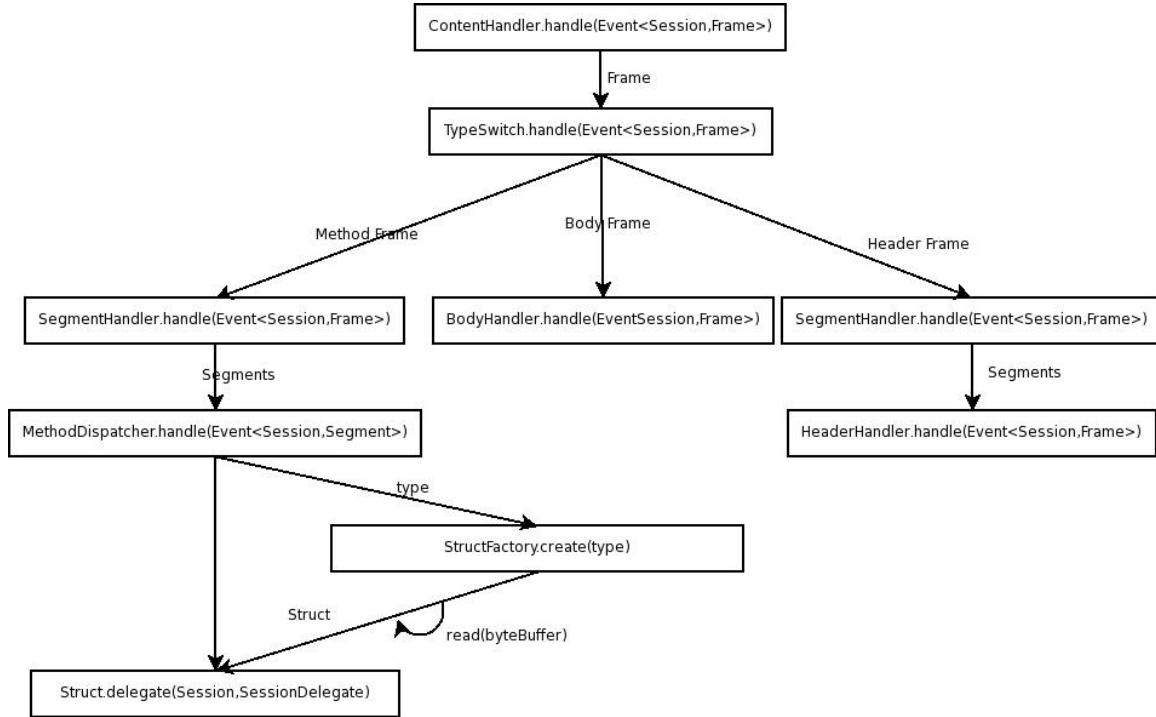
Communication Layer

Frame Dispatch Tree

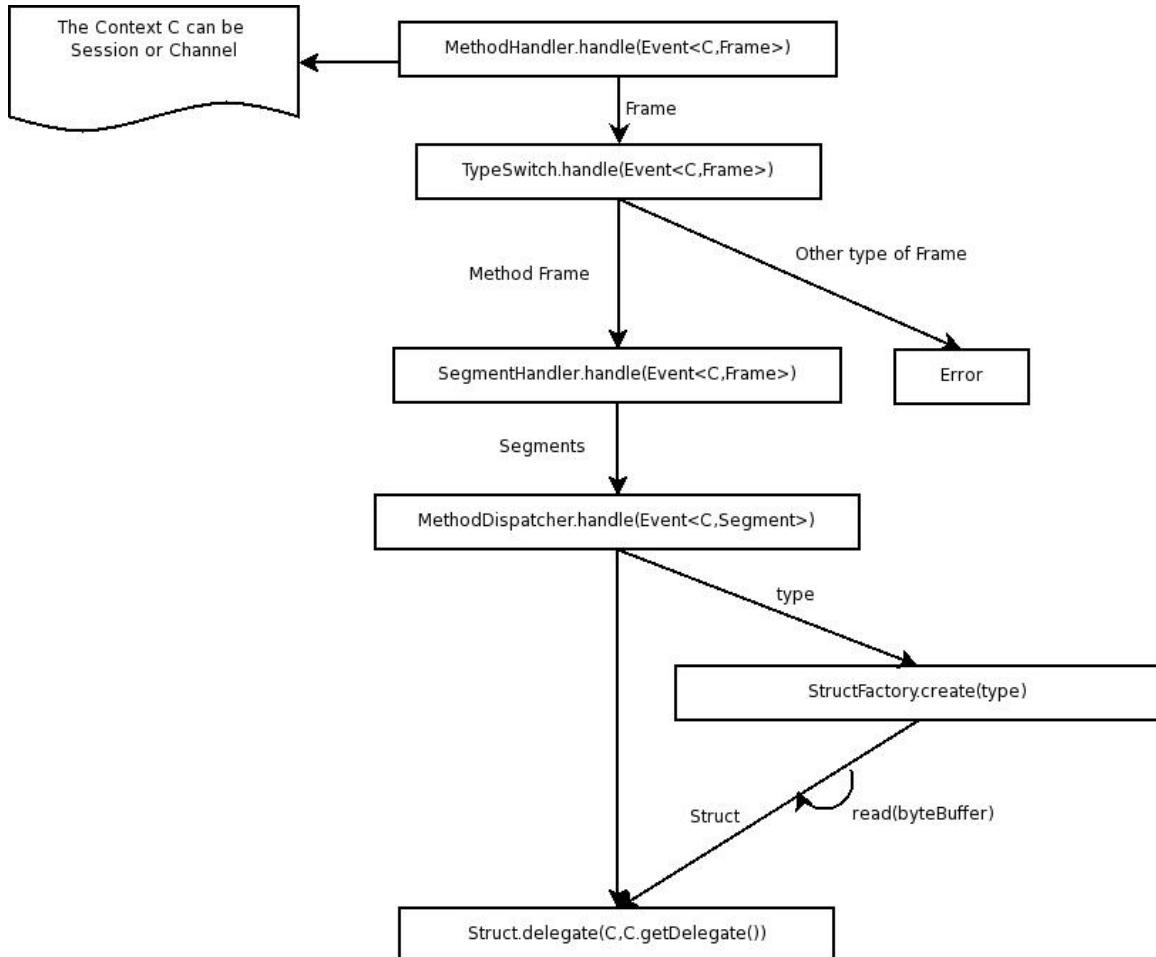
The following diagram depicts how an AMQP frame gets handled.



Content Handler Flow



Method Handler Flow



Message API Design

Message API Design

This document describes the new message API for the restructured client.

- Sending Messages
- Receiving Messages
- Message abstraction
- Java Doc

Sending Messages

The Session class provides the following methods to send messages.

```
public interface Session{
    .....

    //Option1 - for small messages
    public void messageTransfer(String destination, Message msg, short confirmMode, short acquireMode)
        throws IOException;

    //Option2 - for large messages
    public void messageStream(String destination, Message msg, short confirmMode, short acquireMode)
        throws IOException;

    //Option3 - can use it with any message size, recommended for large messages
    public void messageTransfer(String destination, short confirmMode, short acquireMode);
    public void headers(Struct... headers);
    public void data(byte[] data);
    public void data(ByteBuffer buf);
    public void data(String str);
    public void endData();

    .....

}
```

Sending small Messages

Option1 provides a convenience method to send small messages.

You could use the ByteBufferMessage to create small in memory messages (or your own implementation).
Underneath it maps onto methods defined under option3

Sending large Messages

You have two options for sending large messages, using either pull style or push style semantics

Using the Session class methods (Option3)

Option3 provides a more natural AMQP style set of methods

You can stream data using Option3 by pushing your data using one of the data methods defined in the session class.

Using Option2 (pull style)

The messageStream method will pull data from the message and stream using the methods defined in option3.

You could use FileMessage or StreamingMessage or your own Message implementation that backs a large data stream.

- FileMessage takes in a FileInputStream and create a.nio FileChannel. It then uses a MappedByteBuffer to map a region of the file when the readData method is invoked. You could specify a chunksize in the constructor to control how much data is mapped each time.
- StreamingMessage takes in a SocketChannel and reads a chunk of data at a time until the SocketChannel is closed. This could be useful when you need to transfer a data stream received from a legacy application or a hardware device. In such cases the StreamingMessage provides a convenient abstraction to stream the data without any intermediate copying.

Receiving Messages

To receive messages you can subscribe using the following method

```

public interface Session{
    .....

    public void messageSubscribe(String queue, String destination, short confirmMode, short
        acquireMode,
                                MessagePartListener listener, Map<String, ?> filter, Option...
        options);
    ----
}

```

The API provides support for receiving messages in parts as and when they arrive using the MessagePartListener. This enables the user to start consuming the message while it is being streamed.

```

public interface MessagePartListener{

    public void messageTransfer(long transferId);

    public void messageHeaders(Struct... headers);

    public void data(ByteBuffer src);

    public void messageReceived();

}

```

The messageTransfer method signals the start of a transfer and passes the transferId. The Transfer Id is used for the following operations defined in the Session API.

- to Acquire the message (if the message was transferred in no-acquire mode)
- to release the message (if already acquired)
- to Reject or Acknowledge the message

The data method will be called each time Frame arrives. The messageReceived method will signal the end of the message.

Consuming small messages

The API also provides a convenient way for consuming small messages through the MessageListener interface and the MessagePartListenerAdapter.

The MessagePartListenerAdapter will build the message and will notify the user through MessageListener when the message is complete.

```

public interface MessageListener{

    public void onMessage(Message message);

}

```

you can use it the following way.

```

.....
MessageListener myMessageListener .....

session.messageSubscribe(...,new MessagePartListenerDapter(myMessageListener),...);
----
```

Message abstraction

Message Interface provides an abstraction for creating messages from different data streams. Please read the java doc for a complete description of each method.

```

public interface Message{

    public MessageProperties getMessageProperties();
    public DeliveryProperties getDeliveryProperties();

    public void appendData(byte[] src) throws IOException;
    public void appendData(ByteBuffer src) throws IOException;

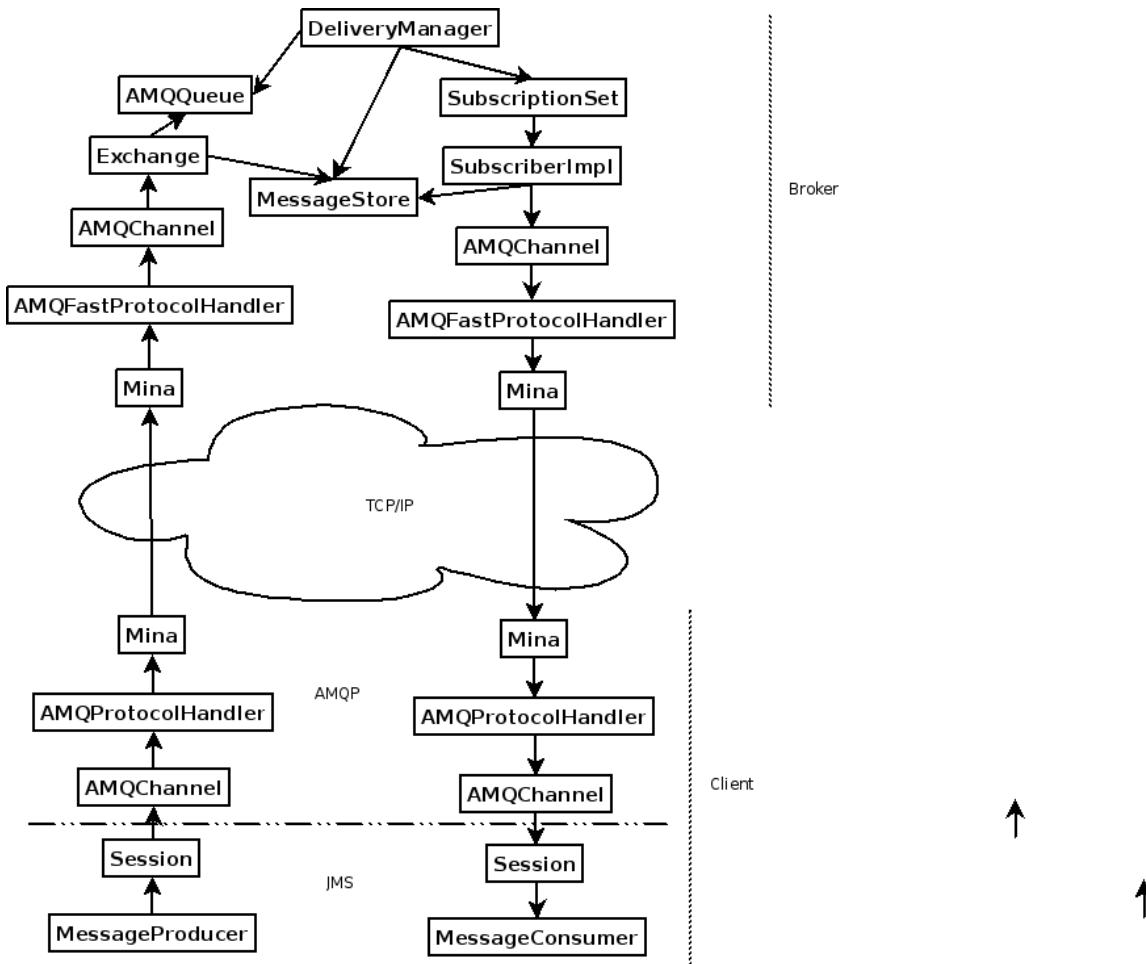
    public void readData(byte[] target) throws IOException;
    public ByteBuffer readData() throws IOException;

    public void clearData();
    public long getMessageTransferId();

}

```

Java Architecture Overview



Producer flow control

Producer flow control is necessary to stop clients from overwhelming a broker if messages are not being consumed fast enough, this is filed in Jira as QPID-942.

Use cases

1. Consuming Client lags Publishing Client (P2P)

Desc

This scenario is where the client consuming cannot process the data published to its queue by another client at the same rate i.e. consumption lags publication. This seems to be an almost de facto use case for P2P messaging in that by its nature (and possibly via the client code) consuming messages involves more processing than publishing i.e. `send()` is a less complex action than `receive()` or `onMessage()`. It can also happen when the consumer goes away.

Result

Message back up in the queues on the broker and are not being drained by the consumer. This may eventually lead to OOM as the VM cannot garbage collect the message refs. This could happen slowly over a period of time, so the MINA buffers may be empty (or at least not represent any significant amount of memory use).

2. Unconsumed messages remain in Queues (PubSub)

Desc

This is where data is being published to topics in the broker for which subscriptions exist, but no consuming client acks the messages.

Result

Messages back up in the broker and with durable subscriptions never go away. The broker OOMs are the queues are full. Again this happens over time and the MINA buffers may not be impacted. Without TTL set (and set low enough) then the data backs up in the client's sub queues.

3. Consuming Client cannot process large messages

Desc

This is where the consumer cannot process a large message sitting in its queue. This may be because it does not have enough memory or disk available for the processing, for example. It may also arise if the message is corrupted in some way i.e. malformed XML etc. The message does not get ack'd and remains in the broker on a queue, currently surviving restart.

Result

The message(s) remain in the broker and can cause OOM, particularly when there's a burst of large messages together. An example of this is the Qlib issue where they had a spate of large messages and client side OOM precluded them being processed. It can happen slowly, and with persistent messages costs at least twice as much heap. Broker OOM follows eventually.

Plan

To implement this, the following changes are necessary:

Client

send() needs to become potentially blocking, if the producer has been flow controlled then send() should not either throw an exception (which will be the default behaviour), or it will block until the producer has been unflowed (this will only occur if a system property has been set).

Broker

When a message has been enqueued, the broker should check if the producer is publishing to a queue which has violated its policy, if so then the producer will be flow controlled. When a consumer has had a message delivered, if the queue is no longer violating its policy then producers will be unflow controlled. This check will occur after enqueueing so as not to slow down the broker.

Queues and exchanges will have policies attached to them (queues will inherit from their exchange if they do not have one), which will specify the point at which producers should be flow controlled in terms of queue count or queue depth. These policies will be manageable over JMX, so they can be applied or removed without having to restart the broker.

The management console will also gain a "stop all producers" button to enable throttling of arbitrary queues, and a "start all producers" button which will start all flow controlled producers.

Disadvantages of this approach

The producer will not be flow controlled until they publish to a queue which is violating its policy, so if you have N producers each publishing to a queue, you will get N messages on top of the one that pushes the queue into a delinquent state.

Java Broker - AMQP0-9 Tactical Producer Flow Control

Problem Statement

The Java Broker currently performs no throttling of producing clients. In combination with the way that the Java Broker holds every transient message in memory until consumed, we can encounter scenarios where the Java broker runs out of heap space. For example, if a producer P sends messages at a rate of 100msg/s to a queue Q, but the only consumer, C, of queue Q processes messages at a rate of 10msg/s, then Q will grow at a rate of 90msg/s until such time as the broker runs out of heap space.

Tactically we may attempt to solve the problem of Queues becoming overfull, and thus causing out of memory exceptions, without attempting to solve the totality of out of memory issues.

Analysis

AMQP0-8/0-9/0-9-1 provides no mechanism for throttling producers of messages based on credit (either for a given destination, or even at the granularity of a session). There are two mechanisms available to throttle a producing client - the use of TCP flow control, and the use of the AMQP Channel.Flow command.

The use of TCP flow control throttles the producer to the rate at which the Broker can process the incoming messages, but does not address the throttling of the producer to the consumption of messages by a third part consuming client.

The Channel.Flow command instructs the recipient to either cease (or resume) sending messages. The receiver of the command should send Channel.Flow-ok once the flow command has been received.

In AMQP0-9-1 and earlier we cannot determine prior to a producer sending a message, which queues a producer wishes to send to. Thus we are limited in general to a reactive flow control - that is, when a producer attempts to send to an overful queue we can request that the

sender send no more messages, my issuing a Channel.Flow. Further, since many messages may already be "on-the-wire" by the time our Channel.Flow is received, we cannot guarantee by how much the producer may "overfill" the queue before it ceases publishing.

Proposal

Allow each queue on the Java Broker to be configured with a "full" size. Implement flow control such that the publisher of a message which is enqueued on a "full" queue is immediately sent a Channel.Flow command to cease publication. Monitor queue sizes such that when an "overfull" Queue has available space, then sessions which are blocked waiting for this event are free to send messages again.

Ensure that the Java Client respects the Channel.Flow command, and causes all attempts to send Messages to block, until the session is unflowed.

Design

Broker Changes

Add the following configurable properties to Queues:

capacity: size in bytes at which the queue is thought to be full (and thus publishes which send messages which take the total queue size above this mark will be blocked). Default 0 (no maximum)

flowResumeCapacity: the queue size at which producers are unflowed (defaulted to **capacity**)

Like other such values these may be set on individual queues in the config, or on a per-virtualhost basis.

Alter the following files in the org.apache.qpid.server.configuration package to set the queue properties based on this configuration:

VirtualHostConfiguration
QueueConfiguration
ServerConfiguration

Alter the AMQQueue.java interface to add the following method

```
/** Post enqueue check to ensure that the queue is not overfull. If the queue is overfull
 * then request the channel to begin flow control */

void checkCapacity(AMQChannel channel);
```

Update the following two classes in package org.apache.qpid.server.txn to call checkCapacity on the queue after they have enqueued a message

LocalTransactionalContext
NonTransactionalContext

Add the following code to AMQChannel in package org.apache.qpid.server

```

    /** The set of queues on which the session is currently blocking. Only a session blocking on
no queues can be unblocked */
    private final ConcurrentMap<AMQQueue, Boolean> _blockingQueues = new
ConcurrentHashMap<AMQQueue, Boolean>();

    /** Toggle to indicate whether the session is currently being blocked by an overfull queue
condition or not */
    private final AtomicBoolean _blocking = new AtomicBoolean(false);

    /** Add the given queue to the set of those which the session is blocking on (ignore if we are
already blocking on this queue)
        if this moves us from being unblocked to blocked, issue a flow command */
    public void block(AMQQueue queue)
    {
        if(_blockingQueues.putIfAbsent(queue, Boolean.TRUE) == null)
        {
            if(_blocking.compareAndSet(false,true))
            {
                flow(false);
            }
        }
    }

    /** Remove the given queue to the set of those which the session is blocking on (ignore if we
are no longer blocking on this queue)
        If this moves us from a blocking to an unblocked condition, allow client to resume
publishing by issuing a flow */
    public void unblock(AMQQueue queue)
    {
        if(_blockingQueues.remove(queue))
        {
            if(_blocking.compareAndSet(true,false))
            {
                flow(true);
            }
        }
    }

    /** Send a Channel.Flow command to the client */
    private void flow(boolean flow)
    {
        MethodRegistry methodRegistry = _session.getMethodRegistry();
        AMQMethodBody responseBody = methodRegistry.createChannelFlowBody(flow);
        _session.writeFrame(responseBody.generateFrame(_channelId));
    }
}

```

Modify SimpleAMQQueue to perform the capacity check, and also to unblock blocked channels when the queue reduces in size.

```

private final ConcurrentHashMap<AMQChannel, Boolean> _blockedChannels = new
ConcurrentHashMap<AMQChannel, Boolean>();

public void checkCapacity(AMQChannel channel)
{
    if(_capacity != 0L && _atomicQueueSize.get() > _capacity)
    {
        if(_blockedChannels.putIfAbsent(channel, Boolean.TRUE)==null)
        {
            channel.block(this);
        }

        // guard against race condition where consumer takes messages, decreasing queue size
        message
        // but not seeing that the queue was blocked so not issuing unblock
        if(_atomicQueueSize.get() <= _flowResumeCapacity)
        {
            channel.unblock(this);
            _blockedChannels.remove(channel);

        }
    }
}

private void decrementQueueSize(final QueueEntry entry)
{
    getAtomicQueueSize().addAndGet(-entry.getMessage().getSize());
    checkFreeCapacity();
}

private void checkFreeCapacity()
{
    if(_capacity != 0L && !_blockedChannels.isEmpty() && _atomicQueueSize.get() <=
_flowResumeCapacity)
    {
        for(AMQChannel c : _blockedChannels.keySet())
        {
            c.unblock(this);
            _blockedChannels.remove(c);
        }
    }
}

```

Client Changes

Hook in the existing handler for ChannelFlow commands by altering the dispatchChannelFlow method in the ClientMethodDispatcherImpl class

```

public boolean dispatchChannelFlow(ChannelFlowBody body, int channelId) throws AMQException
{
    _channelFlowMethodHandler.methodReceived(_session, body, channelId);
    return true;
}

```

Logging

Additionally logging messages should be emitted

- 1) on the broker each time the queue issues an overfull request to a session to start flow control
- 2) on the client every time it receives a flow control command from the broker
- 3) on the client every time it attempts to send a message but finds itself blocked by broker flow control - in particular this message should repeat periodically until the message is sent

Java Broker Refactor (QPID-950)

The attached doc describes the refactoring work done as part of QPID-950

Name	Size	Creator	Creation Date	Comment
 Java Broker Refactor.doc	124 kB	Rob Godfrey	Dec 02, 2008 06:47	

Java Broker Modularisation

In order to support multiple protocol versions and to provide greater flexibility, the Java broker should be better modularised than it is currently

Layers

Clean separation of responsibility into the following areas:

- I/O Layer - TCP, HTTP etc.
- Protocol Layer - AMQP 0-8/0-9, AMQP 1-0, potentially others
- Sessions - Active thread
- Model - Queues, Messages etc.
- Store - Two types of store, transaction log for recovery and disk store for flow to disk
- Broker internals - Message expiry, configuration file parsing, management etc.

There will be a thread pool which services Runnable entities in the broker internals and sessions.

The I/O layer, Protocol Layer and Model can all be considered passive entities, although they may have threads for their own purposes. They do not initiate actions within the overall broker.

Sessions

When a thread is allocated to a Session it will do one of three things: send things to the protocol layer for encoding, execute events from the protocol layer and pull messages from a queue. A session has 0 or more links to queues in the model, which it can create, send, receive, or drop, and a transaction which is in either not begun or in progress. Committing a transaction moves it to not begun.

Events

Events are read from the I/O layer by the protocol layer when asked for by a session.

A protocol event's handler is called by a Session which has a thread. The handler has access to the current transaction. It is responsible for interacting with the model, enqueueing messages, creating queues etc

Model

The model provides entities modelling version independent Queues, Messages and other entities which Protocol Events interact with, and which are maintained by the broker internals. The layout of model entities such as queues is likely to be stored in a SQL database for ease of tooling.

I/O layer

This is responsible for reading and writing byte streams and making them available to the protocol layer for decoding. See [Network IO Interface](#) for more details.

Stores

There are two types of store. There are transaction logs, which record events in a way which makes it possible to recover to a known state in the event of broker failure without losing messages. There are also on-disk stores which provide random-access retrieval of messages which are too large or too numerous to store in memory.

Protocol plugin

A protocol plugin provides a set of ProtocolHandlers which implement a particular version of the AMQP specification. They contain handlers for the events necessary, codecs for message from other protocol versions and model entities to implement protocol-specific behaviour.

For instance, the AMQP 0-8/0-9 plugin will contain a subclass of Queue which implements the Exchange behaviour for routing and support for immediate and mandatory message flags through dead letter queues and 0 span message TTLs. These queues would be 0 length and would have links to other queues which mapped to bindings. Messages would arrive at those queues and be moved onto the link by the queue subclass.

Java Broker Configuration Design

Java Broker Configuration Design

The Java broker configuration xml files have grown in complexity through M4 and reached a state where we need to look at the simplifying

the design to help both our users and our selves.

QPID-1612

Current Issues

1. We use an XML format but don't validate the XML.
2. We can't validate because we use dynamic tags in the Virtualhost sections.
3. We don't ensure that all configuration values are used in the code.
4. We have two three locations where virtualhost information is declared.
5. Information is duplicated rather than defaulted, such as Virtualhost store class.
6. Advanced configuration options are available and highlighted via the config file but we would never advocate their use.

Improvement Plans

As we are currently using Commons Configuration it would make sense to make use of as much of their code rather than writing our own.

Steps to improving design:

1. Centralise configuration into ServerConfiguration facade
 1. a. Start using a ConfigurationFactory
 - i. Allows us to split up our configuration moving the Virtualhost sections to its own file.
 - ii. Allows validation of the main configuration file.
 - iii. Allows an optional user-config.xml file to be included where users can easily override default values.
 - b. Redesign Virtualhost file to allow validation
 - i. Allow default values for all Virtualhosts
 - c. Investigate mechanism to allow plugins and broker elements to identify sections of configuration they use.
 - i. Fail to start broker if there are missing sections of the configuration.
 - ii. Fail to start broker if there are unused sections of the configuration.

Example for new config.xml that would work with Commons ConfigurationFactory:

```
<configuration>
    <system/>
    <xml fileName="${QPID_HOME}/etc/broker-environment.xml" validating="true"/>
    <xml fileName="${QPID_HOME}/etc/user-config.xml" optional="true" validating="true"/>
    <xml fileName="${QPID_HOME}/etc/previous-broker-config.xml" validating="true"/>
</configuration>
```

Java Broker Design - Flow to Disk

Flow to Disk Design

- Overview
 - Other Implementations
 - Current Functionality
- Design
 - Approach Overview
 - Approach Summary
 - Limitations
 - Future Enhancements
- Design Notes
 - Areas of Note
 - Areas for investigation
 - Validation Rules
 - Alerting
 - Testing
 - Message Flows
 - M4 Flows
 - Flows after Flow-To-Disk
- Design Details
 - AMQChannel
 - *TransactionLog* Recovery
 - AMQQueue/QueueEntry
 - Purger
 - Inhaler
 - Implementation details for *queue*Memory* and *flow* state updates
 - QueueBacking
 - BackingFormat

Overview

Currently, the Java Broker can do one of two things with a message it has to deliver:

1. Keep transient messages in memory until delivered
2. Write persistent messages to a message store (like BDB) and keep in memory until delivery complete or memory is full.

This means that the broker is not able to avoid OoM exceptions i.e. send enough messages to the broker, especially if your consumers are not active, and you could bring the broker down once it explodes its available heap.

This page pulls together the ideas from [QPID-949](#).

Other Implementations

Active MQ use the idea of a message cursor and have a number of different policies for performing 'Message Spooling' : [Message Cursors](#).

Current Functionality

Currently the broker treats persistent and transient messages differently. Persistent messages are written to disk as they are received and handles created as *WeakReferenceMessageHandles*. This means that when an Out of Memory(OoM) condition occurs then all the persistent message handles are GC'd. Performance hits the floor as all messages at the front of the queues must be read from disk whilst new messages are kept in memory but at the back of the queue.

Transient messages are created as *InMemoryMessageHandles* and so cannot be purged from memory. When an OoM condition occurs the broker cannot recover.

Design

There are areas of the broker that are in need of improvement that could be affected by this implementation:

1. *MessageStore* : Currently only persistent msgs are written here along with transactional data. The MS should become a *TransactionLog*_ so messages should not be retrieved from here for normal operation.
2. Message Reference Counting : To minimise message data duplication references are used to record how many queues the message has been enqueued on. This is currently maintained by the message but has been a large cause of runtime problems. If the *TransactionLog* maintains a list of Message,Queue Tuples then we can remove the error prone reference count integer that is currently used.

Approach Overview

The approach here is to reduce the overall complexity of the broker so that it is easier to reason about smaller chunks. Focusing at the level of a Queues would make life easier as we move towards AMQP 1-0. To facilitate this we should move to handling persistent and transient messages in a similar way. To this end the **MessageHandle* objects should be merged into *AMQMessage*. The current reference counting will be the responsibility of the new *TransactionLog*. The errors and testing of the reference counting is a tricky issue to tackle cleanly within the Java Broker. The queues will gain additional counts to ensure that it can track its memory usage based on the size of the message body + header + Object Content to represent the in memory data objects. This will allow the Queues to better reason and act upon their memory usage.

Approach Summary

1. Remove shared state from the *AMQMessage* class, and move everything into *QueueEntry*'s (this allows for a message to be flowed to disk on one queue while staying in memory on another).
2. Break apart the *MessageStore* interface creating a new *TransacitonLog* interface that covers only the logging of the durable transactional events (message data arriving, enqueuees, dequeues).
3. Move reference counting into the *TransactionLog*
At this point we will have removed our current (fragile) flow-to-disk capabilities on persistent messages... and all messages will be held in memory while live
4. Add a new properties to queues to keep track of memory usage.
5. Create *QueueBacking* to enable storage and retrieval of flowed messages.
6. Update *QueueEntries* / *AMQMessage* to use *QueueBacking* for, disk to disk
7. Add capabilities to queues to shrink their in memory profile by flowing queue-entries to disk (from the tail upward) until they are under a given notional memory usage.
8. Add check on message enqueue to ensure queue size does not grow beyond defined limit. Mark queue as flowed to disk when that occurs. Immediately flowing new messages on that queue, and potentially starting a *Purger* thread.
9. Add check on message send to potentially start an *Inhale* thread to restore flowed messages.
10. Add properties to *QueueDeclare* for flow to disk control extensions as defined in [C++ broker](#).

Limitations

The current design of the broker that utilises a new queue per topic subscriber does not lend itself well to this design. If a large number of subscribers fall behind due to high volume and start flowing to disk, then the amount of data flowing to disk will impare performance. A change in the broker to implement topics in a more AMQP 1-0 style where new subscribers are actually browsers on a special TopicQueue will alleviate the problem.

Future Enhancements

1. Enable the flow to disk of the queue structure. This will remove the final constraints on memory and only limit the broker to the amount of disk space available.
2. Dynamic sizing of queues. Policies for this sizing, i.e. fn(MessageSize,ConsumeRate) so slow consumers can have their queues flowed sooner with a smaller in memory cache. Whilst fast consumers with large messages would have a larger cache.
3. Conversion of Topic implementation to use an AMQP 1-0 style single queue where each consumer is a browser.

Design Notes

Areas of Note

- Initially a *Purger* thread will not be required if we are to simply let messages be deleted on ack. However, if we do not have a small prefetch on the client then we will quickly OOM the broker. Ensuring the prefetch is set to a more sensible value (<100) is important here.
- Priority Queue implementation will be more tricky as *IncomingMessage* may be destined for the front of the queue unlike the non-priority case where new messages go at the end. The implementation for this will require a *Purger* thread.
- The *queueMinMemory* value will be initially a fixed value or at least a calculation based on *queueMaxMemory*. However, it should be implemented with the future prospect of using a dynamic sizing method.
- Care must be taken for the NO_ACK mode as the dequeue is performed **before** delivery so the message must be in memory before that occurs or the data will be lost.

Areas for investigation

- When to perform an fsync?
- Will NFS be capable of supporting the backing.
 - Sizing guidelines for users, What is the overhead/message on disk
- Raise awareness that the data will now end up on disk. If it is confidential it should be sent encrypted.

Validation Rules

- What checks to run on start-up
 - Defined Queues *queueMaxMemory* sum to less than X * Xmx value.
- How to calculate default values

Alerting

- New Alerts when Queue flows and recovers
- Suggestion to set *queueDepth* less than *queueMaxMemory* to get early warning that flow-to-disk may occur.

Testing

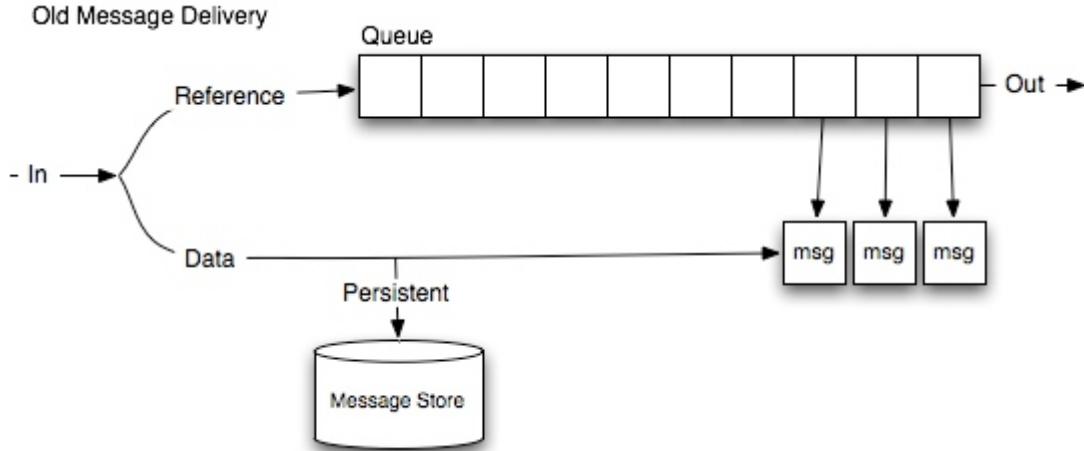
- What happens when the disk fills up.
- Currently the Java Brokers behaviour with WeakReferences is quite poor after it has dropped all the references.
 - Generate test that models the current behaviour and ensure that the new flow-to-disk improves the performance.

Message Flows

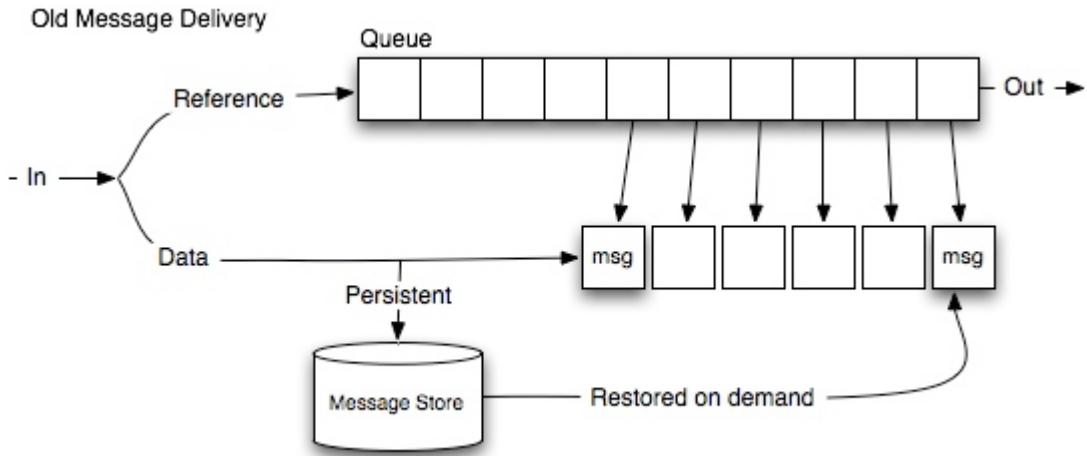
To highlight how these changes will affect the current message delivery flow what follows are a series of message flow diagrams to highlight the changes:

M4 Flows

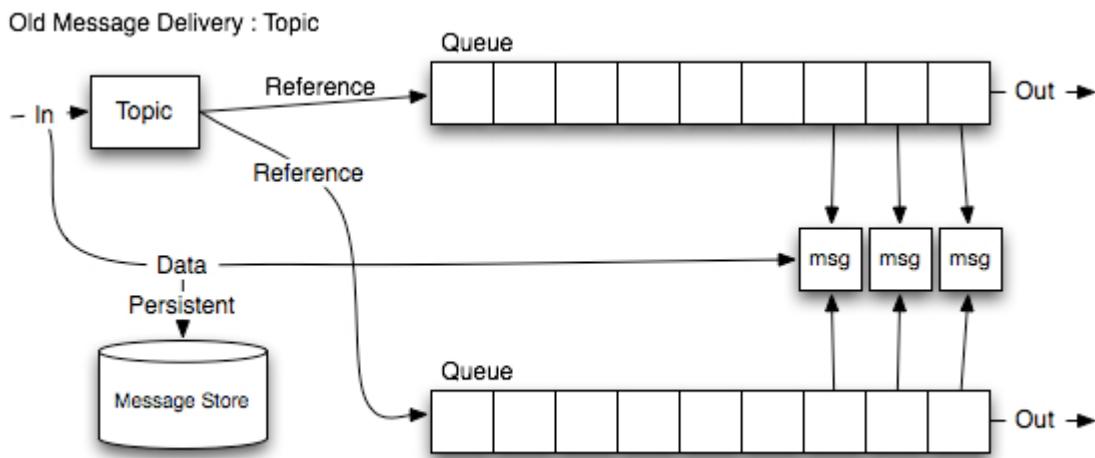
This is what message delivery via a queue would look like in the M4 broker. Messages are received as an *IncomingMessage* and the data if persistent is written to the *MessageStore*. The message is fully received it becomes an *AMQMessage* and is then routed to a Queue where a *QueueEntry* is created that references that *AMQMessage*.



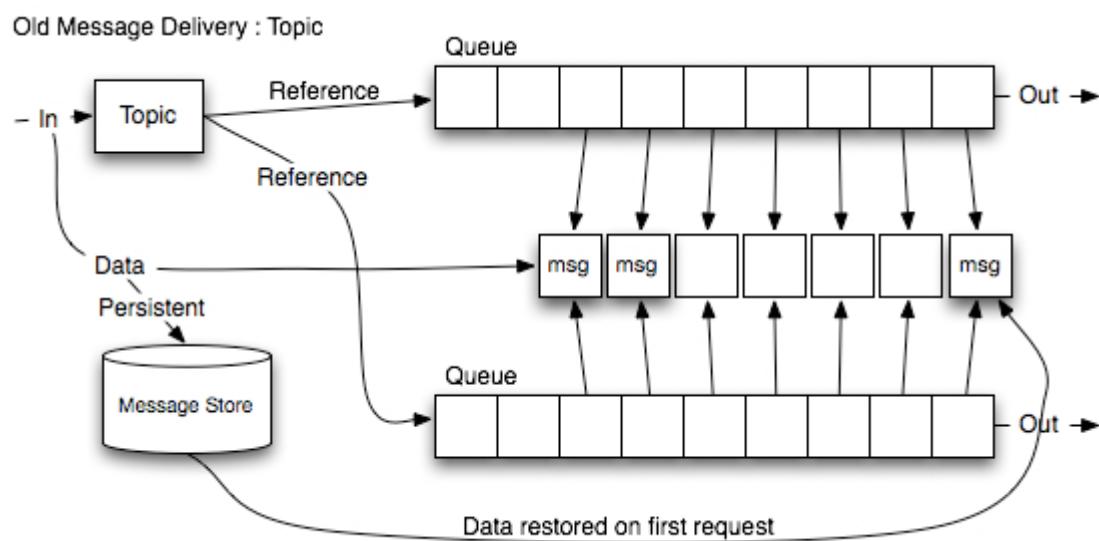
In the case of persistent message delivery the message data is held via a WeakReference. So when the broker reaches a full memory scenario it will purge **ALL** the references to the message data. New messages received by the broker are kept in memory but for message delivery the message must be fetched from the store which may be an expensive operation.



The situation is slightly better for Topics. An `AMQMessage` that is routed to multiple Queues can be referenced by many `QueueEntries` so duplication of memory is not required.

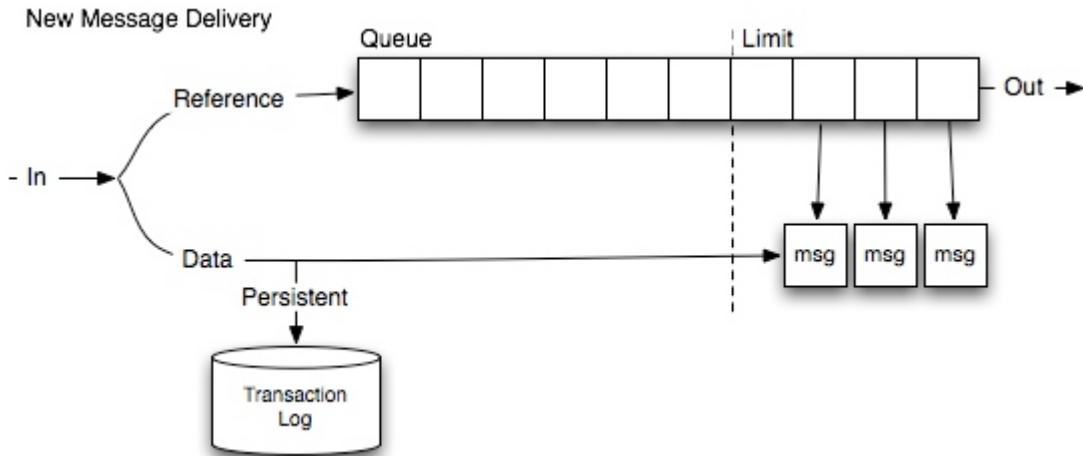


When a full memory situation occurs then as with the Queue case message data is purged. However, the references continue to be shared and an unchecked race to repopulate the message data occurs. No data loss should occur in the race only duplicated effort in reading from disk twice.

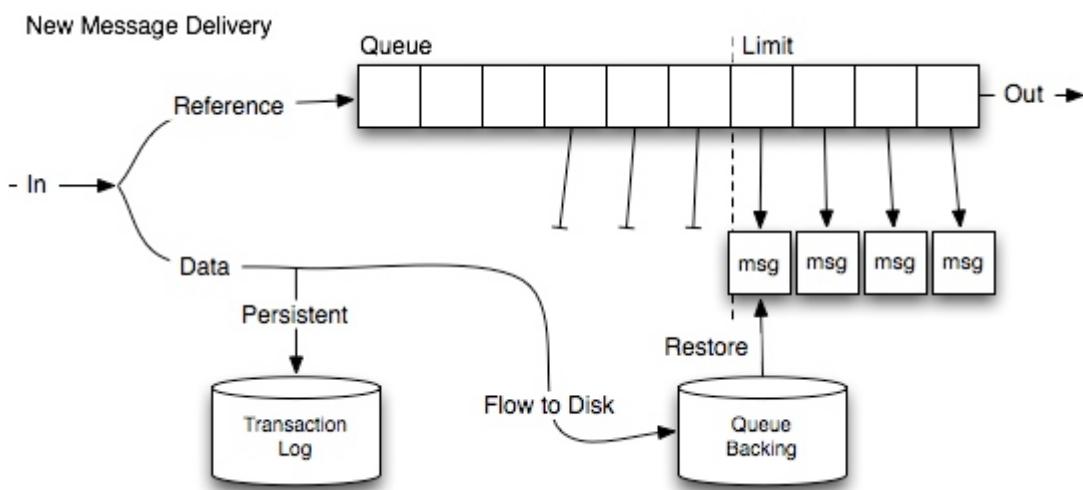


Flows after Flow-To-Disk

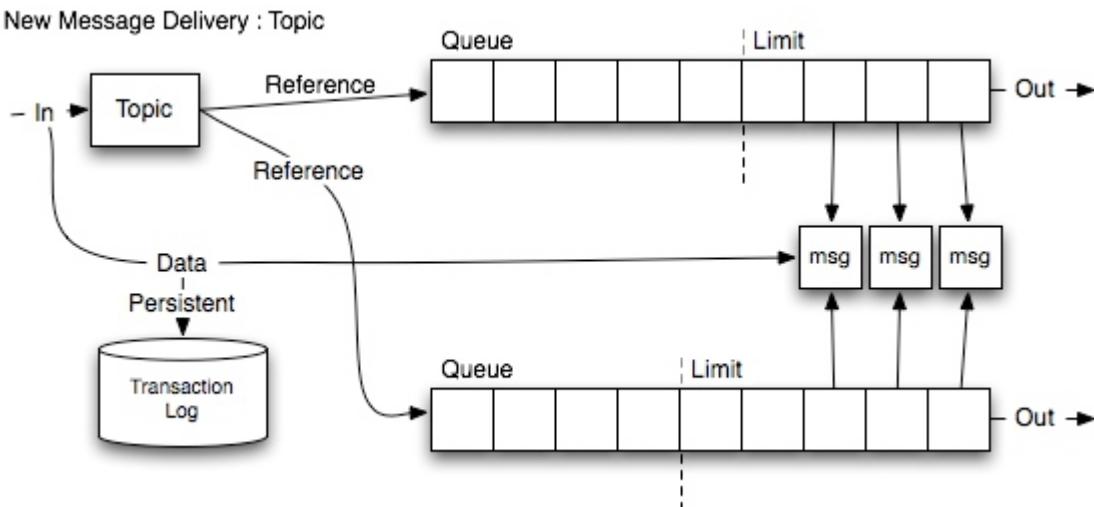
This design for Flow-To-Disk aims not to impact the normal message delivery flow. Here you can see that persistent data is written to the `TransactionLog` as it was in the `MessageStore` for M4.



The change occurs when the queue hits its defined memory limit. At this point **ALL** new messages to that queue are written to the QueueBacking. An *Inhaler* process ensures that the subscribers do not starve for data that has been flowed.

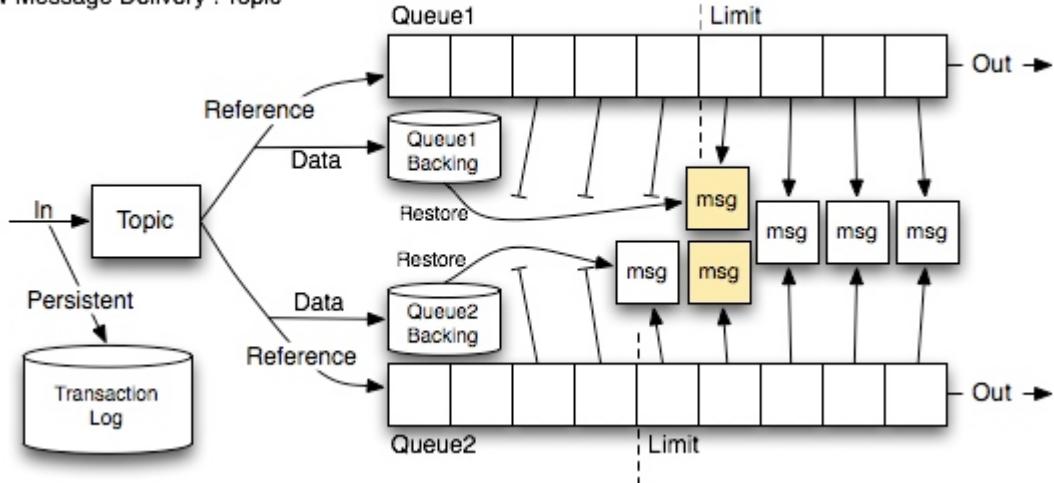


The topic scenario is slightly altered by the desire to keep each queue responsible for its own message flow to disk. *AMQMessage* references can be shared between queues in normal delivery but when a message is flowed to disk for a give queue that queue will restore the message only for its consumption.



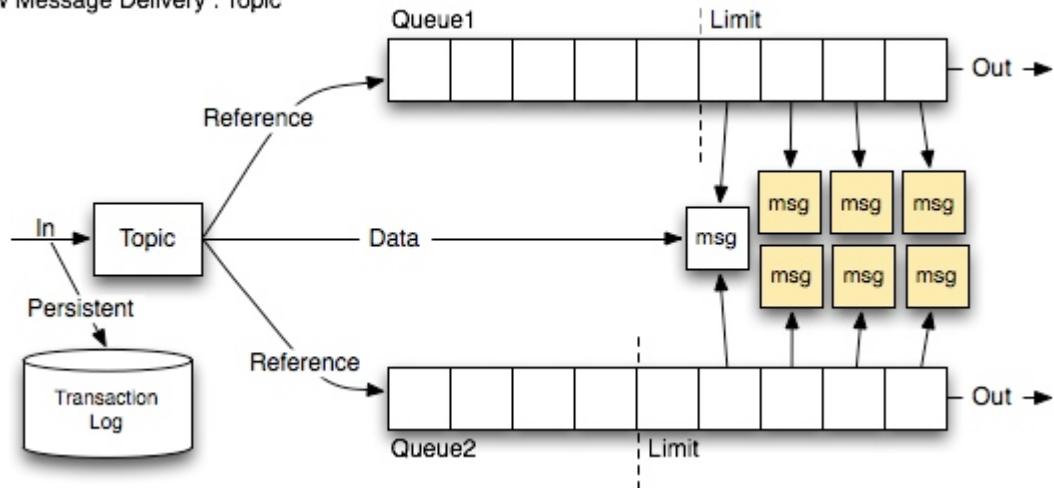
The orange messages highlight the duplication of message data in memory. This duplication vastly simplifies the responsibilities for flowing to and from disk.

New Message Delivery : Topic



Of course when the queue leaves a flowed state newly received message data can again be shared between queues. Again note that the orange messages are pair vertically to represent the duplicated data from messages that have been restored from disk.

New Message Delivery : Topic



Design Details

To highlight the changes that will be required lets look at the processing that is performed on an incoming message:

AMQChannel

When a persistent message is received the headers and chunks are recorded in the new *TransactionLog*, the remainder of the current *MessageStore* will be moved to a new *RoutingTable* interface.

```

++ TransactionLog ++
| enqueue
| dequeue
| storeHeader
| storeChunk
|
| startTransaction
| commit
| abort
+-----+
++ RoutingTable ++
| createQueue
| createExchange
| createBinding
|
| deleteQueue
| deleteExchange
| deleteBinding
+-----+

```

The *TransactionLog* is a distilled version of the current *MessageStore* interfaces. The log is the persistent record of the state of the broker. On start up this log is used to restore the routing and message states. It is not to be used as a lookup mechanism, the queue's must now be responsible for remembering all the enqueue messages and not rely on the previous *MessageStore*. As no random access to the log file is needed it can be implemented as a write ahead log. It can periodically cleanup the old state by writing a new log but as its primary function is to ensure state is persisted to disk it need not maintain maps of the data thus simplifying its implementation. The responsibility for remembering the message data is delegated to the Queue. The *TransactionLog* shall absorb the current reference counting code and be responsible for deciding when to recoverably delete a message. Currently the reference counting is still spread across a number of different classes and has a couple of serious problems. The *TransactionLog* will record a series of Queue/Message tuples so that it can pair enqueue/dequeue calls. When there are no more references to the message then it can safely know that the message is no longer needed. By using a list of tuples rather than an integer count the *TransactionLog* is capable of safely interleaving transactions as there is no shared count value.

TransactionLog Recovery

Currently the *MessageStore* is responsible for providing unique MessageIDs, this is not strictly part of a *TransactionLog* as a result it would not make sense to include it in the interface. What is recommended is that we unify our message creation as part of removing the **MessageHandle* objects. Messages recovered directly from the store currently create the **MessageHandle* directly with a Message ID; while message delivered via the wire ask the *MessageStore* for an ID before creating an *IncomingMessage* which in turn creates the **MessageHandle*. As we will be removing the **MessageHandle* objects it makes sense to unify our message creation through a Factory *MessageFactory*. This will allow the factory to be responsible for the sequence of IDs. When recovery is in progress a call to *createMessage(id)* will take place and the factory need only:

1. ensure the id is unique
2. record the highest value seen to seed its sequence of IDs handed out by *createMessage()*

```
+--MessageFactory---+
| createMessage(id) |
| createMessage()   |
+-----+
```

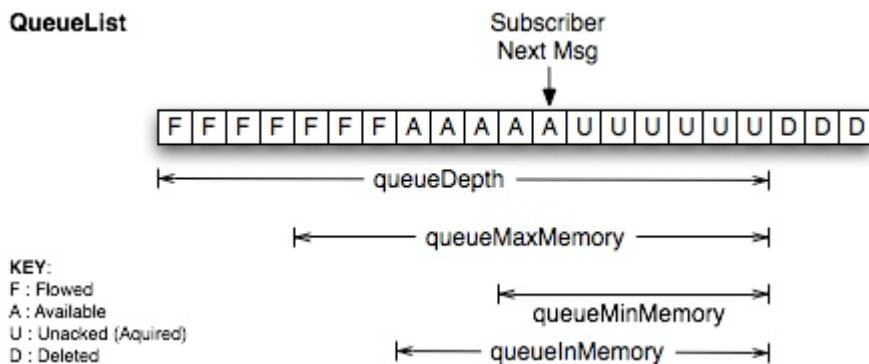
AMQQueue/QueueEntry

When a message has been fully received it is then routed to the required Queues as before. Only persistent messages that are routed to persistent queues are written to the *TransactionLog* which is then responsible for the ultimate deletion of persistent message data.

For this to occur the existing model needs to be updated. The **Handle* objects we currently have need to be merged in to *AMQMessage* and all the state about the message needs to be moved to the *QueueEntry*. This will allow us to null the *AMQMessage* reference as and when the message is flowed to disk. The *QueueEntry* interface will be augmented to allow the Queue to flow the data when required. When the data is recovered then no attempt is made to restore the single instance of the message. i.e. If a single message is sent to 10 queues initially there will be one *AMQMessage* and one copy of the data. When a queue is flowed then it will lose the reference to that message so on recover a new message with a copy of the data will be created solely for that queues use.

```
+--QueueEntry---+
| flow          |
| isFlowed      |
| recover       |
+-----+
```

Our existing Queue needs to be updated to be able to record the additional state of the *QueueEntry*'s. Currently we have *queueCount* and *queueSize* that represent the count and data size used by the queue. The Queue needs to have additional *queueMaxMemory*, *queueMinMemory*, *queueInMemory* and *isFlowed* added. It is proposed that only the data size is used for flow to disk calculations as counting messages will not give us the control that we need over memory usage.



These new variables will be used to control two new threads *Inhaler* and *Purger*.

Purger

When *queueMaxMemory* is reached the queue is set to flow and all new messages on to the queue are sent straight to disk. As messages are sent to a subscriber there are a couple of possibilities when the queue is in a flowed state:

1. The messages are also flowed to disk.

2. A number or percentage of the `queueMaxMemory` could be kept to handle rollbacks.

Using the first mechanism we do not need to have a `Purger` thread for the simple queue case. However for the second case and to handle queues where the position of the incoming message is not known then the `Purger` thread will be required. The `Purger` simply needs to start at the front of the queue and record the amount of data still in memory on the Queue when `queueMaxMemory` is reached all subsequent messages are flowed.

Inhaler

The `Inhaler` is an optimisation to ensure that the broker returns to peek performance after a flow to disk event. Lazily loading messages on demand would be quite slow; so on delivery to a subscriber a check can be performed to see if the current `queueInMemory` is less than the `queueMinMemory` which would indicate that there is room to reload older messages. The `Inhaler` can then begin to load messages from disk until `queueMaxMemory` has been reached or all messages are back in memory. If all the messages are back in memory then the queue flow state can be reset allowing incoming messages to stay in memory.

NOTE: as there are no locks a second check by the `Inhaler` is required to ensure a message was not flowed between the last load and the change of queue state.

The updates to delivery to the queue and to the subscriber are expected to be updated in the following ways:

```
Pseudo-Code - Delivery to Subscriber
while (message in queue)
    subscriber.deliver(message)

// With a low prefetch or an more complex purging thread this should not be required.
if (flowed)
    flowToDisk(message)

if (_queueInMemory < _queueMinMemory)
    startInhaler
```

```
Pseudo-Code - Delivery to Queue
addToQueue(message)

if (flowed)
    flowToDisk(message)
else
    if (_queueInMemory > _queueMaxMemory)
        setFlowed
        startPurger
```

The additional overhead of checking state is done after the message deliveries have been performed and are simple calculations compared to the existing message flow paths. As a result the non-flowed state performance should not be affected.

Implementation details for `queue*Memory` and `flow` state updates

Here are some further details on how the new `queue*Memory` values will be calculated.

Implementation details for Delivery to Subscriber, this is taken from `SimpleAMQQueue` and is also used by `AMQPPriorityQueue`.

```
public void dequeue(StoreContext storeContext, QueueEntry entry) throws FailedDequeueException
{
    // Current _queueDepth calcualtions
    decrementQueueCount();
    decrementQueueSize(entry);

    // Add update for _queueInMemory_
    decrementQueueInMemorySize(entry);

    ... snip complete entry clean up, including any TransactionLog dequeue and QueueBacking delete
    ...

    if (_queueInMemory.get < _queueMinMemory.get())
    {
        _asyncInhaler.execute(inhaler);
    }
}
```

When the `Inhaler` has completely restored all messages to the queue it can call `setFlowed(false)` to continue normal message delivery. **NOTE** : It is expected that the `Inhaler` will have to run once more over the queue to ensure that no new messages were flowed between it retrieving all messages on disk and the resetting of the flowed status.

Implementation details for Delivery to Queue, again this is taken from *SimpleAMQQueue* and is also used by *AMQPPriorityQueue*.

```
public QueueEntry enqueue(StoreContext storeContext, AMQMessage message) throws AMQException
{
    // Current _queueDepth calcualations
    incrementQueueCount();
    incrementQueueSize(message);

    // Add update for _queueInMemory_
    incrementQueueInMemorySize(message);

    // Add check to see if we should be flowing.
    if (_queueInMemory.get() > _queueMaxMemory.get())
    {
        setFlowed(true);
        // if we are to have a purging thread then this is where it would be started.
        // _asyncPurger.execute(purger);
    }

    ... snip section about adding to queue and attempting deliveries ...

    // For the SimpleAMQQueue case messageAddedToEndOfQueue() will always return true.
    // The AMQPpriorityQueue implementation will be more complicated.
    if ( _flowed.get && messageAddedToEndOfQueue() )
    {
        entry.flow();
    }
}
```

QueueBacking

The flow to disk implementation will be performed on a queue by queue basis so a new *QueueBacking* will be created to handle the flowing of messages to and from disk.

```
+--QueueBacking--+
| flow      |
| recover   |
| delete   |
+-----+
```

When a message is dequeue then it must also be removed from *QueueBacking*.

NOTE: care must be taken here for the NO_ACK mode as the dequeue is performed **before** delivery so the message must be in memory before that occurs or the data will be lost.

BackingFormat

The initial implementation of the *QueueBacking* will be *FileQueueBacking*. This will need a new configuration parameter 'flow-to-disk-path' it will default to '\$QPID_WORK/queueBacking'. In this directory a new directory will be created to match the queue name that this backing represents. It is in this directory that the queue contents will be written. Each message will be written to its own file which will include the header and body. So the resulting file structure will be as follows:

```
QPID_WORK/
    /queueBacking/
        /<queueName>/
            /<bin>/<msgID>
```

Whilst NTFS can store over 4M files per directory, FAT32 is limited to 65534 and ext3 only 32000. That coupled with the fact that looking up a file in a large directory is not efficient it makes most sense to implement a hash map on disk using the MessageID as the key. Using the least significant 8 bits as the hashing function will give us 512 bins to evenly spread the messages on disk. This approach will improve look ups times and allow us to write over 16M messages per queue before we hit any file system limits.

FtD Code Review Notes

Flow to Disk review points

ID	Priority (H/M/L)	Review	Status
16	0.5 H	is load() thread safe	Done

13	0.5 H	getMessage contract broken MUST NEVER return null	Done
14	0.5 H	getMessage should do a load, remember gap between load and purger	Done
15	0.5 H	document reuse of setDelivered	Done
19	0.5 H	_priorityListsIndex.memoryUsed() + requiredSize is not threadSafe. Currently hard to reason about.	Removed
21	0.5 H	PriorityQueueEntryList add() not thread safe. Reclaiming memory and then setting later is not atomic.	Removed
24	0.5 H	setMemoryUsageMaximum / setMemoryUsageMinimum : not ThreadSafe - synchronize	Removed
17	0.5 H	Document atomicity of memory counting	Removed
18	0.5 H	Document test cases , plausible , implausible	
1	0.5 M	Removal of old get* Methods from TransactionLog	
2	0.5 M	Create Abstract BaseTransactionLog class to hold commonalities with existing TLogs	
3	0.5 M	Refactor Ref Counting out into BaseTransactionLog	
12	0.5 M	Linux ext3 means 31998 queues max per vhost per instance. as the dir is currently created even if it is not needed.	
22	0.5 M	PriorityQueueEntryList _disabled -> move to isFlowed and name it better as the queue is not disabled. Only FtD is.	Done
27	0.5 M	VirtualHost:L208 : initialiseRoutingTable(hostConfig) should be done when transactionLog != RoutingTable	
31	0.5 M	BDBMS : Enqueue has no rollback operation.	
8	0.5	FlowableQueueEntryList extends QueueEntryList, but there are no unflowable lists, and there shouldn't be in the future either. (AS)	Done
20	0.5	Comments would help	
23		non-atomic get Methods in PriorityQueueEntryList	
29	0.5	BDBMS : _dequeueTxMap should move to StoreContext to remove synchronized	
32	0.5	BDBMS : StoreContext needs enqueue/dequeue added.	
7	0.5	Rename flow/recover -> unload/load	Done
9	0.5 NTH	Extract flow strategy in to a separate interface	
28		BDBMS : linked list per message is not memory conserving	
30		BDBMS : In simple case where there is only one copy of message can do delete in same transactions	
4	0.6	StoreContext update, initially to include the dequeue messageIDs as per BDBStore	
5	0.6	StoreContext -> convert -> Transaction and use that for operation.(commit,abort...)	
6	0.6	Reference Count to use AtomicInts to reduce memory usage.	
10	0.6	QueueEntryImpl allows direct access to the ContentHeader via the Filterable Interface, need an improved way of doing selectors that doesn't always require the message to be pulled from disk. Some checks can be done with data already in memory with QueueEntry; MessageID	
11	0.6	NoLocal requires message to be pulled in to memory.	
25	JIRA	Need to be better at stop(), decide what we actually want to do on AMQQueue.stop().	
26	JIRA	TransactionLog, RoutingTable interfaces separated but not actually split out the implementations.	

Testing

ID	Priority (H/M/L)	Test	Status
1		What happens when the directory cannot hold any more messages, or queues.	

2		Test flowed queues delete backing store on close.	
3		Management Console Functionality, Viewing, Moving Messages	
4		Consuming from flowed queues with all ack modes : Client, Transacted, No-Ack	
5		Browsers with selectors.	DONE
6		Selectors on normal consumer to be completed	
7		Test No_local on flowed queues : Currently flowed messages will be reloaded to check no_local values. Does it make sense to move them to the QueueEntry?	
8		Failure testing: What happens when disk runs out?	
9		Failure testing: Current implementation should log error and keep message in memory. Eventually though it will OOME the broker as it can't keep messages in memory.	
10		Testing on all supported OSes and File systems. Linux : ext3, SAN(vxfs); Solaris (ZFS); Windows NTFS?	
11		What happens if the disk fails/is removed?	
12	0.5	Selector test, 1..4 consumers consuming message ids by chunk of 1000. Fill half then start consuming with consumer 4. Then send final half. How does it performs?	

Still to be discussed
Priority Queues
RoutingTable

Java Broker Design - High Level Overview of Refactoring

- Overview
- Areas of investigation
- Message Representation
 - Summary
 - Approach
- Storage (Message & Model)
 - Summary
 - Approach
 - MessageStore and TransactionLog
 - Model Configuration
- Message Transfer
 - Summary
 - Approach

Overview

There are a number of areas of the current Java broker that we would like to look to improve. The primary aims of this work are to simplify the various modules of the broker and clearly identify their interactions according to the identified [areas of responsibility](#).

Where this design refers to the 'current' code base, this is the 0.5 release and the state of trunk at this point. Work completed previously on [Flow to Disk](#) is being put to one side for the purpose of this design.

Areas of investigation

We have identified three main areas of to improve.

1. Message Representation
2. Storage (Message & Model)
3. Message Transfer

Message Representation

Summary

Currently the broker uses two implementations of AMQMessageHandle, one transient, one persistent, which actively stores and retrieves the underlying data. The AMQMessage class should provide immutable access to the message data (header and body) stored within the broker, there should be a 1:1 relationship between AMQMessage objects and messages delivered through the broker.

Currently the AMQMessage is responsible for holding a count of references but the responsibility for maintaining this value is spread throughout the code base. This responsibility needs to be given to a single class so that we can more easily reason about and test its functionality.

Approach

Adjusting the existing AMQMessage to ensure that the data it stores is immutable will require moving queue specific data such as Redelivered to the QueueEntry. Converting the existing MessageHandles to a single handle will allow us to easily unload the data as required for Flow to Disk. The additional storage work that was done in the persistent message case will need to move to an object in the Message Transfer layer.

Taking our existing single AMQMessage object we can delegate the responsibility to it for maintaining the reference count. This will prevent any need for a globally locked map and any synchronisation will be reduced to the level of the message.

Storage (Message & Model)

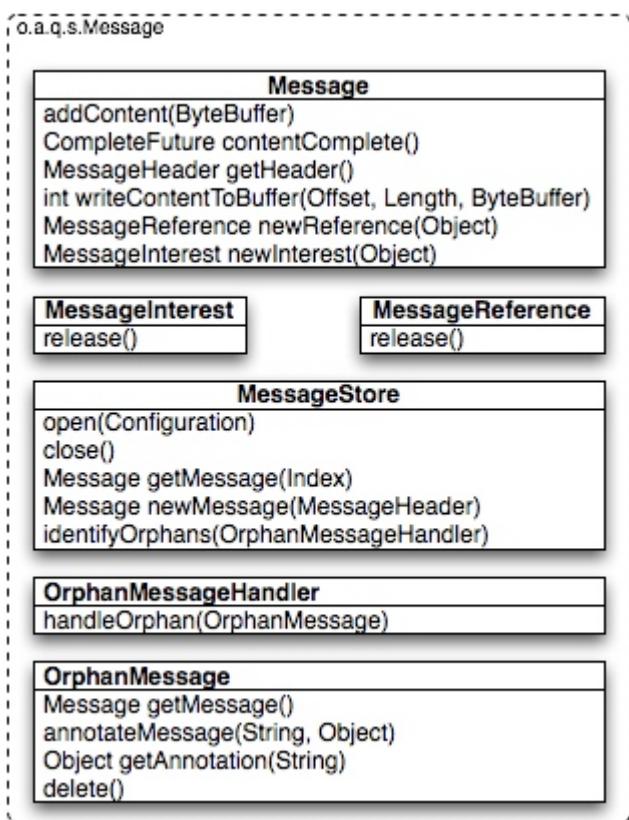
Summary

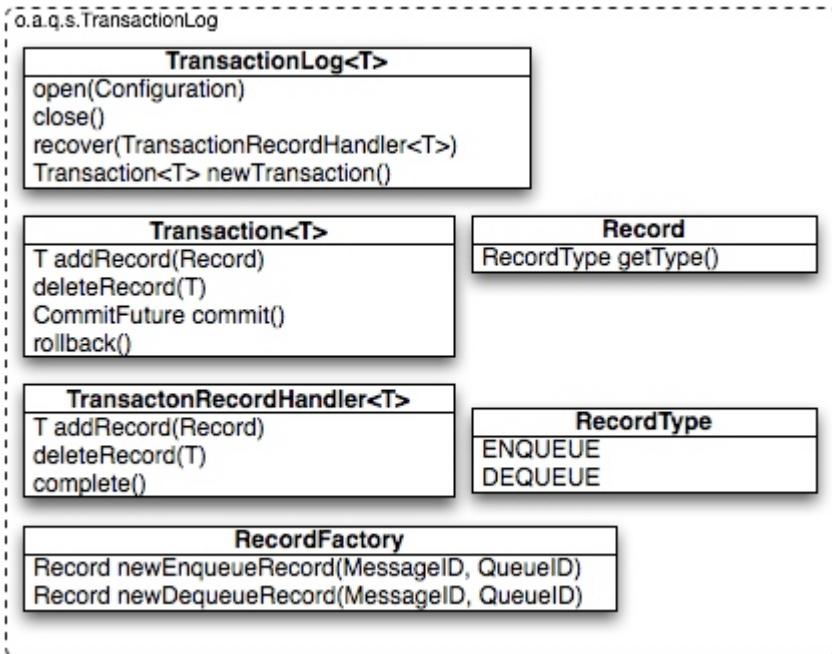
Currently we have the MessageStore interface which is responsible for all persistent storage for the broker. This not only includes Message Data but also, Queueing details, Queue and Exchange Creations and the bindings between them. This makes for difficulties during broker start up as there are also model configuration located in the configuration files which all needs to be processed before message recovery can be performed.

Approach

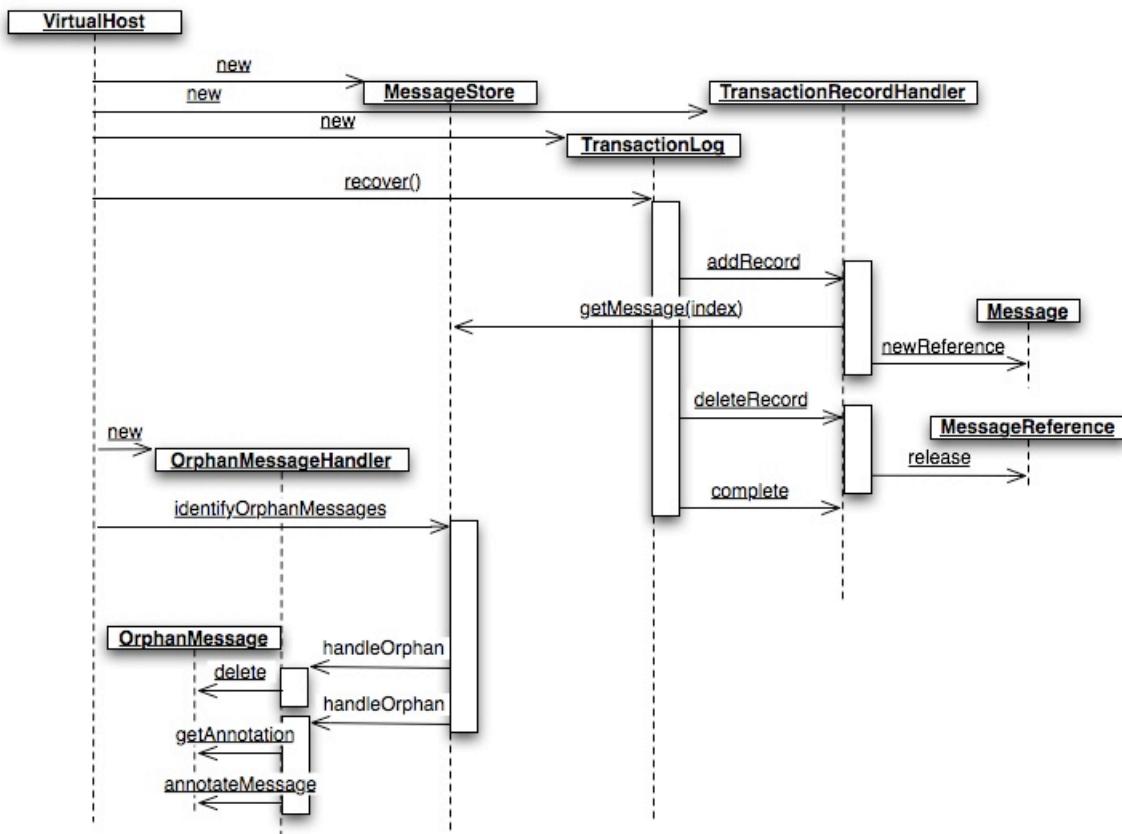
The existing MessageStore interface will be slimmed down to be only responsible for storing and retrieving messages. This will then allow a flow to disk mechanism to put the messages on disk independently. A new TransactionLog will be created to record the enqueue and dequeue operations. To store the Queue, Exchange and Binding model information a new ModelConfiguration will be created to persist this data.

MessageStore and TransactionLog

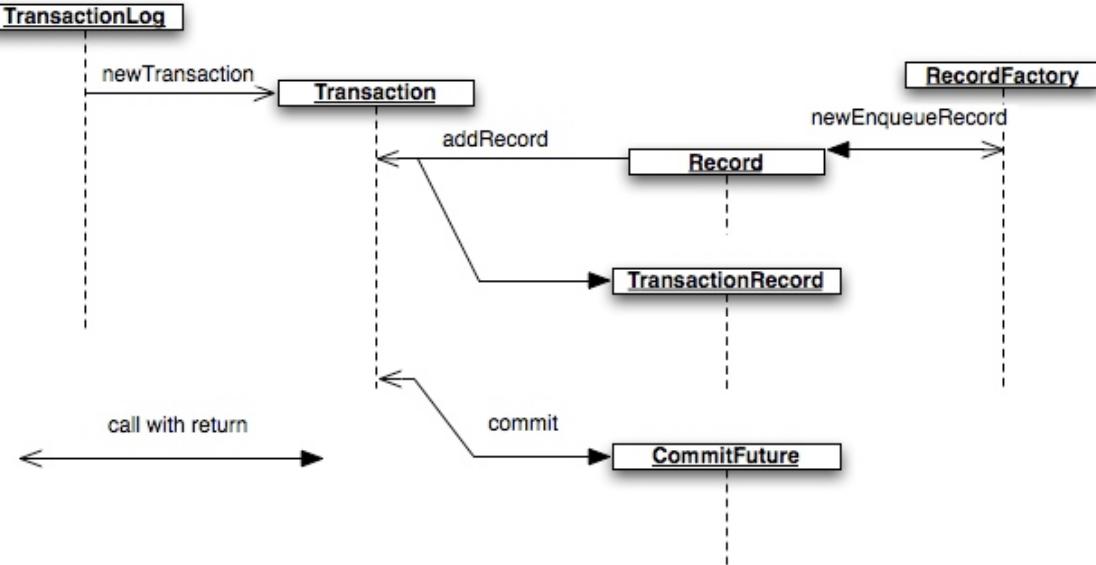




To get a better idea of how these new classes will interact this is the sequence of events that will occur during broker startup. The diagram shows the startup and recovery of the TransactionLog which will retrieve the messages from the MessageStore that are still active. Once the TransactionLog has completed an Orphan detection phase is run on the MessageStore. The exact details of what this process involves can be made a configurable policy decision. However, the initial suggestions are; to leave the messages in the store for tool analysis; the messages can be annotated to record the number of times the message has been detected as an orphan and delete messages that have been marked X number of times; with the introduction of dead letter queues the messages could be moved here.

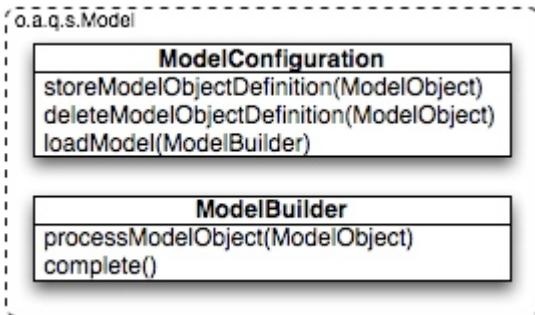


This sequence highlights how the RecordFactory and TransactionLog would interact. A new Transaction would have Records added or deleted that were created via the RecordFactory. The RecordFactory along with the RecordType are the only two classes that need be Qpid specific. The rest of the Transaction package is a generic log of Records.



Model Configuration

Providing an interface to our current configuration will allow us to de-couple our current code from the various sources that are currently used to store this information.



Message Transfer

Summary

The owner of processing a new message takes to a queue is currently split between a number of objects. `IncomingMessage`, `AMQChannel`, and the `TransactionalContext`. In addition we have the `StoreContext` which is used to store any required Transactional data.

Approach

The new `Transaction` object will be created from the context and all operations will be done through this `Transaction` object. The `Transaction` will be passed around where the `StoreContext` currently is and operated on directly.

Java Broker Design - Message Representation

Message Representation

Following on from the [high level design](#) this page will provide a more detailed design of the changes to the broker.

- Overview
- Design
 - Message Representation Interface
 - Unified Reference Counting.
 - Message Interest
 - MessageHeader
- Technical Design

Overview

Currently the broker uses two implementations of `AMQMessageHandle`, one transient, one persistent, which actively stores and retrieves the underlying data. The `AMQMessage` class should provide immutable access to the message data (header and body) stored within the broker, there should be a 1:1 relationship between `AMQMessage` objects and messages delivered through the broker.

Currently the `AMQMessage` is responsible for holding a count of references but the responsibility for maintaining this value is spread throughout the code base. This responsibility needs to be given to a single class so that we can more easily reason about and test its functionality.

This work is part of the [MessageStore](#) refactoring.

Design

As part of the initial high level design the following interfaces were designed.

Message Representation Interface

```
Message
```

```
public interface Message<Header extends MessageHeader>
{
    /**
     * Add additional content data to this message.
     *
     * @param buffer The data that is to be added.
     */
    void addContent(ByteBuffer buffer);

    /**
     * Signal that all content has been received.
     *
     * The returned CompleteFuture can then be used to ensure that all
     * persistent data has been safely persisted.
     *
     * @return CompleteFuture to ensure data is persisted.
     */
    CompleteFuture contentComplete();

    /**
     * Get the Message ID for this Message
     *
     * @return the message id
     */
    long getMessageID();

    /**
     * Retrieve the Header for this Message
     * The return is parameterised so the caller does not need to immediately
     * cast from the {@link MessageHeader} that the MessageStore knows about.
     *
     * @return the Header for this Message
     */
    Header getHeader();

    /**
     * Write the Message to the specified ByteBuffer.
     *
     * If the Message does not fit into the specified ByteBuffer then a
     * subsequent call must be made using the offset value.
     *
     * The offset value is the offset in to the underlying data of the Message.
     * It is needed on subsequent calls if the first ByteBuffer provided was
     * too small.
     *
     * The method will return zero when no data has been written to the Buffer.
     *
     * @param offset position in the underlying data start from which to start
     *               to the ByteBuffer.
     * @param length of data to write into the ByteBuffer
     * @param buffer to write into
     *
     * @return the amount of data written to the buffer
     */
    long writeContentToBuffer(long offset, long length, ByteBuffer buffer);

    /**
     * Create a new Reference to this Message
     *
     * @param referer the object that is referring to this Message.
     *
     * @return a MessageReference.
     */
}
```

```
 */
MessageReference newReference(Object referer);

/**
 * Signal to the MessageStore that the requestor is interested in this
 * Message.
 *
 * This means that the requestor is likely to use the Message shortly so
 * the MessageStore would be advised to keep this Message in memory.
 *
 * @param requestor that is interested in this Message
 *
 * @return a MessageInterest
 */
```

```

        MessageInterest newInterest(Object requestor);
    }
}

```

The new Message interface unifies the current IncomingMessage, AMQMessage and the Store operations becoming a container for the ByteBuffers of data that make up the content of the Message. The Message becomes a proxy for the MessageStore allowing incoming ByteBuffers of content to be added and the ability to use a future to check that the data has been safely stored by the MessageStore.

CompleteFuture

```

public interface CompleteFuture
{
    /** Block the calling thread the task that created this future has completed. */
    void complete();

    /**
     * Non-Blocking call to test that the task that created this future
     * has completed.
     *
     * @return true if the task is complete
     */
    boolean isComplete();
}

```

To enable support for multiple protocols the type of MessageHeader will depend on the protocol of the publishing connection. In addition the Message will be responsible for writing its self to a given ByteBuffer. This allows the Framing layer to be responsible for setting up the ByteBuffer and then requesting that the Message fill in its data.

Unified Reference Counting.

Reference counting is performed in a number of locations in the code so the new Message interface provides a new approach to handling these references. The previous approach used an AtomicInteger count to know when a Message was no longer required this made debugging difficult and so the newReference call takes the object that is referring to this Message. This will allow for the potential to provide a list of what is referencing this Message.

The returned MessageReference object will have a release() to relinquish the reference. However, by also using a the finalizer to automatically release the reference at GC time instances we do not explicitly need to call release().

This pattern is also going to be used for the new concept of Interest.

MessageReference/Interest

```

public abstract class MessageReference/Interest
{
    /** Release the reference in the underlying Message. */
    abstract void release();

    /**
     * Finalise - AutoRelease
     * @throws Throwable
     */
    protected final void finalize() throws Throwable
    {
        release();
        super.finalize();
    }
}

```

Message Interest

As mentioned above in Reference Counting, Message Interest is a new concept for the Java Broker. This allows the running broker to provide a hint to the MessageStore that it is interested in the Message and is likely to use it shortly. This will enable the MessageStore to better control its memory usage. Messages that have interested parties could be safely removed from memory such as by a Flow to Disk routine. However, if Interest is requested then it would be a beneficial to load this message back in to memory or performance will suffer. Message Interests will follow the same pattern as MessageReference, interest will be requested by an Object and the resulting MessageInterest will allow that interest to be explicitly released.

MessageHeader

The current header (ContentHeaderBody) is used to access the header properties and protocol frame values, as a result it is focused on a single protocol version. The introduction of a number of MessageHeader interfaces will allow the various layers to have their own MessageHeader interface and in so doing limit the required access to the header. The main interface will contain all the existing common

properties accessors (CommonContentHeaderProperties). However, the interface should only provide accessors to the data as one of the goals is of this change is to create immutable messages. There is an argument that the broker may wish to add additional data to the message such as routing details. This could be performed in a similar way to SMTP by the addition of properties to the Message. An alternative approach would be to encapsulate original message in a new message. As the broker does not currently need to add any additional properties to messages that it delivers we can delay the decision/inclusion of this feature until the AMQP-WG decide what approach they want implementers to use.

```
MessageHeader
```

```
public interface MessageHeader
{
    /* The size in bytes of the header. */
    long getHeaderSize();
    /* The size in bytes of the message body content. */
    long getBodySize();

    /*
     * General getter to allow arbitrary properties to be
     * used by JMS Selectors
     */
    Object getProperty(String key);

    /*
     * Getters for Common Header properties primarily used
     * by JMS Selectors
     */
    AMQShortString getAppId();
    AMQShortString getContentType();
    AMQShortString getCorrelationId();
    byte getDeliveryMode();
    AMQShortString getEncoding();
    long getExpiration();
    AMQShortString getMessageId();
    byte getPriority();
    AMQShortString getReplyTo();
    AMQShortString getTimestamp();
    AMQShortString getMessageType();
    AMQShortString getUserId();
}
```

The changes to the storage classes allow the introduction of an interface to a MessageHeader that is specific to the store classes. This interface will allow 3rd party storage plugins written in the storage layer to be independant of any future protocol changes.

store.MessageHeader

```
public interface MessageHeader
{
    /**
     * is this Message a Persistent Message?
     *
     * @return true if it is persistent
     */
    boolean isPersistent();

    /**
     * Write the MessageHeader to the specified ByteBuffer in the correct
     * format for the specified Session.
     *
     * If the MessageHeader does not fit into the specified ByteBuffer then a
     * subsequent call must be made using the offset value.
     *
     * The offset value is the offset in to the underlying data of the
     * MessageHeader. It is needed on subsequent calls if the first ByteBuffer
     * provided was too small.
     *
     * The method will return zero when no data has been written to the Buffer.
     *
     * @param session the target session for which this Header should be
     *                 formatted.
     * @param offset position in the underlying data start from which to start
     *               to the ByteBuffer.
     * @param length of data to write into the ByteBuffer
     * @param buffer to write into
     *
     * @return the amount of data written to the buffer
     */
    long writeContentToBuffer(Session session, long offset, long length, ByteBuffer buffer);
}
```

In a similar vein, a new Framing layer MessageHeader will allow access to only the properties/methods that are needed by that layer. This approach allows the Framing layer to be independent of the message header/data it is transporting and focus on the correct formatting of frames for the given consumer.

framing.MessageHeader

```
public interface MessageHeader
{
    long getHeaderSize();
    long getBodySize();

    long writeContentToBuffer(Session session, long offset, long length, ByteBuffer buffer);
}
```

As we will eventually have clients connected on a variety of protocol versions isolating the framing layer from any transformations that must be performed through the MessageHeader interface should reduce any potential complexity at this layer and require that the MessageHeader is capable of writing itself to a ByteBuffer for the given session.

MessageHeaderFactory

```
public interface MessageHeaderFactory<Header extends MessageHeader>
{
    /**
     * Convert the given ByteBuffer in to an MessageHeader
     * @param buffer containing the data to convert in to an MessageHeader
     * @return the MessageHeader that was in the buffer
     */
    Header createMessageHeaderFromBuffer(ByteBuffer buffer);
}
```

Technical Design

TBC

Network IO Interface

- Objective and Scope.
 - Overview
 - Problem Statement
 - 1 High message throughput on one publishing connection
 - 2 Medium-High throughput on several publishing connections
 - 3. Environmental issue prevents broker from processing buffer data
 - 4. Slow client causes broker network buffers to grow
- Exclusions: / Assumptions
- Functional Requirements
- Non Functional Requirements
- Architecture Design
 - Overview of Design
 - Breakdown of work
- Testing
- Impact
- Compatibility / Migration Implications
- Risks
- Network IO Interface discussion points
- New common network and protocol interfaces
- Port server to new interface

Objective and Scope.

Overview

The broker is prone to heap exhaustion, leading to OutOfMemory errors. One major source of memory consumption are the buffers used to hold unprocessed AMQP frames once they have been read from the socket. These are currently able to grow uninhibited, and there is no means available to control them. Further, the transport layer is poorly implemented and difficult to work in. Improving encapsulation is an explicit goal of this work.

For more information on the current design, please see [Current Architecture](#).

Problem Statement

When the broker is unable to process frames as quickly as they are being sent these buffers begin to fill up and the broker has no way to limit those. For the broker to effectively manage its memory usage, it needs to be able to at least place an upper bound on the size of its network buffers. It also has no way to know how large those buffers are.

1 High message throughput on one publishing connection

This is where the publishing client is sending a consistently sustained high rate of messages to the broker, and is more likely to happen where some of the messages are persistent

Data from the client gets out of the client side buffers and into the broker side buffers. The broker is processing messages onto the queues as fast as it can, but gets backed up and the broker side buffers grow until eventually the broker OOM's, the heap filled by the MINA buffers along with the data in queues.

2 Medium-High throughput on several publishing connections

This is where there are multiple connections sending data to the broker, at varying rates from medium to high (as measured in Qpid terms as greater than the ave/max throughputs measured in test). The broker threads are being managed by the JVM in terms of processor time i.e. yielding to each other in an unpredictable way.

As for case 1 above, with the caveat that it's far harder to predict how long it'd take to happen and how the TCP socket level behaviour will impact the client. It's also a more likely real world scenario i.e. probably more than one connection for MDS publication (for example) would result in a set of growing buffers resulting in OOM.

3. Environmental issue prevents broker from processing buffer data

This is where the broker, for example, runs out of SAN or something else (disk for logging etc) and so cannot process the messages out of the buffers and onto the queues. CPU ?

As for case 1, assuming that the broker just needs to be up to have data being buffered which previous tests certainly indicate i.e. a publisher can happily pump data into a disabled broker for some time before the connection gets killed.

4. Slow client causes broker network buffers to grow

This is where the client is not reading data as fast as the broker is sending it. The data will be buffered on the broker, increasing memory usage. This is particularly problematic when sending messages to the client since the payload will then be in both the message store and the network buffers at the same time, doubling usage per message.

Exclusions: / Assumptions

1. No AMQP semantics are involved. The aim of this work is purely to limit the size of the network buffers between the client producing AMQP frames and the broker processing them. It does not involve any protocol specific work. In OSI terms, this work is aimed at layer 4, not layer 7.
2. Higher level information should be determined by the broker itself. No policy will be applied beyond blocking reads if the buffer is full.
3. Buffers are sized uniformly across all connections
4. Buffers are fixed at startup and do not change
5. Standard TCP flow control is the only mechanism used to signal to the client that it should cease to send data.
6. It is better for the client to block further writes to the socket than allowing memory consumption to grow unimpeded
7. The broker should not block

Functional Requirements

1. Buffer size control - all buffers have an upper size limit other than the queue itself
2. TCP options: SO_KEEPALIVE, OOBINLINE, SO_RCVBUF, SO_REUSEADDR, SO_SNDBUF, SO_LINGER, SO_TIMEOUT, TCP_NODELAY
3. SSL: link level encryption, do we want to consider things like certificate validation etc here or at a higher level? Consult with RHS
4. signal on idle requires timer support
5. Need to be notified when socket has been closed
6. The broker needs to know that the transport layer is full and the write would / did not succeed - "don't send anymore just now until I clear this Future"?
7. Non-TCP transports such as InVM, infiniband.
8. Network buffers can be of unlimited size
9. Rate statistics need to be available, including total throughput and average time for send() to complete. See [Java Broker Design - Operational Logging](#) for details
10. send() should (optionally) fail after a configurable timeout rather than block forever
11. Need to be able to change buffer sizes on new connections at runtime, existing connections can remain unchanged
12. Send and receive buffers should be independently sized

Non Functional Requirements

1. Startup loading of transport plugins
2. User can select specific transport to use
3. Peer A running transport A can talk to Peer B running transport B
4. Connections do not require a thread each (broker only, client can probably live with that)
5. the semantics of org.apache.qpid.BasicMessageProducer.send() need to change. It may now block if there isn't enough free space to write the entire message out. The change to this methods semantics needs to be considered in the light of the stated JMS semantics and the change to support acknowledgement of publishes in AMQP 0-10 and higher.
6. Need to document relative impact of buffer sizes

Architecture Design

Common should have an interface which all transport plugins can implement and which the server and client can use. The interface would include a means to set the standard socket options and to limit its total memory usage.

TCP itself has a flow control mechanism which kicks in when the receiver of data cannot read from the socket as fast as data is being sent. TCP sockets use send and receive buffers to attempt to smooth the flow from application to network and maximize network performance. By limiting the rate at which the application reads from the network to the rate at which it can process the data the sender of the data is throttled to that production rate.

Overview of Design

1. Common will hold a transport layer interface which the existing MINA transport will be ported too. We will also port the 0-10 client o.a.q.transport.network.io package to that interface. This interface should be quite simple.
2. Methods to send, receive, flush, close, open, listen and a method to set TCP options are likely to be sufficient. These would operate on a QpidByteBuffer, essentially MinaByteBuffer to avoid having to fix our use of expanding buffers at the same time.
3. The server and client both use common for their network layer, and will need to be updated to use the new interface. They will need to pass through the configured socket options.
4. When processing the incoming data, one frame at a time will be processed and that frames processing will be completed before the next one is read. There will be no other data structures used to hold unprocessed frames. This will mean that the sender will become aware of variations in the receivers processing speed much sooner than is currently the case. Slow downs or pauses in processing incoming frames will cause the buffer to fill up and flow control to kick in. This can be mitigated if desired by increasing the relevant buffer sizes.
5. The server will need to be substantively modified to push the MINA specific parts into the appropriate plugin. This primarily involves replacing the MINA ByteBuffer with a QpidByteBuffer and refactoring the MINA specific parts of the protocol handlers.
6. Implementing fixed size IO buffers would require replacing MINA with an alternative implementation of the transport layer. The first step in any such process would be to clearly encapsulate the concept of a transport layer using the existing MINA code as the initial implementation. The next step would be to adapt the already used 0-10 transport to sit clearly behind the same interface. Finally a network transport more attuned to the needs of a broker (supporting large numbers of incoming connections) could be developed from the base of the existing 0-10 transport.
7. The changes would impact the client and broker where they interface with the transport layer. In the first phase the client and broker would be altered to use this implementation independent interface. Once this work had been completed the client and broker could be tested and released using the MINA implementation proving that no adverse impact from the encapsulation of the transport layer had occurred.
8. Once the first phase of the work has been completed, alternative transport implementations could be developed. this would require no code changes to the client or broker, but would require system testing to prove that behaviour was correct when using the

alternative transport implementation.

9. Bounding the buffers attempts to address the issue of regulating incoming data flow to the rate at which it can be processed by the receiver. This will generally occur when the sender is capable of sending bursts of data at a high rate. This is most evident with persistent messages where the rate at which messages can be persisted to disk is much lower than the rate at which they can be sent over the network. Fundamentally if it is not able to process messages at the rate at which they are being sent, Qpid should not accept them, pretending to do so is giving the application a false impression about what Qpid is doing, and is potentially only deferring an issue to a later point when there will be a great deal of message loss. Further more the behaviour is non-JMS compliant (JMS expect publishing to be a synchronous activity).

Breakdown of work

1. Encapsulate existing networking layer better
 - a. [New common network and protocol interfaces](#)
 - b. [Port server to new interface](#)
2. [Port client to new interface]
3. Remove Job/Event
4. Bind network buffers
5. Tests
 - a. Representative workload tests need to be developed and put into perftests.

Testing

Testing under load and handling error conditions (unexpected disconnection etc) will need to be carried out.

New load tests which simulate application workloads need to be developed so that we can provide accurate configuration guidance. These tests then need to be carried out on Windows, Linux and Solaris in all permutations of client/server.

New unit tests will need to be written to cover the transport plugins and the new interfaces. Existing test coverage in this area is minimal.

Impact

There is a potential effect upon performance, we will need to measure this once it has been implemented to quantify what effect, if any, it has had.

Compatibility / Migration Implications

1. Older clients connected to a new broker may suffer OOM when tcp flow control kicks in. This seems preferable to the broker suffering OOM.
2. Clients which upgrade their library may experience a change in behaviour of the send() method, since it may now block if the clients network buffer is full. This needs to be appropriately communicated. It should not be significantly different in behaviour from using transactions in the producer session however.

Risks

1. MINA is quite deeply embedded in the server and will require some work to excise it fully. This is somewhat mitigated by the decision to import mina.ByteBuffer and continue using that.
2. Differences in behaviour of transport layer may expose other bugs in the broker which were being hidden before.
3. Inadequate test coverage, in particular the lack of representative application workloads in the performance test suite.

Network IO Interface discussion points

Discussion Points : 2009-07-03

This page captures points to be addressed from a discussion between:

Robery Godfrey (RG)
Marnie McCormack (MM)
Martin Ritchie (MR)
Aidan Skinner (AS)

Use Cases

ID	Raised By	Description	Status	Outcome
UC-1	AS	Need more details on what the client changes that need to be done, in a new Doc.		
UC-2	MM	Use Case 4 Outbound buffers can fill if client is slow.	Added	

Functional Requirements

ID	Raised By	Description	Status	Outcome
F-1	AS	We are trying to bind all buffers/queues other than the AMQP Queue itself.	Clarified	
F-2	MR	More clarity : F-6: Transport layer will not block but report full	Clarified	

F-3	MM	Move Marnie's Points from Compatibility / Migration in to the functional requirements section	Moved	
F-4	MM	from C/M:3 Ability to switch IO implementation (mina/new io)/ Run with bounded/unbounded buffers at start up. Not dynamically.	Added	
F-5	RG	Keep rate statistics rather than logging on the buffers. Capturing data is cheaper than logging directly.	Added reference to operational logging	
F-6	RG	Average time for sends to complete	Added to stat gathering req	
F-7	ALL	C/M:4 Logging: Goal is to identify problem area Client/Network/Broker, capturing the size of the buffers will help us identify if it is the client or broker that is the cause.	Added	
F-8	MM	C/M:5 Bound Changes: Bounding buffers will have impact. A) what size do you set it to. B) What paradigms will need their buffers changed.	Added sizing documentation to NFRs	
F-9	MM	C/M:5 BC: Buffer should be configurable at in a dynamic context, new connections will have the new buffer size. Existing connections will remain unchanged.	Added to FRs	
F-10	RG	C/M:5 It is not necessary for the input and output buffers to be the same size. That is the buffer used to receive messages from a publish(input) and the buffer used to hold messages being sent to the a client(output).	Added to FRs	

Non-Functional Requirements

ID	Raised By	Description	Status	Outcome
NF-1	AS	4. Current 0-10 client IO has one thread per connection. So not suitable for direct use in broker just now.	Clarified relevant NFR	
NF-2	RG	5. We need to document current semantics before we can say it will change.	Added MessageProducer.send() behaviour	
NF-3	MR	(From comments) 5. send() should have option for not blocking.	Added to FR	

Comments

ID	Raised By	Description	Status	Outcome
C-1	RG	All IO buffers would be affected		
C-2	RG	Mina more likely with persistent message		
C-3	MM	Break this down in to components.	Added work breakdown and suggested order	
C-4	MM	Expand all TCP options (TCPNoDelay...) detail what they are being exposed for, setting/reading	Added complete list	

Discussion Points : 2009-07-07

This page captures points to be addressed from a discussion between:

Robery Godfrey (RG)
Marnie McCormack (MM)
Martin Ritchie (MR)
Aidan Skinner (AS)

General

ID	Raised By	Description	Status	Outcome
G-1	MM	Diagrams would help		

Functional Requirements

ID	Raised By	Description	Status	Outcome
F-11	MM	F-8,9,10 : Need to be reworked to be in 3rd person	Done	

Overview of Design

ID	Raised By	Description	Status	Outcome
O-1	ALL	O-4 : Capture the impact of removing the Job Queue/Limiting. The implications of the changes perhaps accompanied with statistics showing improved performance. Client is now exposed to latency, i.e. If broker is impacted then client will see this.	Done	
O-2	MR	o-5 : Expand details of substantive changes	Done	

Breakdown of work

ID	Raised By	Description	Status	Outcome
B-1	RG	B-1 : Highlight that the inclusion of QpidByteBuffer will be done as part of common interface creation	Done	
B-2	MM/RG	Put BofCP into the BoWork section, highlighting which parts relate to which task.	Done	

Testing

ID	Raised By	Description	Status	Outcome
T-1	MM	Define representative testing platforms, LAN, MAN, WAN, etc		

Discussion Points : 2009-07-28

ID	Raised By	Description	Status	Outcome
1	LB	Change breakdown of work into phases. Phase 1 encapsulation of broker, phase 2 encapsulation of client etc		

New common network and protocol interfaces

Purpose

This design page describes the low level design for the new interface which is aimed at facilitating encapsulation for the Network code in both the Java Broker & Client.

This is the first step in decoupling the existing IO layer from both the surrounding Qpid code and more specifically from the current tie-in to MINA.

This document will provide sufficient information for architecture review and also for input to task breakdown & planning.

Interface Requirements

1. Provide an API which supports pluggable network layers
2. Facilitate the replacement of instantiations of MINA classes with an abstraction
3. Network interface and drivers should be thread model agnostic. The
4. Ability to set TCP options (see main design doc for details)
5. Provide support for configuration of related properties including buffer size
6. The interface will support an SSLEngine

Current design

For details on the current implementation see [Current Architecture](#)

New Design

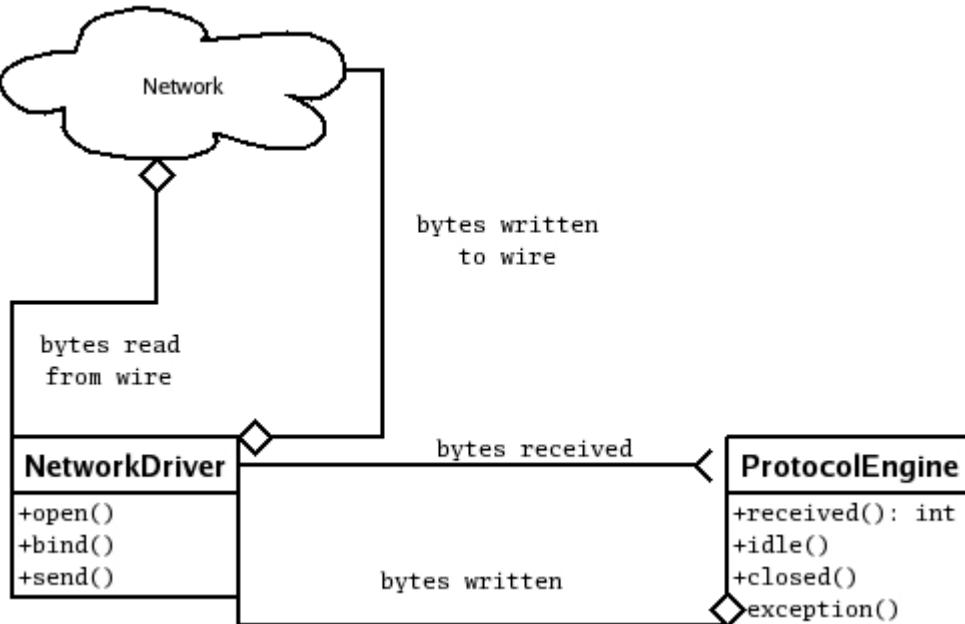
NetworkDriver takes bytes from the network and passes them to the **ProtocolEngine**. It also accepts bytes from the **ProtocolEngine** and writes them to the network.

ProtocolEngine accepts bytes from the **NetworkDriver** and turns them into AMQFrames for processing. It accepts frames and encodes them into bytes which it then hands off to the **NetworkDriver**.

Design choices

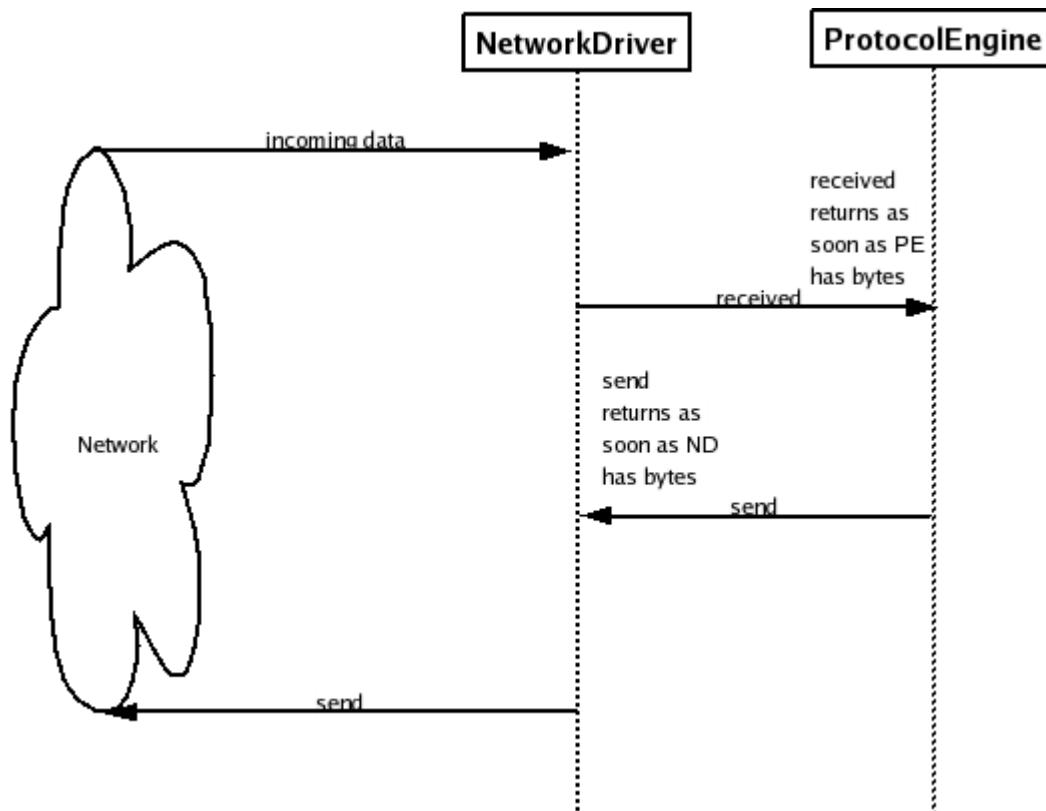
1. Initial designs will only support TCP (see main design doc for info)
2. The `NetworkDriver.send()` method will not block, and neither will the `ProtocolEngine.received()`. As soon as they have stored the data for later processing they will return.

New network / protocol engine interface in org.apache.qpid.common



In the new version, a NetworkDriver is created by a ProtocolEngine (in the case of outgoing connections) or is bound to a socket and creates a ProtocolEngine when new connections are created. The network driver passes raw data to the ProtocolEngine which is responsible for both decoding the frames and processing them. When the ProtocolEngine wishes to send data, it does so by calling the NetworkDriver. The existing mechanisms for frame listeners etc are retained, but are decoupled from the network processing parts.

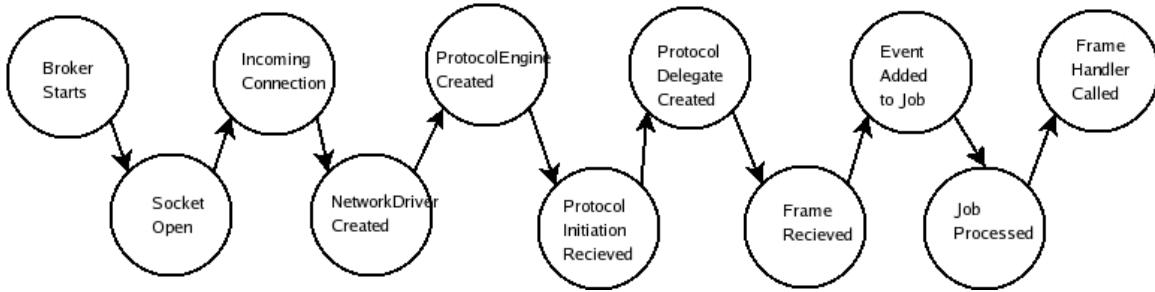
At the start of a connection the the NetworkDriver will pass data to a ProtocolEngine which will handle protocol negotiation. The implementation will use the existing Sender and Reciever interfaces in org.apache.qpid.transport which will allow the use of the existing alternate transport layer implementations.



The thread of control remains with the network driver up until received(), when the ProtocolEngine becomes responsible. The ProtocolEngine should return from received after it has accepted the data for processing without blocking the network thread. Conversely, the NetworkDriver should return control to the thread calling send() as soon as possible after accepting the data for writing.

Data comes in from the operating system, is read from the socket by the NetworkDriver and given to the ProtocolEngines received method. The ProtocolEngine is responsible for processing the bytes and interfacing to the rest of the broker or client.

The ProtocolEngine will write bytes to the wire using the NetworkDriver which implements the existing Sender interface from org.apache.qpid.transport



Sender (already exists):

```

/**
 * This interface is implemented by things which accept data for sending to a remote end point
 */
public interface Sender<T>
{
    // Sets the TCP idle time out
    void setIdleTimeout(long l);

    // Accepts the data for sending
    void send(T msg);

    // Flushes all data pending
    void flush();

    // Closes the connection
    void close();
}
  
```

The ProtocolEngine will implement the Reciever interface to be given bytes by the NetworkDriver in it's received method.

Receiever (already exists):

```

/**
 * This interface is implemented by things which accept data for processing
 */
public interface Receiver<T>
{
    // Called when data has been received from the network
    void received(T msg);

    // Called when an exception has occurred
    void exception(Throwable t);

    // Called when the underlying socket has been closed for reading
    void closed();
}
  
```

The ProtocolEngine will implement the following interface:

```

/**
 * A ProtocolEngine is a Receiver for java.nio.ByteBuffers. It takes the data passed to it in the
received
 * decodes it and then process the result.
 */
public interface ProtocolEngine extends Receiver<java.nio.ByteBuffer>
{
    // Sets the network driver providing data for this ProtocolEngine
void setNetworkDriver (NetworkDriver driver)

    // Returns the remote address of the NetworkDriver
void SocketAddress getRemoteAddress()

    // Returns number of bytes written
long getWrittenBytes()

    // Returns number of bytes read
long getReadBytes()

    // Called by the NetworkDriver when the socket has been closed for reading
void closed()

    // Called when the NetworkEngine has not written data for the specified period of time (will
trigger a
// heartbeat)
void writerIdle()

    // Called when the NetworkEngine has not read data for the specified period of time (will close
the connection)
void readerIdle()

    /**
     * Accepts an AMQFrame for writing to the network. The ProtocolEngine encodes the frame into
bytes and
     * passes the data onto the NetworkDriver for sending
     */
    void writeFrame(AMQDataBlock frame)
}

```

```

public interface ProtocolEngineFactory
{

    // Returns a new instance of a ProtocolEngine
ProtocolEngine newProtocolEngine()

}

```

The NetworkDriver will implement the following interface:

```

public interface NetworkDriver extends Sender<java.nio.ByteBuffer>
{
    // Creates a NetworkDriver which attempts to connect to destination on port and attaches the
    ProtocolEngine to
    // it using the SSLEngine if provided
    static NetworkDriver open(int port, InetAddress destination, ProtocolEngine engine,
    NetworkDriverConfiguration config, SSLEngine engine) throws OpenException;

    // listens for incoming connections on the specified ports and address and creates a new
    NetworkDriver which
    // processes incoming connections with ProtocolEngines created from factory using the SSLEngine if
    provided
    static void bind (int port, InetAddress[] addresses, ProtocolEngineFactory factory,
                      NetworkDriverConfiguration config, SSLEngine engine) throws
    BindException;

    // Returns the remote address of underlying socket
    void SocketAddress getRemoteAddress()

    /**
     * The length of time after which the ProtocolEngines readIdle() method should be called if no
     data has been
     * read
     */
    void setMaxReadIdle(int idleTime)

    /**
     * The length of time after which the ProtocolEngines writeIdle() method should be called if no
     data has been
     * written
     */
    void setMaxWriteIdle(int idleTime)

}

```

The NetworkConfiguration interface provides configuration data for the NetworkDriver:

```

/**
 * This interface provides a means for NetworkDrivers to configure TCP options such as incoming
and outgoing
 * buffer sizes and set particular options on the socket. NetworkDrivers should honour the values
returned
 * from here if the underlying implementation supports them.
 */
public interface NetworkDriverConfiguration
{
    // Taken from Socket
    boolean getKeepAlive()
    boolean getOOBInline()
    boolean getReuseAddress()
    Integer getSoLinger() // null means off
    int getSoTimeout()
    boolean getTcpNoDelay()
    int getTrafficClass()

    // The amount of memory in bytes to allocate to the incoming buffer
    int getReceiveBufferSize();

    // The amount of memory in bytes to allocate to the outgoing buffer
    int getSendBufferSize(int size);
}

```

Relationship to existing design

The [Current Architecture](#) can be thought of with AMQMinProtocolSession taking the place of the ProtocolEngine and AMQPFastProtocolHandler being the NetworkDriver. However the separation of responsibility is not clear between the two and they are both tied directly to MINA.

Current and proposed network interfaces notes

ID	Raised By	Description	Status	Outcome
1	RG	Requirements 5,6,8,9 are design choices	Done	Moved to new section
2	RG	Requirement 10 is observation / scope limit	Done	Moved to new section
3	RG	Requirement 7 should be removed	Done	Removed
4	RG	Requirement 9 expand buffering of bytes before wire, writers are not synch it's about the contract. Similar to requirement 5	Done	Folded into 5
5	MM	Requirement 3 should be reworded since it's not exposing methods	Done	Reworded
6	RG	Explain configuration interface better later in documentation	Done	added text
7	MM	Document needs better flow requirements, current implemenation, new implemntation, dont' see how it all hangs together	Done	Current implementation removed, replaced with reference to current architecture page
8	MM	Doesn't highlight current problems	Done	Current architecture page talks about this in detail
9	LB	Link to reason why we're doing this work	Done	see point 7, 8
10	RG	Previous / current design - move out and list w/explanation of what is wrong, link here and can then provide what is being done to address this.	Done	Added link to current design page
11	RG	Add link to old design parts	Done	added linke
12	MM	Not clear how docs link together. Hard to review, tick boxes w/out expected content defined	Comment	
13	RG	new implemntation needs to specify that protocol engine turns frames into bytes	Done	added text
14	RG	Move diagram / overview of approach top top before detailed discussion	Done	Moved
15	RG	Highlight that phase 1 is removing MINA leakage behind new interfaces and that's all	Done	Clarified in Port server to new interface
16	MM	Highlight now and phase 1, show MINA leakage	Done	see 15
17	RG	Phaseing - interfaces, new decoders. Smaller parts better	Done	see Network IO page
18	ALL	new diagram - future state, phase 1	Done	see Port server to new interface page
19	MM	Description does not tie with interfaces	Done	Amended
20	RG	Interfaces could do with class level doc, how it's used, what it does. Network driver extends sender, explain how it uses sender. ProtocolEngine is a reciever. ND is a sender	Done	added doc
21	MM	Hard to see links between sender/reciever and class	Done	made explicit that ND is a Sender, PE a Receiver
22	RG	Put more details about the abstract interfaces and the concrete classes later when we talk about this and refer back to interfaces	Done	see 21
23	MM	ND clearer seperation between what is Sender and what is ND and that ND extends Sender	Done	see 21
24	MM	Need state transition diagram to show data transfer	Done	see 25
25	RG	Sequence diagram showing flow of control	Done	added diagram
26	RG	Detail to do with Job should move to impl doc	Done	removed Job info
27	MM	Exception handling on interfaces, open, bind, configuration	Done	added throws clauses
28	RG	Mention exception needs handled, throws OpenExcep/BindExcept. NOT AMQE. AMQE delenda est	Done	see 27
29	RG	"New design" not new implementation	Done	changed title
30	RG	ProtocolEngine desc should mention that it also converts frames to bytes	Done	added to description
31	RG	NetworkDriverConfiguration needs to text to explain what is going with these details	Done	added text
32	RG	Reduce interface diagram to just who network driver / protocol engine	Done	added diagram

33	MM	Highlight difference with old	Done	added section detailing this
34	RG/MR	map ND -> MINA, Rest -> Decoder	Done	see 33
35	RG	Highlight lack of MINA leakage	Done	see Port server to new interface
36	RG	Implementation details (AMQProtocolEngine_0_N) should be removed	Done	removed

Port server to new interface

- Objectives
- Overview of new implementation
- Work required
 - Add QpidByteBuffer
 - Implement NetworkDriver
 - MINANetworkDriver class
 - Changes required to implement MINANetworkDriver
 - Implementation of ProtocolEngine
 - Changes required to implement AMQProtocolEngine
 - Changes required to implement AMQProtocolEngineFactory
 - Implement NetworkConfiguration
 - Transport layer selection in o.a.q.server.Main

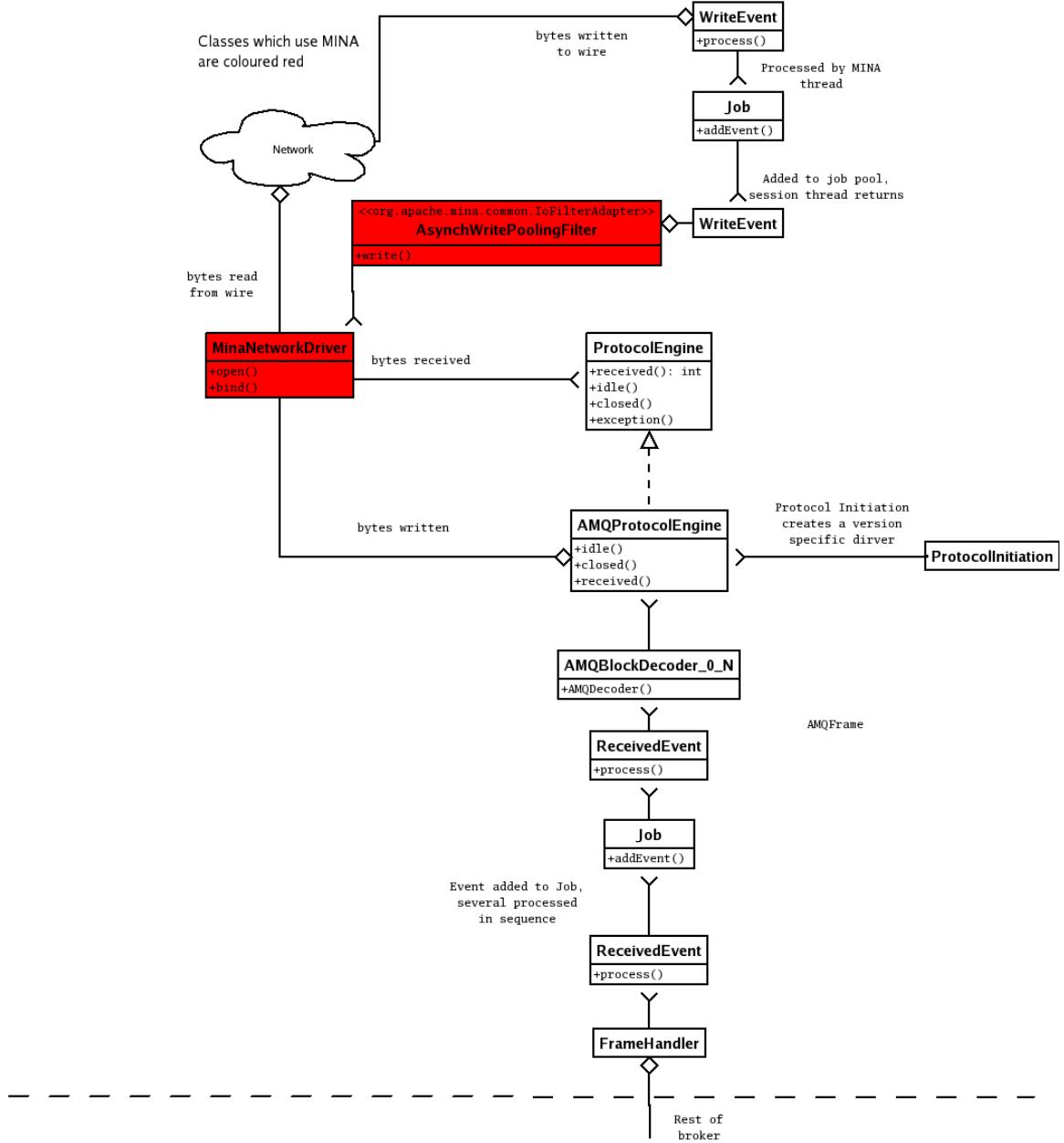
Notes:

- Port server to new interface notes
- Port server to new interface tests

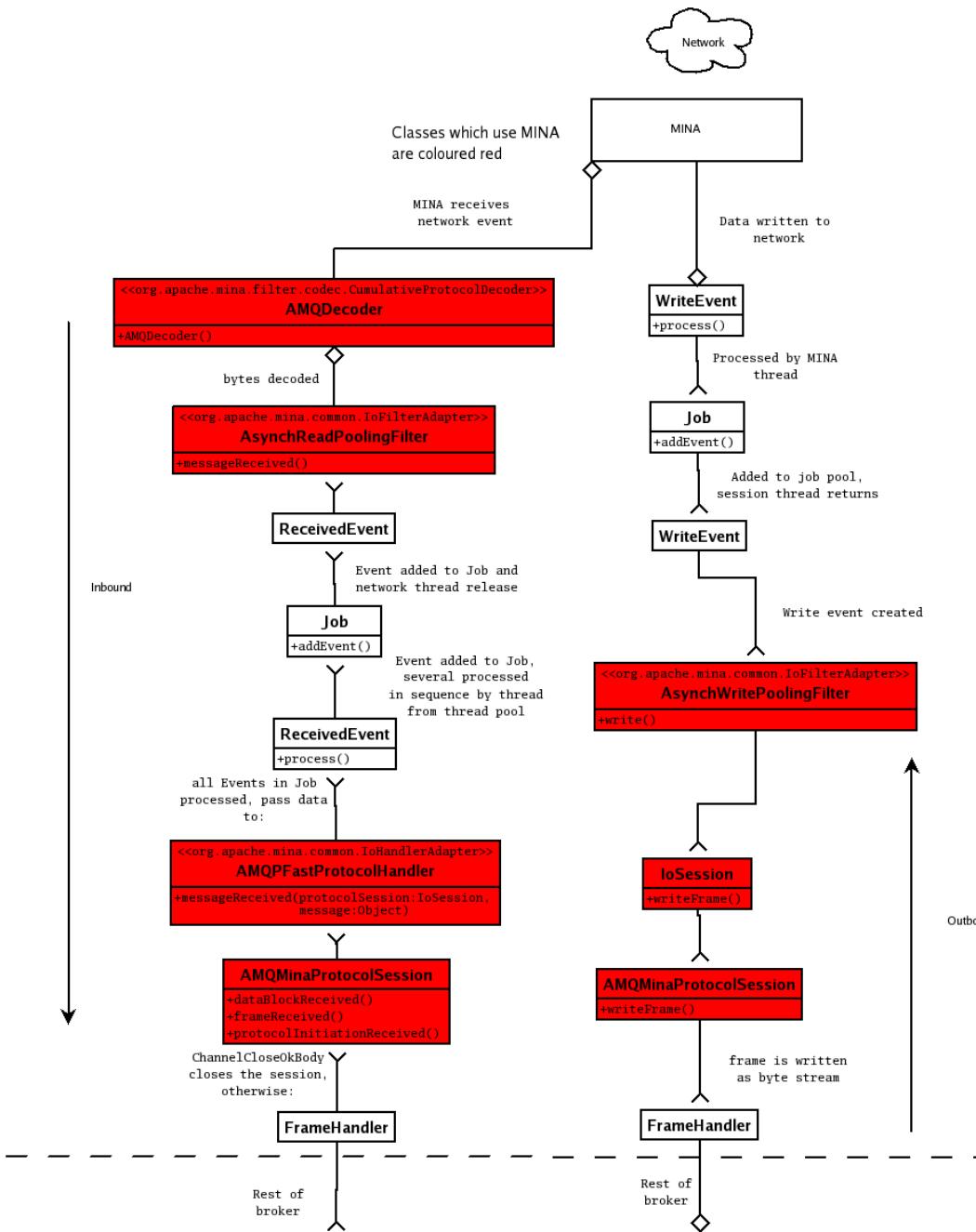
Objectives

1. Implement ProtocolEngine and NetworkDriver from New common network and protocol interfaces
2. Isolate MINA dependencies into MINANetworkDriver
3. No changes to threading model
4. No changes to configuration file

Overview of new implementation



This first phase confines the MINA dependency behind the NetworkDriver interface. As can be seen by comparison from the following diagram describing the 0.5 implementation, the changes to the flow of control are not significantly different. The only significant change is to decouple the protocol processing from the network processing.



Work required

Add QpidByteBuffer

Class	Method	Change
org.apache.mina.common.ByteBuffer	all	<p>At a number of points in the code we rely on behaviour present in MINA ByteBuffers but not java.nio.ByteBuffers, in particular the auto-expanding functionality. To avoid having to rewrite those parts at this stage we should import the MINA ByteBuffer class as QpidByteBuffer and continue to use it as is. Note that this functionality is not related to the network buffers, the QpidByteBuffers are the outputted data structure. The existing implementations will have their MINA or java.nio.ByteBuffers converted to QpidByteBuffers by the ProtocolEngine after decoding</p>

Implement NetworkDriver

MINANetworkDriver class

```

/**
 * This class wraps the existing MINA implementation behind the NetworkDriver interface. This uses
 * the AsyncWritePoolingFilter to buffer network writes so that send() is not synchronous.
 */
public class MINANetworkDriver implements NetworkDriver extends IoHandlerAdapter
{
    /**
     * Creates a concrete NetworkDriver which opens a connection to the specified address on the
     port
     * and starts the given ProtocolEngine instance using MINA
     */
    static NetworkDriver open (int port, InetSocketAddress destination, ProtocolEngine engine,
                           NetworkDriverConfiguration config);

    /**
     * Listens for incoming connections on the specified port and IPAddresses. Creates a new
     concrete
     * NetworkDriver for each incoming connection and creates a ProtocolEngine with the given
     factory
     */
    static NetworkDriver bind (int port, InetSocketAddress[] addresses, ProtocolEngineFactory factory,
                           NetworkDriverConfiguration config);

    void SocketAddress getRemoteAddress()

    /**
     * The length of time after which the ProtocolEngines readIdle() method should be called if no
     data has been
     * read
     */
    void setMaxReadIdle(int idleTime)

    /**
     * The length of time after which the ProtocolEngines writeIdle() method should be called if no
     data has been
     * written
     */
    void setMaxWriteIdle(int idleTime)

    /**
     * Adds the data to an Event through AsynchWritePoolingFilter for writing to MINA
     */
    void send(java.nio.ByteBuffer data)
}

```

Changes required to implement MINANetworkDriver

It replaces AMQPFastProtocolHandler and the majority of the implementation will be taken from that class. bind() will be taken from the existing org.apache.qpid.server.Main

Class	Method	Change
AMQPFastProtocolHandler	general	Renamed to MINANetworkDriver, implements NetworkDriver interface
AMQPFastProtocolHandler	exceptionCaught	AMQP functionality removed, passes exception onto ProtocolEngine
AMQPFastProtocolHandler	messageReceived	calls ProtocolEngine.received()
AMQPFastProtocolHandler	sessionClosed	calls ProtocolEngine.closed()
AMQPFastProtocolHandler	sessionCreated	creates a new ProtocolEngine from ProtocolEngineFactory, other AMQP functionality moved to AMQProtocolEngine. Configuration taken from NetworkDriverConfiguration
AMQPFastProtocolHandler	sessionIdle	calls ProtocolEngine.readerIdle() or ProtocolEngine.writerIdle() as appropriate
AMQMinaProtocolSession	AMQMinaProtocolSession	Threading pool configuration moved to NetworkDriver
AMQMinaProtocolSession	initHeartbeats	moved to NetworkDriver.setMaxReadIdle() and NetworkDriver.setMaxWriteIdle()
AMQMinaProtocolSession	getRemoteAddress	moved to NetworkDriver.getRemoteAddress

Main	bind	Moved to NetworkDriver
Main	startup	Network configuration moved to NetworkDriver, replaced with construction of NetworkDriver

Implementation of ProtocolEngine

```

/**
 * This class replaces the existing AMQMinProtocolSession and implements the ProtocolEngine
 interface. It is
 * responsible for processing bytes into frames and for processing those frames. It is also
 responsible for
 * turning frames into bytes and passing them onto the NetworkDriver for writing.
 */
public class AMQProtocolEngine implements ProtocolEngine
{
    /**
     * Processes data. It uses AMQDecoder to decode the frame and place it into a Job for later
processing
     */
    void received(java.nio.ByteBuffer data)

    void setNetworkDriver (NetworkDriver driver)

    /**
     * calls NetworkDriver.getRemoteAddress()
     */
    SocketAddress getRemoteAddress()

    long getWrittenBytes()

    long getReadBytes()

    /**
     * Closes the protocol engine, aborts in-progress transactions etc.
     */
    void closed()

    /**
     * Generates a heartbeat frame
     */
    void writerIdle()

    /**
     * Closes the connection
     */
    void readerIdle()

}

```

Changes required to implement AMQProtocolEngine

Class	Method	Change
AMQMinProtocolSession	Whole class	Rename to AMQProtocolEngine. decouple the AMQP semantics from the underlying networking, for which it should now use MINANetworkDriver. This class will implement the ProtocolEngine interface and become the central point for processing AMQP frames. It will use a Job to hold ReceivedEvents for processing outside of the network thread.
AMQMinProtocolSession	received	implement and use AMQDecoder to decode the byte stream before placing onto Job for processing
AMQProtocolSessionMBean	Needs updated to use AMQProtocolSession without relying on the underlying implementation	
Event, Job	Created by AMQProtocolEngine.received rather than AsynchPoolingReadFilter	

AMQDecoder	Should no longer extend CumulativeProtocolDecoder	
AMQPFastProtocolHandler		AMQP related functionality moved to AMQProtocolEngine
AMQPProtocolProvider		Removed

```

public class AMQProtocolEngineFactory implements ProtocolEngineFactory
{
    // Returns a new instance of an AMQProtocolEngine
    ProtocolEngine newProtocolEngine()
}

```

Changes required to implement AMQProtocolEngineFactory

No existing code needs to be changed here. The factory will simply create an AMQProtocolEngine and return it.

Implement NetworkConfiguration

Class	Method	Change
ServerConfiguration	getNetworkConfiguration	new method, needs to construct and return network configuration information

Transport layer selection in o.a.q.server.Main

Currently the broker's Main method contains setup code specific to MINA.

Main needs to be modified to remove the MINA specific option (NIO, MultiIO, executor pool etc) processing from there. This should be replaced with creation of a Network Driver instance with configuration being picked up & applied in the same way as other subsystems are configured ie. from ServerConfiguration.

Class	Method	Change
Main	bind	Moved to NetworkDriver
Main	startup	Network configuration moved to NetworkDriver, replaced with construction of NetworkDriver and calling MINANetworkDriver.bind

Port server to new interface notes

ID	Raised By	Description	Status	Outcome
1	RG	Overview not helpful, should reflect contents of doc	Done	Removed, replaced with ToC
2	MM	Overview points 3/4 don't clearly relate to sections of doc	Done	Removed
3	MM	Highlight no changes to configuration file	Done	Added to objectives
4	MM	Overview should be Table of Contents	Done	Removed, replaced with ToC
5	RG	MINANetworkDriver should talk about how it's implemented using Mina	Done	Added implementation details
6	MM	Implementation of ProtocolEngine should highlight new code / changed code	Done	Added detailed list of code changes
7	MM	Describe the work / be clear on how they hang together		
8	MM	List of tasks between encapsulate / detailed changes, approach work in discrete tasks	Done	Split work on a class-by-class basis
9	RG	Intermediate steps unlikely to be useful	Just a note	
10	MM	Taks list needs to relate to document content	Done	Task list now integral part of structure
11	AS/RG	Split into better tasks - new interfaces, heartbeating, MINANetworkDriver, ProtocolEngine		
12	MM	Diagram should show difference between current state and new state	Done	Diagram coloured in

13	MM	Tasks should related to overview, remove dependency on MINA	Done	
14	LB	Highlight this is phase 1	Done	In Network IO page

Port server to new interface tests

- Test Objectives
- Performance Test Plan
- Existing System Tests
 - ConnectionTest
 - AcknowledgeTest
 - MultipleConnectionTest
 - SimpleACLTTest
- New system tests required
 - ServerHeartbeatTest
 - testClientWriteldleTest
- Existing Unit Tests
- New unit tests required
 - MINANetworkDriver tests
 - testBindOpen
 - testBindSocketInUse
 - testSend
 - testSetReadIdle
 - testSetWriteldle
 - testClosed
 - testExceptionCaught
 - testGetRemoteAddress
 - AMQDecoder tests
 - testDecodePI09
 - testDecodePI08
 - testDecodePartialDataBlock
 - testDecodeCompleteDataBlock
 - testEncodeFrame
 - AMQProtocolEngine tests
 - testPartialReceived
 - testCompleteReceived
 - testReaderldle
 - testWriterldle
 - testGetRemoteAddress
 - AMQProtocolEngineMbean tests
 - testGetRemoteAddress
 - NetworkDriverConfiguration tests

Test Objectives

This initial phase of work involves minimal changes to existing classes and behaviour. There should be no impact on performance and existing functionality should remain unchanged. The only significant change is the stage at which the byte stream is encoded and decoded. Testing needs to verify that there has not been a negative affect on the fundamental parts of connection processing: connection establishment, encoding, decoding, frame handling, virtualhost selection failure, authentication failure, authorization failure and handling multiple connections at the same time.

Performance Test Plan

There should be no impact on performance since the underlying implementation and threading model remains the same. The performance test suite will be used to validate that there has not been a negative impact on performance.

Existing System Tests

There should be no change in functionality, the existing system tests should continue to function as is. System tests will present the main validation that this work has not had any negative impact. All system tests which create a connection to a broker will exercise the changed code.

In particular the following test cases contain tests which fully exercise the changed code paths in the broker:

ConnectionTest

This TestCase contains:

1. testSimpleConnection which verifies that a basic AMQP connection can be created establishing that:
 - a. TCP connections can be created from client to broker
 - b. Protocol negotiation continues to work
 - c. The broker can decode and encode frames properly
 - d. The broker is properly handling decoded frames
2. testPasswordFailureConnection which verifies that a connection with incorrect authentication fails to be established and throws the correct exception
3. testUnresolvedVirtualHostFailure which verifies that a connection to the wrong virtual host fails to be established and throws the correct exception

AcknowledgeTest

This TestCase contains:

1. test2ConsumersAutoAck which verifies that multiple connections doing input/output simultaneously works
2. test2ConsumersTx which verifies that multiple connections doing input/output simultaneously inside a transaction works

MultipleConnectionTest

This TestCase contains:

1. test which verifies that many connections reading and writing to the broker at the same time works

SimpleACLTest

This TestCase contains multiple tests which will verify that connection attempts where authorization fails continue to throw the correct exception.

New system tests required

No system tests exist which test the AMQP heartbeating functionality.

ServerHeartbeatTest

testClientWriteldleTest

This test will open a client connection and let it idle for longer than the timeout period. It will verify that the connection remains usable after this idle period.

Existing Unit Tests

Existing unit test coverage of the classes involved is minimal to non-existent. I will add tests for the changed functionality, but not for the existing functionality which has not been modified. org.apache.mina.common.ByteBuffer has an existing comprehensive unit test which will be imported along with it.

New unit tests required

Unit tests will need to be written for MINANetworkDriver, AMQProtocolEngine and AMQDecoder.

MINANetworkDriver tests

The unit test will create a ProtocolEngine to implement a simple echo server which will send back any received data.

- `testClientWriteldleTest`
- `testBindOpen`
- `testBindSocketInUse`
- `testSend`
- `testSetReadIdle`
- `testSetWriteldle`
- `testClosed`
- `testExceptionCaught`
- `testGetRemoteAddress`

testBindOpen

This test will create two NetworkDrivers, one of which binds to a socket and one of which opens a socket. This test will assert that the open fails before the bind and that open succeeds after the bind.

testBindSocketInUse

This test will create two NetworkDrivers, bind one to a port and check that that succeeds. It will then attempt to bind the second NetworkDriver to the same port and verify that BindException is thrown.

testSend

This test will create one NetworkDriver and call its send method and verify that the data is passed to the ProtocolEngines receive() method.

testSetReadIdle

This test will create one NetworkDriver and set its read idle timeout. It will verify that the ProtocolEngines readidle method is called after the appropriate time.

testSetWriteldle

This test will create one NetworkDriver and set its write idle timeout. It will verify that the ProtocolEngines writeldle method is called after the appropriate time.

testClosed

This test will create one NetworkDriver and close it. It will verify that the ProtocolEngines closed method is called.

testExceptionCaught

This test will create one NetworkDriver bind it and open a socket. It will forcibly close the socket to generate an exception. It will verify that the ProtocolEngines exception method is called.

testGetRemoteAddress

This getRemoteAddress method and returns a SocketAddress that corresponds to localhost.

AMQDecoder tests

- [testDecodePI09](#)
- [testDecodePI08](#)
- [testDecodePartialDataBlock](#)
- [testDecodeCompleteDataBlock](#)
- [testEncodeFrame](#)

testDecodePI09

This test will create a byte buffer containing an AMQP 0-9 protocol header and check that testDecode returns a ProtocolInitiation with the protocolMajor set to 0 and protocolMinor set to 9.

testDecodePI08

This test will create a byte buffer containing an AMQP 0-9 protocol header and check that testDecode returns a ProtocolInitiation with the protocolMajor set to 0 and protocolMinor set to 8.

testDecodePartialDataBlock

This test will create a byte buffer containing a partial data block and verify that doDecode returns null to indicate that the data should be held until more arrives.

testDecodeCompleteDataBlock

This test will create a byte buffer containing a complete AMQP data block and verify that doDecode returns an AMQFrame of the appropriate type.

testEncodeFrame

This test will create an AMQP frame and verify that encodeFrame returns a byte array with the appropriate contents.

AMQProtocolEngine tests

- [testPartialReceived](#)
- [testCompleteReceived](#)
- [testReaderIdle](#)
- [testWriterIdle](#)
- [testGetRemoteAddress](#)

These tests will use a MockNetworkDriver to test the functionality of the ProtocolEngine parts of AMQProtocolEngine.

testPartialReceived

This test will pass in a ByteBuffer containing a partial AMQP frame and check that the frame handler is not called. It will then pass in a second ByteBuffer containing the rest of the AMQ frame and check that the frame handler is called.

testCompleteReceived

This test will pass in a ByteBuffer containing a complete AMQP frame and check that the frame handler is called.

testReaderIdle

This test will call the ProtocolEngines readerIdle method and check that the ProtocolEngine closes itself

testWriterIdle

This test will call the ProtocolEngines writerIdle method and check that the ProtocolEngine calls the NetworkDriver.send() method with an encoded HeartBeatBody frame.

testGetRemoteAddress

This test will verify that the ProtocolEngine.getRemoteAddress method calls the NetworkDriver.getRemoteAddress method and returns the same data.

AMQProtocolEngineMbean tests

- [testGetRemoteAddress](#)

testGetRemoteAddress

This test will verify that the ProtocolEngineMbean.getRemoteAddress method calls the ProtocolEngine.getRemoteAddress method and returns the same data.

NetworkDriverConfiguration tests

This test case will construct a ServerConfiguration with known values.

h7. testGetKeepAlive

This test will verify that the method returns the expected value.

h7. testGetOOBInline

This test will verify that the method returns the expected value.

h7. testGetReuseAddress

This test will verify that the method returns the expected value.

h7. testGetSoLinger

This test will verify that the method returns the expected value.

h7. testGetSoTimeout

This test will verify that the method returns the expected value.

h7. testGetTcpNoDelay

This test will verify that the method returns the expected value.

h7. testGetTrafficClass

This test will verify that the method returns the expected value.

h7. testGetReceiveBufferSize

This test will verify that the method returns the expected value.

h7. testGetSendBufferSize

This test will verify that the method returns the expected value.

Java Broker Design - Operational Logging

Operational Logging

The current logging configuration in the Java broker is focused for developers. Logging is performed on a class basis and as a result it is not easy to enable logging to get an operational view of the broker. Some work has been done to create configuration files that set the levels on various classes to provide the operational view of the broker (see [Debug using log4j](#)). While this provides some good detail it does not provide the full picture.

The page will document the logging information that would be useful to provide, a suggested approach that should be followed and guidelines to developers for adding operational logging messages.

- Overview
- Log Streams
- Logging Content
 - Status Updates
 - Statistics updates
- Logging Format
- Logging Hierarchy
- Guidelines for logging changes
- Further Design Details

Overview

Currently logging is performed added in an ad-hoc manner by the developer, usually to assist in the development of the code base.

Log messages should be aimed at helping to provide users and/or support staff with information on the health of the broker; and - in the case where there has been some issue - help them diagnose the cause of that issue.

As such these messages should be readable without knowledge of the Qpid code base, they should not be so frequent as to impact the performance of the broker but should be frequent enough such that diagnosis of issues is possible.

Log messages should occur whenever a significant event occurs, for instance the creation or destruction of a connection to the broker. The log message should contain enough information to be able to correlate the message with a business process event. In the case of a connection open the remote address, the login name, the virtual host, and the application id should be included in the log message.

Log messages at **INFO** level and above are expected to be turned on in a production environment.

Log Streams

There are number of data streams that this work should aim to address either directly as part of the standard operation or through a

dynamically configurable change.

- Major status changes reflecting broker state.
- Receive periodic statistical data on broker performance/processing.
- Ability to full track a single message through the broker.

Logging Content

There are two types of logging that are valuable to the operation of a Qpid broker:

1. State updates
2. Statistical updates

The main focus of the logging update is to improve the log file output however it should be remembered that the values should also be easily queried via the management console.

The existing logging within the broker should also be taken into consideration when performing this work. The analysis of this logging can be found [here](#).

Status Updates

The following model objects should log updates on state changes, creation / destruction events, this will usually mean a single log instruction as the event occurs.

- Broker
- VirtualHost
- MessageStore
- Connection
- Channel
- Queue
- Exchange
- Binding

Statistics updates

In addition to status events being logged, we should periodically log statistics. Each model object may have a set of statistics that they wish to report but it is expected that the statistics will be based on the period between logs. There may also be desire to log at more than one interval. i.e once per minute, hour, day. The models objects may record state that can then be reported with the periodic statistics update.

Rate Statistics

The rate values can be reported at each of the levels as highlighted below. However, for Subscription only the outgoing rate makes sense.

Broker

VirtualHost

MessageStore

Connection

Channel

Queue

Exchange

Incoming message rate (count / bytes)

Outgoing message rate (count / bytes)

Min / Max / Average message size

Subscription

Outgoing message rate (count / bytes)

Min / Max / Average message size

Total Statistics

The total values can be reported for a number of objects as listed here, as with the rate values the totals for subscription only make sense for outgoing messages.

Broker

VirtualHost

MessageStore

Connection

Channel

Last Activity / Idle Notification

Total Incoming message (count / bytes)

Total Outgoing message (count / bytes)

Peak Incoming message (count / bytes)

Peak Outgoing message (count / bytes)

Broker

VirtualHost

MessageStore

Number of current connections

Total number of connections made

Subscription

Last Activity / Idle Notification

Total Outgoing messages (count / bytes)

Peak Outgoing message (count / bytes)

In addition there are a number of statistics that can be reported by the various model objects. These values can be included in any periodic report.

Broker

Heap current usage

Heap peak usage

MessageStore

Disk space used

Memory used

Entities Stored (Queue, Exchange, Binding, Message)

Connection

Number of active sessions

Number of current sessions

Number of consumers

Channel

Number of consumers

Queue

Current subscription count

Active subscription count

Binding Count

Current Queue size (count / bytes)

Messages sent (count / bytes)

Exchange

Min / Max / Average message size

Last Activity / Idle Notification

Bound queues

Total Messages sent (count / bytes)

Subscription

Min / Max / Average message size

Last Activity / Idle Notification

Total Outgoing messages (count / bytes)

Peak Outgoing message (count / bytes)

Unacknowledged size (count / bytes)

As will be mentioned in the [guidelines](#) section, any collection and reporting must be mindful of the performance overhead in doing so.

Logging Format

In order to keep the amount of data logged to the essential the message should be short and unambiguous - but easily recognisable. So the following formats are recommend for each model. The UID listed bellow will allow the disambiguation between multiple entries. This value must be human readable so is expected to be an integer value.

Broker

b-

VirtualHost

vh(<name>)

Connection

con:<uid>(<username>, <ip>, <vhost-name>)

Channel

ch:<uid>

Subscription

sub:<uid>(<queue-name>)

Example:

```
[conn:1(guest, 127.0.0.1, /)]/[ch:2]/[sub:1(myqueue)]
```

Logging Hierarchy

When looking at augmenting the logging of the broker it makes sense to take a step back and provide an operational based logging hierarchy *qpid*. **in addition to the developer focused *org.apache.qpid*.**

The hierarchy of loggers should be structured such that it is easy to enable start or stop monitoring at runtime. The suggested hierarchy is as follows. **NOTE** The hierarchy is more of a graph and as a result it is possible to have more than one path to a logger. When implementing care must be taken so that each event is only logged once. i.e. Enabling *Exchange* and *Queue* logging should **not** result in the *Binding*

operation being logged twice, once for the 'queue to exchange' and once for the 'exchange to queue'. In both cases the log statement will be the same however only one instance should actually be logged.

```
apid Broker  
[<username>]  
Connection [<id>] Channel [<id>] Subscription [<id>]  
VirtualHost <name> Binding  
Exchange [<name>]  
Queue [<name>]  
MessageStore
```

This would allow the quick enabling or disabling of the various logging events.

Guidelines for logging changes

To date there has been no discussion around what, who or even when to log and each developer has been left to provide logging that they see fit. As mentioned earlier Log messages should occur whenever a significant event occurs. The log message should contain enough information to be able to correlate the message with a business process event. In the case of a connection open the remote address, the login name, the virtual host, and the application id should be included in the log message.

The log statements should be short and the reader should not need to refer to previous log statements in order to fully understand the situation. i.e. A new consumer log statement should include the connection and virtualhost details rather than just the connection details and so requiring the user to read back to find the details around the connection creation.

All log statements that perform any computation before the log call must be wrapped with the `is<LEVEL>Enabled` calls to remove the computation should the log statement not be required.

When adding a new log statement a comment should be placed before hand highlighting if this is on the critical message routing path to allow reviewers to better gauge the potential impact. In addition the performance suit should be run with and without the log statement to get an actual measure of the impact.

Further Design Details

- Existing Logging Analysis
- Logging Format Design
- Status Update Design

Existing Logging Analysis

Existing Logging Analysis

Taking a look at the the logging levels (extracted details attached) already present in the broker the following log statements should be incorporated in to any detailed design for status logging.

There should be two levels of logging. Standard info (Detailed 'I:' below) is expected to be the default status of the broker. A debug ('D:') setting could also be provided to present more details. This debug level should also be sufficient to address the message tracing goal.

Broker
I:Startup Complete(Broker Ready)/Port Bind

MessageStore
D:Enqueue
D:Dequeue

Channel
D:MessageDropping(DeadLetter)

Queue
D:Enqueue
D:Dequeue

Exchange
D:Route

Subscription
D:Send
D:Rejection

Authentication

I:Auth Success
I:Auth Failure

Management

I:Startup Complete/Port Bind
I:Admin Changes

Logging Format Design

Logging Format Design

This design follows on from the high level design [work](#) to provide a more detailed description of the format that all logged messages will take. The design is split in to two sections:

- LogSubject Detail
- Log Formating

LogSubject Detail

Each LogSubject in the system has a format that for logging its own identifier. It is envisaged that each Model instance will have a LogSubject member variable to act as a cache for their log format.

LogSubject Identifiers

The following is the basic LogSubject identifiers:

LogSubject	Identifier
Broker	b
MessageStore	ms
VirtualHost	vh(<name>)
Connection	con:<uid>(<username>@<ip>/<vhost-name>)
Channel	ch:<uid>
Queue	qu(<queueName>)
Exchange	ex(<exchangeName>)
Binding	bd(<routingKey>)
Subscription	sub:<uid>:qu(<queueName>)
Plugin	pl(name[, <optional values>]*)

The plugin format allows for simple identification of the plugin such as 'ACL', 'Firewall' as well as giving the plugin the option to extend its base format. This extension is to allow easy processing of the log file.

LogSubject Paths

Broker, VirtualHost & Connection are root identifiers which means no parent nodes need be pre-appended to the log statement.

LogSubject	Log Path	Example
MessageStore	vh(name)/ms	[vh(/)ms]
Channel	<Connection>/ch:<uid>	[con:1(user@127.0.0.1)/ch:1]
Queue	<Virtualhost>/qu(<queueName>)	[vh(/)qu(testQueue)]
Exchange	<Virtualhost>/ex(<exchangeName>)	[vh(/)ex(amq.direct)]
Binding	<Virtualhost>/<Exchange>/<Queue>/bd(<routingKey>)	[vh(/)ex(amq.direct)/qu(testQueue)/bd(testQueue)]
Subscription	sub:<uid>:qu(<queueName>)	[sub:2:qu(testQueue)]
Plugin	<Entity>/pl(name[, <optional values>]*)	[pl(ACL, Consume, qu(testQueue))]

The plugin entity allows for plugins to log additional details about their operation on an entity. For example as shown above an ACL plugin can log details about the attempt to consume from Queue 'testQueue'.

Log Formatting

To ensure that all log messages are displayed consistently the logging framework will provide the Datetime and Entity details, the requested log message will be added to the end this preamble:

Log Format
<ISO-8601 Datetime (UTC based w/ TZ)> <Logging Level> [<LogActor>] [<LogSubject>] <LogMessage>
Example Log Statement
2009-06-29 13:35:10,1234 +0100 Message [con:1(user@127.0.0.1)/ch:2] [sub:1:qu(myqueue)] Subscription Event Occurred

Status Update Design

Status Update

Following on from the high level [design](#), this page will provide a more detailed design approach to implement status update logging in the Java broker.

The logging [hierarchy](#) identified is not suitable to be directly used by Log4j as there is multiple routes and loops in the graph.

Abstracting the logging is recommended as this will allow us to simply provide Qpid specific optimisations such as providing the log prefix.

This design will cover the following areas:

Contents

- Logging Configuration
- Status Updates
- Logging Abstraction
- Logging Usage
- Initial Status Messages

Additional Documentation

- Logging Format Design
- Functional Specification
- Test Plan
- Test Specification
- Technical Specification

Logging Configuration

At this stage configuration will be limited to the addition to the main config.xml file of the following option:

```

<broker>
  ...
  <status-updates>ON</status-updates>
  ...
</broker>
```

This `status-update` on setting will be the default value if the section is not included in the configuration file. The setting will be global for all virtualhosts and will be exposed via the management console as logger 'qpid.status' to allow dynamic setting.

The ability to configure more fine grained logging will be investigated [here](#), but will not be implemented in the initial phase.

Status Updates

In the first phase updates only status updates will be provided. Status updates should take the form of general operational logging level, no logging on message delivery path way and No performance impact. The recommendation will be to have these enabled for production use. e.g. Creation/Destruction events

The status updates can also be used in a second phase to provide additional logging to assist development.

The additional logging can be performed on the message delivery path way. This may have performance impact and so would not be recommended for long term production use.

e.g. Message Enqueue/Dequeue

Logging Abstraction

The abstraction layer allows us to fully decouple the logging mechanism from any operational logging that the broker may wish to perform. The following code highlights show how we would abstract the logging operations.

The approach to logging is that a `LogActor` will be recorded as a `ThreadLocal` and will be used to perform logging the log messages as highlighted on [Logging Format Design](#). The `LogActor` will take two parameters, the `LogSubject` and the `LogMessage`. When a Status event occurs that should be logged the `LogActor` can be retrieved from the thread thus avoiding passing the `LogActor` through as a parameter to all locations were it must be logged. The initial `LogActors` will be **AMQPActor** and **ManagementActor**. Later phases would introduce **HouseKeepingActor**. These `LogActors` are responsible for checking that the logging should be performed for both themselves and the

LogSubject. The *LogActor* then provides their log formatted name as per the format [design](#) along with the message to the *RootMessageLogger*. Initially the configuration will be a simple on/off, however, in a future phase the details can be used to identify if logging should proceed for that *LogActor* and *LogSubject* combination. At this stage selective configurations is not part of this design.

The use of the *LogActor* allows for situations such as *Binding* to have a *Connection* associated with the *Binding*. This will allow a *Binding* create event to be logged like this:

```
2009-06-29 13:35:10,1234 +0100 MESSAGE [con:1(guest@127.0.0.1)/ch:2]
[ex(amq.direct)/qu(testQueue)/bd(routingKey)] BND-1001 : Binding Created
```

rather having no details about how the creation occurred:

```
2009-06-29 13:35:10,1234 +0100 MESSAGE [ vh(/)/ex(amq.direct)/qu(testQueue)/bd(routingKey) ]
BDN-1001 : Create
```

Interfaces

LogActor

```
/**
 * LogActor the entity that is stored as in a ThreadLocal and used to perform logging.
 *
 * The actor is responsible for formatting its display name for the log entry.
 *
 * The actor performs the requested logging.
 */
public interface LogActor
{
    /**
     * Logs the specified LogMessage about the LogSubject
     *
     * Currently logging has a global setting however this will later be revised and
     * as such the LogActor will need to take into consideration any new configuration
     * as a means of enabling the logging of LogActors and LogSubjects.
     *
     * @param actor The actor that is requesting the logging
     * @param message The message to log
     */
    public void message(LogSubject subject, LogMessage message);
}
```

LogSubject

```
/**
 * Each LogSubject that wishes to be logged will implement this to provide their
 * own display representation.
 *
 * The display representation is retrieved through the toString() method.
 */
public interface LogSubject
{
    /**
     * Logs the message as provided by String.valueOf(message).
     *
     * @returns String the display representation of this LogSubject
     */
    public String toString();
}
```

RootMessageLogger

```
/**  
 * The RootMessageLogger is used by the LogActors to query if  
 * logging is enabled for the requested message and to provide the actual  
 * message that should be logged.  
 */  
public interface RootMessageLogger  
{  
    /**  
     * Determine if the LogSubject and the LogActor should be  
     * generating log messages.  
     *  
     * @param logSubject The subject of this log request  
     * @param logActor The actor requesting the logging  
     * @return boolean true if the message should be logged.  
     */  
    boolean isMessageEnabled(LogActor actor, LogSubject subject);  
  
    /**  
     * Log the raw message to the configured logger.  
     *  
     * @param message The message to log  
     * @param throwable Optional Throwable that should provide stack trace  
     */  
    void rawMessage(String message, Throwable throwable);  
}
```

RawMessageLogger

```
/**  
 * A RawMessage Logger takes the given String and any Throwable and writes the  
 * data to its resource.  
 */  
public interface RawMessageLogger  
{  
    /**  
     * Log the message and formatted stack trace for any Throwable.  
     *  
     * @param message String to log.  
     * @param throwable Throwable for which to provide stack trace.  
     */  
    public void rawMessage(String message, Throwable throwable);  
}
```

Logging Usage

Logging of a Channel Creation

```
public class Connection  
{  
    ...  
    LogActor amqpActor = // retrieved from ThreadLocal.  
    //_channelSubject is an instance LogSubject that knows how to represent this Connection  
    amqpActor.logMessage(_connectionSubject, LogMessages.CHANNEL_CREATE(this));  
    ...  
}
```

Would result in the following based on the Logging Format Design.

```
2009-06-29 13:35:10,1234 +0100 MESSAGE [con:1(guest@127.0.0.1/)] [ch:2] ChM-1001 : Channel Created
```

Logging of a new consumer creation

```
...  
    amqpActor.logMessage(_subscriptionSubject, LogMessages.SUBSCRIPTION_CREATE(this));  
...
```

Would result in the following:

```
2009-06-29 13:35:10,1234 +0100 MESSAGE [con:1(guest@127.0.0.1)/ch:2] [sub:1:qu(myqueue)] Sub-1001  
: Subscription Created
```

Initial Status Messages

Broker

Startup

Configuration details

Ready

Shutdown

ManagementConsole

Startup

Configuration details

Ready

Close

VirtualHost

Create

Configuration details

Close

MessageStore

Startup

Recover Status

Start

Progress

End

Close

Connection

Open

Close

Channel

Create

Flow Status

Destroy

Queue

Create

Destroy

Exchange

Create

Destroy

Binding

Create

Destroy

Subscription

Create

Destroy

Operational Logging - Status Update - Functional Specification

Functional Specification

This page documents the functional specification for the status updates improvement to the Java Broker.

Log Messages

This is the list of initial status messages that the broker will be configured to produce at for status logging. These messages will be parameterised as shown and will be accessed via an interface so that we need only maintain the text in a single location. While the messages here do no show these standardised messages will also allow for easy internationalisation. Each section includes the expected full format of a log message. So taking the *Broker* logging as an example an entry in the log file for the startup of a 0.6 release broker would be:

```
2009-07-09 15:50:20 +0100 MESSAGE BRK-1001 : Startup : Version 0.6 Build: exported
```

The expected format is shown at the start of each section in *italics*. This is then followed by the list of messages that can be logged. This formatting (as fully defined [here](#)) is the reason that each log message that follows does not need to contain a lot of details. For example, when logging the creation of a Channel you would want to know more details than just the prefetch count hence when the message is logged it would look like this:

```
2009-07-09 15:50:20 +0100 MESSAGE [ con:1(guest@127.0.0.1/test)/ch:2 ] CHN-1001 : Create :  
Prefetch 400
```

Broker

```
<DATETIME> MESSAGE <Message>  
BRK-1001 : Startup : Version: <Version> Build: <Build>  
BRK-1002 : Starting : Listening on <Transport> port <Port>  
BRK-1003 : Shuting down : <Transport> port <Port>  
BRK-1004 : Ready  
BRK-1005 : Stopped  
BRK-1006 : Using configuration : <path>  
BRK-1007 : Using logging configuration : <path>
```

ManagementConsole

```
<DATETIME> MESSAGE <Message>  
MNG-1001 : Startup  
MNG-1002 : Starting : <service> : Listening on port <Port>  
MNG-1003 : Shuting down : <service> : port <Port>  
MNG-1004 : Ready  
MNG-1005 : Stopped  
MNG-1006 : Using SSL Keystore : <path>
```

VirtualHost

```
<DATETIME> MESSAGE [ vh:<(name)> ] <Message>  
VHT-1001 : Created : <name>  
VHT-1002 : Closed
```

MessageStore

```
<DATETIME> MESSAGE [ vh:<(name)> ] <Message>  
MST-1001 : Created : <name>  
MST-1002 : Store location : <path>  
MST-1003 : Closed  
MST-1004 : Recovery Start [: <queue.name>]  
MST-1005 : Recovered <count> messages for queue <queue.name>  
MST-1006 : Recovery Complete [: <queue.name>]
```

Connection

```
<DATETIME> MESSAGE [ con:1(guest@127.0.0.1/test) ] <Message>  
CON-1001 : Open : Client ID <id> : Protocol Version : <version>  
CON-1002 : Close
```

Channel

```
<DATETIME> MESSAGE [ con:1(guest@127.0.0.1/test)/ch:2 ] <Message>  
CHN-1001 : Create : Prefetch <count>  
CHN-1002 : Flow <value>  
CHN-1003 : Close
```

Queue

```
<DATETIME> MESSAGE [ con:1(guest@127.0.0.1/test)/ch:2/qu(myqueue) ] <Message>  
QUE-1001 : Create : [AutoDelete] [Durable|Transient] [Priority:<levels>] Owner:<name>  
QUE-1002 : Deleted
```

Exchange

```
<DATETIME> MESSAGE [ con:1(guest@127.0.0.1/test)/ch:2/ex(amq.direct) ] <Message>  
EXH-1001 : Create : [Durable] Type:<value> Name:<value>  
EXH-1002 : Deleted
```

Binding

```
<DATETIME> MESSAGE [ con:1(guest@127.0.0.1/test)/ch:2/ex(amq.direct)/qu(myQueue)/rk(myQueue) ] <Message>  
BND-1001 : Create [: Arguments : <key=value>]  
BND-1002 : Deleted
```

Subscription

<DATETIME> MESSAGE [con:1(guest@127.0.0.1/test)/ch:2/sub:1:qu(myqueue)] <Message>

SUB-1001 : Create : [Durable] [Arguments : <key=value>]

SUB-1002 : Close

Comments

ID	by	Summary	Status
1	Robbie	MC has two ports one for RMI Registry, one for RMI ConnectorServer	I've parameterized startup/shutdown to take a service value. So I'd expect two entries in the log for our current MC
2	Robbie	MC IDs are not unique	fixed

Operational Logging - Status Update - Technical Specification

Technical Specification

- Overview
- New classes
 - Interface list
 - Class List
 - Psuedo-Code Example
- How to provide fixed log messages
- Additions to existing classes
 - Main
 - ApplicationRegistry
 - ConfigurationFileApplicationRegistry
 - JMXManagedObjectRegistry
 - VirtualHost
 - DerbyMessageStore/MemoryMessageStore
 - DerbyMessageStore
 - AMQMinProtocolSession
 - AMQChannel
 - QueueRegistry
 - AbstractExchange
 - ExchangeBindings
 - SubscriptionImpl
- Deletions from existing classes
 - AMQMinProtocolSession
 - AMQPFastProtocolHandler
 - BasicConsumeMethodHandler
 - ChannelFlowHandler.java:
 - Configuration
 - ConnectionCloseMethodHandler
 - DerbyMessageStore
 - HeadersExchange
 - JMXManagedObjectRegistry
 - Main
 - MemoryMessageStore
 - QueueBindHandler
 - QueueDeclareHandler
 - QueueUnbindHandler
 - SimpleAMQQueue
 - SubscriptionImpl
 - VirtualHost
- Feedback

Overview

This technical specification page will detail four areas of work to complete the Status Update change:

- The new classes required
- How to provide fixed log messages
- The additions to existing classes
- The deletions from existing classes

New classes

The new classes required draws on the [design](#) work already completed. The abstraction layer will be the new code created as part of this work. This can be split in to Interfaces and Classes.

Interface list

These interfaces form the abstraction layer.

Interface	Description
-----------	-------------

LogActor	Actor that will perform the logging.
LogSubject	Subject of the logging .
LogMessage	Factory to generate LogMessages.
RootMessageLogger	Root logger that performs logging.
RawMessageLogger	Wrapper for object that actually performs the logging.

Class List Implementation of the Actors

Class	Description
AMQPActor	Responsible for providing data about the Connection when logging.
ManagementActor	Responsible for providing data about the Management Connection when logging.

Implementation of the LogSubject

Each of these *LogSubjects* will ensure they `toString` according to the format design.

Class	Description
ConnectionLogSubject	Logger responsible for the Connection format.
ChannelLogSubject	Logger responsible for the Channel format.
QueueLogSubject	Logger responsible for the Queue format.
ExchangeLogSubject	Logger responsible for the Exchange format.
BindingLogSubject	Logger responsible for the Binding format.
SubscriptionLogSubject	Logger responsible for the subscription format.
MessageStoreLogSubject	Logger responsible for the MessageStore format.
RootMessageLoggerImpl	Base logger that performs the final message formatting before logging.
LogMessageFactoryImpl	Factory to create the LogMessages.

Logging

Class	Description
Log4jRawMessageLogger	Wrapper to use log4j as the output mechanism.
TestRawMessageLogger	Wrapper that provides an inspectable log for testing.

Log Messages

Class	Description
BrokerLogMessages	A static class that contains accessors to the various parametrised log messages.

Pseudo-Code Example

```

// logActor is retrieved from the ThreadLocal
// a logMessage of type logMessage (with parms) is then requested for the specified subject.
logActor.logMessage(logSubject, LogMessage_parms)

...
instance of LogActor{

RootLogger logger = ...getRootLogger();L

public void logMessage(LogSubject subject, LogMessage message)
{
    if (logger.isMessageEnabled(this, subject)
    {
        // FormatMessage in to :
// MESSAGE [ this.toString() ] [ subject.toString() ] <messageID> : <message value>
logger.logMessage(FormatMessage(this, subject, message));
    }
}
}

```

How to provide fixed log messages

The [design](#) calls for providing a fixed method of accessing the messages. Such as the following

```

String version="0.6";
int build=794277;
String message = BrokerLogMessages.BRK-1001(version, build);

```

The value of *message* above would be

```
BRK-1001 : Startup : Version: 0.6 Build: 794277
```

This can be done easily with the use of a *MessageFormatter* and a property file.

```
BRK-1001 = Startup : Version {0} Build: {1}
```

Initially the BrokerLogMessages class could be hand coded but in a future iteration it could be generated based on the content of the property file.

Additions to existing classes

The following classes will have logging added to provide the required log messages specified in the [Functional Specification](#).

Main

```

BRK-1001 : Startup : Version: <Version> Build: <Build>
BRK-1004 : Ready
BRK-1007 : Using logging configuration : <path>

```

ApplicationRegistry

```

BRK-1002 : Starting : Listening on <Transport> port <Port>
BRK-1003 : Shuting down : <Transport> port <Port>
BRK-1005 : Stopped

```

ConfigurationFileApplicationRegistry

```
BRK-1006 : Using configuration : <path>
```

JMXManagedObjectRegistry

```
MNG-1001 : Startup  
MNG-1002 : Starting : <service> : Listening on port <Port>  
MNG-1003 : Shuting down : <service> : port <Port>  
MNG-1004 : Ready  
MNG-1005 : Stopped  
MNG-1006 : Using SSL Keystore : <path>
```

VirtualHost

```
VHT-1001 : Created : <name>  
VHT-1002 : Closed
```

DerbyMessageStore/MemoryMessageStore

```
MST-1001 : Created : <name>  
MST-1003 : Closed
```

DerbyMessageStore

```
MST-1002 : Store location : <path>  
MST-1004 : Recovery Start [: <queue.name>]  
MST-1005 : Recovered <count> messages for queue <queue.name>  
MST-1006 : Recovery Complete [: <queue.name>]
```

AMQMinProtocolSession

```
CON-1001 : Open : Client ID <id> : Protocol Version : <version>  
CON-1002 : Close
```

AMQChannel

```
CHN-1001 : Create : Prefetch <count>  
CHN-1002 : Flow <value>  
CHN-1003 : Close
```

QueueRegistry

```
QUE-1001 : Create : [AutoDelete] [Durable|Transient] [Priority:<levels>] Owner:<name>  
QUE-1002 : Deleted
```

AbstractExchange

```
EXH-1001 : Create : [Durable] Type:<value> Name:<value>  
EXH-1002 : Deleted
```

ExchangeBindings

```
BND-1001 : Create [: Arguments : <key=value>]  
BND-1002 : Deleted
```

SubscriptionImpl

```
SUB-1001 : Create : [Durable] [Arguments : <key=value>]  
SUB-1002 : Close
```

Deletions from existing classes

The following log statements should be removed from the broker packages as they are being replaced with a new message.

AMQMinProtocolSession

```
_logger.info("Channel[" + channelId + "] awaiting closure - processing close-ok");
_logger.info("Closing channel due to: " + e.getMessage());
_logger.info("Closing connection due to: " + e.getMessage());
_logger.info("Closing connection due to: " + e.getMessage());
_logger.debug("REALLY Closing protocol session:" + _minaProtocolSession);
```

AMQPFastProtocolHandler

```
_logger.info("Protocol session created for:" + protocolSession.getRemoteAddress());
_logger.info("Session opened for:" + protocolSession.getRemoteAddress());
_logger.info("Protocol Session closed for:" + protocolSession.getRemoteAddress());
_logger.debug("AMQPFastProtocolHandler created");
```

BasicConsumeMethodHandler

```
_logger.debug("BasicConsume: from '" + body.getQueue() +
_logger.debug("No queue for '" + body.getQueue() + "'");
_logger.debug("Closing connection due to invalid selector");
```

ChannelFlowHandler.java:

```
_logger.debug("Channel.Flow for channel " + channelId + ", active=" + body.getActive());
```

Configuration

```
_devlog.info("Configuring logger using configuration file " + logConfigFile.getAbsolutePath());
_devlog.info("log file " + logConfigFile.getAbsolutePath() + " will be checked for changes every
_devlog.debug("Using configuration file " + _configFile.getAbsolutePath());
ex.getMessage());
```

ConnectionCloseMethodHandler

```
_logger.info("ConnectionClose received with reply code/reply text " + body.getReplyText() + " for
" + session);
```

DerbyMessageStore

```
_logger.info("Configuring Derby message store for virtual host " + virtualHost.getName());
_logger.info("Recovering persistent state...");
_logger.info("Persistent state recovered successfully");
_logger.info("Recovering durable exchange " + exchange.getName() + " of type " +
exchange.getType() + "...");
_logger.info("Restoring binding: (Exchange: " + exchange.getName() + ", Queue: " + queueName);
_logger.info("Recovered message counts: " + queueRecoveries);
_logger.debug("public void createQueue(AMQQueue queue = " + queue + "): called");
_logger.debug("public void removeQueue(AMQShortString name = " + name + "): called");
_logger.debug("On recovery, delivering " + message.getMessageId() + " to " + queue.getName());
```

HeadersExchange

```
_logger.debug("Exchange " + getName() + ": Unbinding " + queue.getName());
_logger.debug("Exchange " + getName() + ": routing message with headers " + headers);
```

JMXManagedObjectRegistry

```
log.info("Initialising managed object registry using platform MBean server");
_log.info("JMX ConnectorServer using SSL keystore file " + ksf.getAbsolutePath());
_startupLog.info("JMX ConnectorServer using SSL keystore file " + ksf.getAbsolutePath());
```

Main

```

_brokerLogger.info("Starting Qpid Broker " + QpidProperties.getReleaseVersion());
_brokerLogger.info("Qpid.AMQP listening on non-SSL address " + bindAddress);
_brokerLogger.info("Qpid.AMQP listening on SSL port " + config.getSSLPort());
_brokerLogger.info("Qpid Broker Ready :" + QpidProperties.getReleaseVersion())

```

MemoryMessageStore

```

_log.info("Using capacity " + DEFAULT_HASHTABLE_CAPACITY + " for hash tables");
_log.info("Using capacity " + hashtableCapacity + " for hash tables");

```

QueueBindHandler

```

_log.info("Binding queue " + queue + " to exchange " + exch + " with routing key " + routingKey);

```

QueueDeclareHandler

```

_logger.info("Queue " + queueName + " bound to default exchange(" + defaultExchange.getName() +
 ")");
_logger.info("Queue " + queueName + " declared successfully");

```

QueueUnbindHandler

```

_log.info("Binding queue " + queue + " to exchange " + exch + " with routing key " + routingKey);

```

SimpleAMQQueue

```

_logger.info("Auto-deleteing queue:" + this);

```

SubscriptionImpl

```

_logger.info("Closing subscription (" + debugIdentity() + "):" + this);

```

VirtualHost

```

_logger.info("Binding queue:" + queue + " with routing key '" + routingKey + "' to exchange:" + this);
_logger.debug("Loading configuration for virtualhost: " + config.getName());

```

Feedback

ID	From	Comment	Response
1	Marnie	Ensure data logged in messaged due to be deleted is not lost.	
2	Marnie	Following 1: Exception handling messages need further thought as replacing with 'Close' messages loses the cause	
3	Marnie	Be mindful of performance in generating these log messages	
4	Marnie	It is not clear that the LogActors, LogSubjects will be created and attached to their repetitive model objects	

Operational Logging - Status Update - Test Plan

Test Plan

This plan is to test the new logging functionality presented in the [Status Update Design](#).

- Test Objectives
- Performance Testing
- Operation Testing
- Performance Test Plan
- Operational Test Plan
- Test Specification

Test Objectives

This plan will define the various areas that must be tested to validate the new logging meets its requirements. This information will be used during the subsequent technical design and development phases to ensure that the testing approaches defined in this plan are possible.

The plan will focus on two areas:

- Performance Testing
- Operation Testing

Performance Testing

One of the biggest risks of adding more logging to the broker is the potential performance impact they will have in terms of a) creating the messages to log and b) actually logging the message. Therefore validating our changes have a negligible performance impact is key.

Approach

A series of test must be written that cover all the log messages that the broker will generate. The approach to validate any performance changes should then be to run the test multiple times to generate an average performance. The impact of the logging additions must be negligible. The performance test suite has shown that there can be up to 5% variance between test runs so if this testing is to be performed as an automated system test we must be careful to ensure that we do not end up with test failures due to an unusually variance. Testing should be performed to establish a baseline from which we can determine what amount of variance occurs between the two test runs. This variance can then be used to determine the failure criteria. However, it should be noted that such a comparison technique will only ensure that the impact of the logging does not shift between each run. The test will not address any potential drift in performance of the broker as a whole, only the difference between logging on and logging off.

The test should run with interleaved test setups, i.e. Logging On then Logging off. This will help minimise any external factors that could impact the timing. The time spent logging however, will still represent a very small percentage of time for a test case and as such the test will be more susceptible to external factors during the test run.

Operation Testing

There are two components to testing the functionality of the new logging.

1. Unit Testing
2. System Testing

Unit Testing

Unit testing must be completed on each module and code coverage should cover at least 80% (aiming towards 100%) of the code base. The unit testing however, will only verify that the module performs as expected with the use of a test output logger.

System Testing

System testing will need to be performed to validate that the correct log messages appear in the right order when run as a full system. By using the test output logger used for the Unit testing it will be possible to validate an InVM broker correctly logs at the appropriate time. To complete system testing the log4j output from an external broker test run must be examined to ensure it contains the expected output. The alerting tests already perform this sort of external log validation so this should be easy to replicate.

Additionally, we will want to add tests to understand how the system behaves under certain failure conditions. This will not be required as part of this initial work. However, when we remove log4j we need to understand what the differences will with any new logging framework in failure situations, i.e. disk full, disk loss(crash, NFS delay). The difference with a disk loss (crash, NFS delay) is the write IO will not necessarily respond immediately while a disk full notification will usually respond immediately.

Performance Test Plan

The performance of the broker can be measured by the JMS Client by ensuring that all the updated modules are executed in the test run. The time it takes to startup create a new JMS Session and subscribe and the cleanly shut everything down will ensure all of the new log messages have the option of logging. Now the overhead in broker startup and shutdown will have a large effect on the test run however this should mean it will be easier to compare startup times as they should be very similar.

Operational Test Plan

The [Functional Specification](#) lists the various status log messages that the broker will be configured to produce. What this section details is how testing will be carried out for each of these log statements.

Broker

The *Broker* messages can be split in to two categories. Startup and shutdown messages. The testing here focuses on the *Broker* messages and ignore the fact that during a system testing of the broker other log messages will also be printed.

Startup

As all the startup messages are printed on startup then they cannot be printed individually. So by monitoring the logging output on startup we can verify that messages BKR-1001,1002,1004,1006,1007 are logged. It is expected that the output would be very similar to this:

```
2009-07-09 15:50:20 +0100 MESSAGE BRK-1006 : Using configuration : build/etc/config.xml
2009-07-09 15:50:20 +0100 MESSAGE BRK-1007 : Using logging configuration : build/etc/log4j.xml
2009-07-09 15:50:20 +0100 MESSAGE BRK-1001 : Startup : Version: 0.6 Build: <svn revision>
2009-07-09 15:50:20 +0100 MESSAGE BRK-1002 : Starting : Listening on TCP port 5672
2009-07-09 15:50:20 +0100 MESSAGE BRK-1004 : Ready
```

Additionally testing should be performed with SSL enabled to verify that the additional BRK-1002 message is logged.

```
2009-07-09 15:50:20 +0100 MESSAGE BRK-1002 : Starting : Listening on TCP/SSL port 8672.
```

Shutdown

On shutdown the broker will log a BRK-1003 for each interface that has been started. So if we have started both the TCP and the TCP/SSL listeners then we would expect both to be printed out before the Stopped message.

```
2009-07-09 15:50:20 +0100 MESSAGE BRK-1003 : Shutting down : TCP port 5672
2009-07-09 15:50:20 +0100 MESSAGE BRK-1003 : Shutting down : TCP/SSL port 8672
2009-07-09 15:50:20 +0100 MESSAGE BRK-1005 : Stopped
```

ManagementConsole

The management console also has a startup and shutdown phase like the broker.

Startup

The Management console uses two ports so both of these will be logged at startup. Two test runs must be performed to validate that the 'SSL Keystore' output is only present when correctly enabled.

```
2009-07-09 15:50:20 +0100 MESSAGE MNG-1001 : Startup
2009-07-09 15:50:20 +0100 MESSAGE MNG-1002 : Starting : RMI Registry : Listening on port 8999
2009-07-09 15:50:20 +0100 MESSAGE MNG-1002 : Starting : RMI ConnectorServer : Listening on port
9099
2009-07-09 15:50:20 +0100 MESSAGE MNG-1006 : Using SSL Keystore : test_resources/ssl/keystore.jks
2009-07-09 15:50:20 +0100 MESSAGE MNG-1004 : Ready
```

Shutdown

During shutdown the two listeners created during startup will shutdown. The use of SSL will not provide any additional shutdown message to verify.

```
2009-07-09 15:50:20 +0100 MESSAGE MNG-1003 : Shutting Registry : Listening on port 8999
2009-07-09 15:50:20 +0100 MESSAGE MNG-1003 : Shutting down : RMI ConnectorServer : Listening on
port 9099
2009-07-09 15:50:20 +0100 MESSAGE MNG-1005 : Stopped
```

VirtualHost

Virtualhosts cannot be programmatically created so the log messages will have to be found during the startup/shutdown of the broker.

Startup

During startup the following VHT messages will be logged as the Virtualhost is created.

```
2009-07-09 15:50:20 +0100 MESSAGE [ vh:(test) ] VHT-1001 : Created : test
```

Shutdown

On shutdown the Virtualhost will only log that it has been closed.

```
2009-07-09 15:50:20 +0100 MESSAGE [ vh:(test) ] VHT-1002 : Closed
```

MessageStore

The MessageStore details will be logged as part of the Virtualhost startup and shutdown.

Startup

Aside from the easy to test create(MST-1001) and store location(MST-1002) messages, a persistent store such as DerbyDB will also need to be used for testing so that the recovery messages MST-1004-6 can be tested.

It is expected that on recovery start MST-1004 will be logged without a queue name. Then as each queue is recovered then a start(MST-1004), count (MST-1005) and complete (MST-1006) will be logged. Only when all queues have been recovered will a final complete (MST-1006) be logged. This would give a sequence such as:

```
2009-07-09 15:50:20 +0100 MESSAGE [ vh:(test) ] MST-1001 : Created
2009-07-09 15:50:20 +0100 MESSAGE [ vh:(test) ] MST-1001 : Created : DerbyMessageStore
2009-07-09 15:50:20 +0100 MESSAGE [ vh:(test) ] MST-1002 : Store location : ./derbyDB
2009-07-09 15:50:20 +0100 MESSAGE [ vh:(test) ] MST-1004 : Recovery Start
2009-07-09 15:50:20 +0100 MESSAGE [ vh:(test) ] MST-1004 : Recovery Start \[: qu(myQueue)]
2009-07-09 15:50:20 +0100 MESSAGE [ vh:(test) ] MST-1005 : Recovered 100 messages for qu(myQueue)
2009-07-09 15:50:20 +0100 MESSAGE [ vh:(test) ] MST-1006 : Recovery Complete \[: qu(myQueue)]
2009-07-09 15:50:20 +0100 MESSAGE [ vh:(test) ] MST-1006 : Recovery Complete
```

Shutdown

On shutdown the MessageStore will only log that it has been closed.

```
2009-07-09 15:50:20 +0100 MESSAGE [ vh:(test) ] MST-1003 : Closed
```

Connection

New connections can be easily performed in isolation by connecting to the running broker. The connection open and close log message should be presented in response to the new connection and the closure of the connection.

```
2009-07-09 15:50:20 +0100 MESSAGE [ con:1(guest@127.0.0.1/test) ] CON-1001 : Open : Client ID
client1 : Protocol Version : 0-9
2009-07-09 15:50:20 +0100 MESSAGE [ con:1(guest@127.0.0.1/test) ] CON-1002 : Close
```

Channel

As with Connection creation Channel creation can be tested in isolation. Whilst a channel will be created along with the Connection. A new channel can be created from the Java client by creating a new JMS Session. The current Java client will start all Channels on the JMS Connection flowed. This means that an initial 'CHN-1002 : Flow Stopped' message will be logged. Only when the JMS Connection is started will the Channel unflowed and a 'CHN-1002 : Flow Started' logged.

Additional testing should be done to ensure that CHN-1002 messages are logged when the client exceeds its prefetch count and the broker flows the client.

```
2009-07-09 15:50:20 +0100 MESSAGE [ con:1(guest@127.0.0.1/test)/ch:2 ] CHN-1001 : Create :
Prefetch 500
2009-07-09 15:50:20 +0100 MESSAGE [ con:1(guest@127.0.0.1/test)/ch:2 ] CHN-1002 : Flow Stopped
2009-07-09 15:50:20 +0100 MESSAGE [ con:1(guest@127.0.0.1/test)/ch:2 ] CHN-1002 : Flow Started
2009-07-09 15:50:20 +0100 MESSAGE [ con:1(guest@127.0.0.1/test)/ch:2 ] CHN-1003 : Close
```

Queue

There are a number of properties that can be set on a Queue and as a result they all need to be tested in isolation and validated that the correct log message is generated based on this template:

```
2009-07-09 15:50:20 +0100 MESSAGE [ con:1(guest@127.0.0.1/test)/ch:2/qu(myqueue) ] \\
QUE-1001 : Create : [AutoDelete] [Durable|Transient] [Priority:<levels>]
Owner:<name>
```

Property combinations to test:

- Durable | Transient
 - AutoDelete
 - Priority

This results in 8 tests as each Combination of AutoDelete|Priority|None is tested against Durable|Transient.

A simple deleted is logged when the queue is finally deleted.

```
2009-07-09 15:50:20 +0100 MESSAGE [ con:1(guest@127.0.0.1/test)/ch:2/qu(myqueue) ] QUE-1002 :
Deleted
```

Exchange

As with Queue logging the Exchange has the Durable option that must be tested independently. During broker startup the default exchanges will be created and so will be logged. Testing should ensure that these log messages are indeed correctly logged. However as Exchanges can be programmatically declared this should also be tested.

```

2009-07-09 15:50:20 +0100 MESSAGE [ con:1(guest@127.0.0.1/test)/ch:2/ex(amq.direct) ] EXH-1001 :
Create : Durable Type:amq.direct Name:amq.direct
2009-07-09 15:50:20 +0100 MESSAGE [ con:1(guest@127.0.0.1/test)/ch:2/ex(amq.direct) ] EXH-1001 :
Create : Type:amq.direct Name:amq.direct
2009-07-09 15:50:20 +0100 MESSAGE [ con:1(guest@127.0.0.1/test)/ch:2/ex(amq.direct) ] EXH-1002 :
Deleted

```

Binding

Bindings will be created via JMS in the Java client along side Queue and Subscriber creation. The Qpid Java Client uses the Arguments field to ensure exclusive queues do not get filled with messages when a selector is in use. This adds another dimension to the testing as part of the system testing needs to include validation of the message prefix (content between []) as testing needs to cover the binding of Queues to a different exchanges and with a variety of routing keys.

This gives us the following dimensions of testing that needs to be performed:

- Exchange
 - Arguments
 - RoutingKey

If we take bindings between amq.direct and amq.topic and vary the routing key (for Queues the default is the queue name when used with the Java Client). We have 4 test cases that we need to run on Exclusive and Non-Exclusive queues.

```

2009-07-09 15:50:20 +0100 MESSAGE [
con:1(guest@127.0.0.1/test)/ch:2/ex(amq.direct)/qu(myQueue)/rk(myQueue) ] \\
BND-1001 : Create [: Arguments : <key=value>]
2009-07-09 15:50:20 +0100 MESSAGE [
con:1(guest@127.0.0.1/test)/ch:2/ex(amq.direct)/qu(myQueue)/rk(myQueue) ] \\
BND-1002 : Delete

```

Subscription

There are two types of subscription, durable and non-durable. The durable subscription can be tested from the Java Client by creating a JMS Durable Topic Subscription. Additionally both types of subscription can have an additional argument for the JMS selector. The non-durable subscription type can also operate as a JMS Queue Browser which has an additional 'autoclose' argument.

This makes 5 different possible Create (SUB-1001) log messages.

```

2009-07-09 15:50:20 +0100 MESSAGE [ con:1(guest@127.0.0.1//)ch:2/sub:1:qu(myqueue) ] SUB-1001 :
Create : Durable [Arguments : <key=value>]
2009-07-09 15:50:20 +0100 MESSAGE [ con:1(guest@127.0.0.1//)ch:2/sub:1:qu(myqueue) ] SUB-1001 :
Create [: Arguments : <key=value>]
2009-07-09 15:50:20 +0100 MESSAGE [ con:1(guest@127.0.0.1//)ch:2/sub:1:qu(myqueue) ] SUB-1002 :
Close

```

Test Specification

The next step is to provide an enumerated list of tests for completion.

Each test in the list will include:

- Functional description of what is being tested.
- Input(actions and/or data)
- Expected outputs:
 - ... that will cause failure
 - ... that can safely be ignored.

Test Specification

Operational Logging - Status Update - Test Specification

Test Specification

- Overview
- Operational Test Cases
- Performance Test Case

Overview

The Test Specification will detail a 1:1 mapping from specification to test case.
Each test specification in the list will include:

- Functional description of what is being tested.
- Input(actions and/or data)
- Expected outputs:

- ... that will cause failure
- ... that can safely be ignored.

These details will then be used as the basis of each test that is created allowing for better maintainability in the test code.

Operational Test Cases

- Broker Test Suite
 - Broker Startup
 - testBrokerStartupConfiguration
 - testBrokerStartupCustomLog4j
 - testBrokerStartupDefaultLog4j
 - testBrokerStartupStartup
 - testBrokerStartupListeningTCPDefault
 - testBrokerStartupListeningTCPSSL
 - testBrokerStartupReady
 - Broker Shutdown
 - testBrokerShutdownListeningTCPDefault
 - testBrokerShutdownListeningTCPSSL
 - testBrokerShutdownStopped
- Management Console Test Suite
 - Management Startup
 - testManagementStartupEnabled
 - testManagementStartupDisabled
 - testManagementStartupRMIRegistry
 - testManagementStartupRMIRegistryCustom
 - testManagementStartupRMICConnectorServer
 - testManagementStartupRMICConnectorServerCustom
 - testManagementStartupSSLKeystore
 - testManagementStartupReady
 - Management Shutdown
 - testManagementShutdownRMIRegistry
 - testManagementShutdownRMICConnectorServer
 - testManagementShutdownStopped
- Virtualhost Test Cases
 - testVirtualhostCreation
 - testVirtualhostClosure
- MessageStore Tests
 - testMessageStoreCreation
 - testMessageStoreStoreLocation
 - testMessageStoreClose
 - testMessageStoreRecoveryStart
 - testMessageStoreQueueRecoveryShowRecovered
 - testMessageStoreQueueRecoveryCountEmpty
 - testMessageStoreQueueRecoveryCountPlural
 - testMessageStoreQueueRecoveryCountSingular
 - testMessageStoreQueueRecoveryComplete
 - testMessageStoreRecoveryComplete
- Connection Test Suite
 - testConnectionOpen
 - testConnectionClose
 - testConnectionCloseViaManagement
- Channel
 - testChannelCreate
 - testChannelConsumerFlowStopped
 - testChannelConsumerFlowStarted
 - testChannelCloseViaConnectionClose
 - testChannelCloseViaChannelClose
 - testChannelCloseViaError
- Queue
 - testQueueCreatePersistent
 - testQueueCreatePersistentAutoDelete
 - testCreateQueuePersistentPriority
 - testCreateQueuePersistentAutoDeletePriority
 - testQueueCreateTransient
 - testQueueCreateTransientAutoDelete
 - testCreateQueueTransientPriority
 - testCreateQueueTransientAutoDeletePriority
 - testCreateQueueTransientViaManagementConsole
 - testQueueDelete
 - testQueueAutoDelete
 - testQueueDeleteViaManagementConsole
- Exchange
 - testExchangeCreateDurable
 - testExchangeCreate
 - testExchangeDelete
- Binding
 - testBindingCreate
 - testBindingCreateWithArguments
 - testBindingCreateViaManagementConsole

- testBindingDelete
- testBindingDeleteViaManagementConsole
- Subscription
 - testSubscriptionCreate
 - testSubscriptionCreateDurable
 - testSubscriptionCreateWithArguments
 - testSubscriptionCreateDurableWithArguments
 - testSubscriptionCreateQueueBrowser
 - testSubscriptionClose
- Test Structure
- Risks

This section enumerates the various operational tests described in the [Test Plan](#) identified from the [Functional Specification](#). This text should form the basis of the Technical Documentation for the specified test class.

Broker Test Suite

The Broker test suite validates that the follow log messages as specified in the [Functional Specification](#).

```
BRK-1001 : Startup : Version: <Version> Build: <Build>
BRK-1002 : Starting : Listening on <Transport> port <Port>
BRK-1003 : Shuting down : <Transport> port <Port>
BRK-1004 : Ready
BRK-1005 : Stopped
BRK-1006 : Using configuration : <path>
BRK-1007 : Using logging configuration : <path>
```

These messages should only occur during startup. The tests need to verify the order of messages. In the case of the BRK-1002 and BRK-1003 the respective ports should only be available between the two log messages.

Broker Startup testBrokerStartupConfiguration

Description: On startup the broker must report the active configuration file. The logging system must output this so that we can know what configuration is being used for this broker instance.

Input:

The value of -c specified on the command line.

Output:

```
<date> MESSAGE BRK-1006 : Using configuration : <config file>
```

Constraints:

This **MUST BE** the first *BRK* log message.

Validation Steps:

1. This is first *BRK* log message.
2. The *BRKID* is correct
3. The config file is the full path to the file specified on the commandline.

testBrokerStartupCustomLog4j

Description:

On startup the broker must report correctly report the log4j file in use. This is important as it can help diagnose why logging messages are not being reported. The broker must also be capable of correctly recognising the command line property to specify the custom logging configuration.

Input:

The value of -l specified on the command line.

Output:

```
<date> MESSAGE BRK-1007 : Using logging configuration : <log4j file>
```

Validation Steps:

1. The *BRKID* is correct
2. This should occur before the BRK-1001 : Startup message
3. The log4j file is the full path to the file specified on the commandline.

testBrokerStartupDefaultLog4j

Description:

On startup the broker must report correctly report the log4j file in use. This is important as it can help diagnose why logging messages are not being reported.

Input:

No custom -l value should be provided on the command line so that the default value is correctly reported.

Output:

```
<date> MESSAGE BRK-1007 : Using logging configuration : <$QPID_HOME>/etc/log4j.xml
```

Validation Steps:

1. The *BRK ID* is correct
2. This occurs before the *BRK-1001* startup message.
3. The log4j file is the full path to the file specified on the commandline.

testBrokerStartupStartup

Description: On startup the broker reports the broker version number and svn build revision. This information is retrieved from the resource 'qpidversion.properties' which is located via the classloader.

Input: The 'qpidversion.properties' file located on the classpath.

Output:

```
<date> MESSAGE BRK-1001 : Startup : qpid Version: 0.6 Build: 767150
```

Validation Steps:

1. The *BRK ID* is correct
2. This occurs before any *BRK-1002* listening messages are reported.

testBrokerStartupListeningTCPDefault

Description:

On startup the broker may listen on a number of ports and protocols. Each of these must be reported as they are made available.

Input:

The default configuration with no SSL

Output:

```
<date> MESSAGE BRK-1002 : Starting : Listening on TCP port 5672
```

Constraints:

Additional broker configuration will occur between the Startup(BRK-1001) and Starting(BRK-1002) messages depending on what VirtualHosts are configured.

Validation Steps:

1. The *BRK ID* is correct
2. This occurs after the *BRK-1001* startup message
3. Using the default configuration a single *BRK-1002* will be printed showing values TCP / 5672

testBrokerStartupListeningTCPSSL

Description:

On startup the broker may listen on a number of ports and protocols. Each of these must be reported as they are made available.

Input:

The default configuration with SSL enabled

Output:

```
<date> MESSAGE BRK-1002 : Starting : Listening on TCP port 5672
<date> MESSAGE BRK-1002 : Starting : Listening on TCP/SSL port 8672
```

Constraints:

Additional broker configuration will occur between the Startup(BRK-1001) and Starting(BRK-1002) messages depending on what VirtualHosts are configured.

Validation Steps:

1. The *BRK ID* is correct
2. This occurs after the *BRK-1001* startup message
3. With SSL enabled in the configuration two *BRK-1002* will be printed (order is not specified)
 - a. One showing values TCP / 5672
 - b. One showing values TCP/SSL / 8672

testBrokerStartupReady

Description:

The final message the broker will print when it has performed all initialisation and listener startups will be to log the *BRK-1004* Ready message

Input:

No input, all successful broker startups will show *BRK-1004* messages.

Output:

```
2009-07-09 15:50:20 +0100 MESSAGE BRK-1004 : Ready
```

Validation Steps:

1. The *BRKID* is correct
2. This occurs after the *BRK-1001* startup message
3. This must be the last message the broker prints after startup. Currently, if there is no further interaction with the broker then there should be no more logging.

Broker Shutdown testBrokerShutdownListeningTCPDefault

Description:

On startup the broker may listen on a number of ports and protocols. Each of these must then report a shutting down message as they stop listening.

Input:

The default configuration with no SSL

Output:

```
<date> MESSAGE BRK-1003 : Shutting down : TCP port 5672
```

Validation Steps:

1. The *BRKID* is correct
2. Only TCP is reported with the default configuration with no SSL.
3. The default port is correct
4. The port is not accessible after this message

testBrokerShutdownListeningTCPSSL

Description:

On startup the broker may listen on a number of ports and protocols. Each of these must then report a shutting down message as they stop listening.

Input:

The default configuration with SSL enabled

Output:

```
<date> MESSAGE BRK-1003 : Shutting down : TCP port 5672  
<date> MESSAGE BRK-1003 : Shutting down : TCP/SSL port 8672
```

Validation Steps:

1. The *BRKID* is correct
2. With SSL enabled in the configuration two *BRK-1003* will be printed (order is not specified)
3. The default port is correct
4. The port is not accessible after this message

testBrokerShutdownStopped

Description:**Input:**

No input, all clean broker shutdowns will show *BRK-1005* messages.

Output:

```
<date> MESSAGE BRK-1005 : Stopped
```

Constraints:

This is the **LAST** message the broker will log.

Validation Steps:

1. The *BRKID* is correct
2. This is the last message the broker will log.

Management Console Test Suite

The Management Console test suite validates that the follow log messages as specified in the [Functional Specification](#).

This suite of tests validate that the management console messages occur correctly and according to the following format:

```
MNG-1001 : Startup  
MNG-1002 : Starting : <service> : Listening on port <Port>  
MNG-1003 : Shutting down : <service> : port <Port>  
MNG-1004 : Ready  
MNG-1005 : Stopped  
MNG-1006 : Using SSL Keystore : <path>
```

Management Startup testManagementStartupEnabled

Description:

Using the startup configuration validate that the management startup message is logged correctly.

Input:

Standard configuration with management enabled

Output:

```
<date> MNG-1001 : Startup
```

Constraints:

This is the **FIRST** message logged by *MNG*

Validation Steps:

1. The *BRK ID* is correct
2. This is the **FIRST** message logged by *MNG*

testManagementStartupDisabled

Description:

Verify that when management is disabled in the configuration file the startup message is not logged.

Input:

Standard configuration with management disabled

Output:

NO *MNG* messages

Validation Steps:

1. Validate that no *MNG* messages are produced.

testManagementStartupRMIRRegistry

Description:

Using the default configuration validate that the RMI Registry socket is correctly reported as being opened

Input:

The default configuration file

Output:

```
<date> MESSAGE MNG-1002 : Starting : RMI Registry : Listening on port 8999
```

Constraints:

The RMI ConnectorServer and Registry log messages do not have a prescribed order

Validation Steps:

1. The *MNG ID* is correct
2. The specified port is the correct '8999'

testManagementStartupRMIRRegistryCustom

Description:

Using the default configuration validate that the RMI Registry socket is correctly reported when overridden via the command line.

Input:

The default configuration file and a custom -m value

Output:

```
<date> MESSAGE MNG-1002 : Starting : RMI Registry : Listening on port <port>
```

Constraints:

The RMI ConnectorServer and Registry log messages do not have a prescribed order

Validation Steps:

1. The *MNG ID* is correct
2. The specified port is as specified on the command line.

testManagementStartupRMICConnectorServer

Description:

Using the default configuration validate that the RMI ConnectorServer socket is correctly reported as being opened

Input:

The default configuration file

Output:

```
<date> MESSAGE MNG-1002 : Starting : RMI ConnectorServer : Listening on port 9099
```

Constraints:

The RMI ConnectorServer and Registry log messages do not have a prescribed order

Validation Steps:

1. The MNG ID is correct
2. The specified port is the correct '9099'

testManagementStartupRMICConnectorServerCustom

Description:

Using the default configuration validate that the RMI Registry socket is correctly reported when overridden via the command line.

Input:

The default configuration file and a custom -m value

Output:

```
<date> MESSAGE MNG-1002 : Starting : RMI ConnectorServer : Listening on port <port>
```

Constraints:

The RMI ConnectorServer and Registry log messages do not have a prescribed order

Validation Steps:

1. The MNG ID is correct
2. The specified port is as specified on the commandline.

testManagementStartupSSLKeystore

Description:

Using the default configuration with SSL enabled for the management port the SSL Keystore path should be reported via *MNG-1006*

Input:

Management SSL enabled default configuration.

Output:

```
<date> MESSAGE MNG-1006 : Using SSL Keystore : test_resources/ssl/keystore.jks
```

Validation Steps:

1. The MNG ID is correct
2. The keystore path is as specified in the configuration

testManagementStartupReady

Description:

Using the default configuration the final stage of management startup is to report a *MNG-1004* Ready message.

Input:

Default broker configuration.

Output:

```
<date> MESSAGE MNG-1004 : Ready
```

Validation Steps:

1. The MNG ID is correct
2. There has been a *MNG-1001* message
3. There has been at least one *MNG-1002* Listening message
4. No further MNG messages are produced as part of the startup process, i.e. before broker use.

Management Shutdown testManagementShutdownRMIRRegistry

Description:

Using the default configuration the management RMI Registry will start and so on shutdown it will log that it is shutting down.

Input:

The default configuration file.

Output:

```
<date> MNG-1003 : Shutting down : RMI Registry : Listening on port 8999
```

Validation Steps:

1. The MNG ID is correct
2. The *MNG-1004* message has been logged.

testManagementShutdownRMICConnectorServer

Description:

Using the default configuration the management RMI ConnectorServer will start and so on shutdown it will log that it is shutting down.

Input:

The default configuration file.

Output:

```
<date> MNG-1003 : Shutting down : RMI ConnectorServer : Listening on port 9099
```

Validation Steps:

1. The *MNG* ID is correct
2. The *MNG-1004* message has been logged.

testManagementShutdownStopped

Description:

On final shutdown the management console will report that it has stopped. All *MNG* logging must be complete before this message is logged.

Input:

The default configuration file.

Output:

```
<date> MNG-1005 : Stopped
```

Validation Steps:

1. The *MNG* ID is correct
2. The *MNG-1004* message has been logged.
3. For each *MNG-1002* message that was logged a *MNG-1003* is also logged before this message.
4. This is the last *MNG* message reported.

Virtualhost Test Cases

The virtualhost test suite validates that the follow log messages as specified in the [Functional Specification](#).

This suite of tests validate that the management console messages occur correctly and according to the following format:

```
VHT-1001 : Created : <name>
VHT-1002 : Work directory : <path>
VHT-1003 : Closed
```

testVirtualhostCreation

Description:

Testing can be performed using the default configuration. The goal is to validate that for each virtualhost defined in the configuration file a *VHT-1001* *Created* message is provided.

Input:

The default configuration file

Output:

```
<date> VHT-1001 : Created : <name>
```

Validation Steps:

1. The *VHTID* is correct
2. A *VHT-1001* is printed for each virtualhost defined in the configuration file.
3. This must be the **first** message for the specified virtualhost.

testVirtualhostClosure

Description:

Testing can be performed using the default configuration. During broker shutdown a *VHT-1002* *Closed* message will be printed for each of the configured virtualhosts. For every virtualhost that was started a close must be logged. After the close message has been printed no further logging will be performed by this virtualhost.

Input:

The default configuration file

Output:

```
<date> VHT-1002 : Closed
```

Validation Steps:

1. The *VHTID* is correct
2. This is the last *VHT* message for the given virtualhost.

MessageStore Tests

The MessageStore test suite validates that the follow log messages as specified in the [Functional Specification](#).

This suite of tests validate that the MessageStore messages occur correctly and according to the following format:

```
MST-1001 : Created : <name>
MST-1002 : Store location : <path>
MST-1003 : Closed
MST-1004 : Recovery Start [: <queue.name>]
MST-1005 : Recovered <count> messages for queue <queue.name>
MST-1006 : Recovery Complete [: <queue.name>]
```

testMessageStoreCreation

Description:

During Virtualhost startup a MessageStore will be created. The first *MST* message that must be logged is the *MST-1001* MessageStore creation.

Input:

Default configuration

Output:

```
<date> MST-1001 : Created : <name>
```

Validation Steps:

1. The *MST ID* is correct
2. The <name> is the correct MessageStore type as specified in the Default configuration

testMessageStoreStoreLocation

Description:

Persistent MessageStores will require space on disk to persist the data. This value will be logged on startup after the MessageStore has been created.

Input:

Default configuration

Output:

```
<date> MST-1002 : Store location : <path>
```

Validation Steps:

1. The *MST ID* is correct
2. This must occur after *MST-1001*

testMessageStoreClose

Description:

During shutdown the MessageStore will also cleanly close. When this has completed a *MST-1003* closed message will be logged. No further messages from this MessageStore will be logged after this message

Input:

Default configuration

Output:

```
<date> MST-1003 : Closed
```

Validation Steps:

1. The *MST ID* is correct
2. This is the **last** log message from this MessageStore

testMessageStoreRecoveryStart

Description:

Persistent message stores may have state on disk that they must recover during startup. As the MessageStore starts up it will report that it is about to start the recovery process by logging *MST-1004*. This message will always be logged for persistent MessageStores. If there is no data to recover then there will be no subsequent recovery messages.

Input:

Default persistent configuration

Output:

```
<date> MST-1004 : Recovery Start
```

Validation Steps:

1. The *MST ID* is correct
2. The MessageStore must have first logged a creation event.

testMessageStoreQueueRecoveryShowRecovered

Description:

A persistent MessageStore may have data to recover from disk. The message store will use *MST-1004* to report the start of recovery for a specific queue that it has previously persisted.

Input:

Default persistent configuration

Output:

```
<date> MST-1004 : Recovery Start : <queue.name>
```

Validation Steps:

1. The *MST ID* is correct
2. This must occur after the recovery start *MST-1004* has been logged.

testMessageStoreQueueRecoveryCountEmpty

Description:

A persistent queue must be persisted so that on recovery it can be restored independently of any messages that may be stored on it. This test verifies that the MessageStore will log that it has recovered 0 messages for persistent queues that do not have any messages.

Input:

1. Default persistent configuration
2. Persistent queue with no messages enqueued

Output:

```
<date> MST-1005 : Recovered 0 messages for queue <queue.name>
```

Validation Steps:

3. The *MST ID* is correct
4. This must occur after the queue recovery start *MST-1004* has been logged.
5. The count is 0
6. 'messages' is correctly printed
7. The queue.name is non-empty

testMessageStoreQueueRecoveryCountPlural

Description:

On recovery all the persistent messages that are stored on disk must be returned to the queue. *MST-1005* will report the number of messages that have been recovered from disk.

Input:

1. Default persistent configuration
2. Persistent queue with multiple messages enqueued

Output:

```
<date> MST-1005 : Recovered <count> messages for queue <queue.name>
```

Validation Steps:

3. The *MST ID* is correct
4. This must occur after the queue recovery start *MST-1004* has been logged.
5. The count is > 1
6. 'messages' is correctly printed
7. The queue.name is non-empty

testMessageStoreQueueRecoveryCountSingular

Description:

On recovery all the persistent messages that are stored on disk must be returned to the queue. *MST-1005* will report the number of messages that have been recovered from disk.

Input:

1. Default persistent configuration
2. A persistent queue with a single message enqueued.

Output:

```
<date> MST-1005 : Recovered 1 message for queue <queue.name>
```

Validation Steps:

3. The *MST ID* is correct
4. This must occur after the queue recovery start *MST-1004* has been logged.
5. The count is 1

6. 'message' is correctly printed
7. The queue.name is non-empty

testMessageStoreQueueRecoveryComplete

Description:

After the queue has been recovered the store will log that recovery has been completed. The MessageStore must not report further status about the recovery of this queue after this message. In addition every *MST-1004* queue recovery start message must be matched with a *MST-1006* recovery complete.

Input:

Default persistent configuration

Output:

```
<date> MST-1006 : Recovery Complete : <queue.name>
```

Validation Steps:

1. The *MST ID* is correct
2. This must occur after the queue recovery start *MST-1004* has been logged.
3. The queue.name is non-empty
4. The queue.name correlates with a previous recovery start

testMessageStoreRecoveryComplete

Description:

Once all persistent queues have been recovered and the MessageStore has completed all recovery it must log that the recovery process has completed.

Input:

Default persistent configuration

Output:

```
<date> MST-1006 : Recovery Complete
```

Validation Steps:

1. The *MST ID* is correct
2. This is the **last** message from the MessageStore during startup.
3. This must be proceeded by a *MST-1004* Recovery Start.

Connection Test Suite

The Connection test suite validates that the follow log messages as specified in the Functional Specification.

This suite of tests validate that the Connection messages occur correctly and according to the following format:

```
CON-1001 : Open : Client ID <id> : Protocol Version : <version>
CON-1002 : Close
```

testConnectionOpen

Description:

When a new connection is made to the broker this must be logged.

Input:

1. Running Broker
2. Connecting client

Output:

```
<date> CON-1001 : Open : Client ID <id> : Protocol Version : <version>
```

Validation Steps:

3. The CON ID is correct
4. This is the **first** CON message for that Connection

testConnectionClose

Description:

When a connected client closes the connection this will be logged as a CON-1002 message.

Input:

1. Running Broker
2. Connected Client

Output:

```
<date> CON-1002 : Close
```

Validation Steps:

3. The CON ID is correct
4. This must be the last CON message for the Connection
5. It must be preceded by a CON-1001 for this Connection

testConnectionCloseViaManagement

Description:

When a connected client has its connection closed via the Management Console this will be logged as a CON-1002 message.

Input:

1. Running Broker
2. Connected Client
3. Connection is closed via Management Console

Output:

```
<date> CON-1002 : Close
```

Validation Steps:

4. The CON ID is correct
5. This must be the last CON message for the Connection
6. It must be preceded by a CON-1001 for this Connection

Channel

The Channel test suite validates that the follow log messages as specified in the Functional Specification.

This suite of tests validate that the Channel messages occur correctly and according to the following format:

```
CHN-1001 : Create : Prefetch <count>
CHN-1002 : Flow <value>
CHN-1003 : Close
```

testChannelCreate

Description:

When a new Channel (JMS Session) is created this will be logged as a CHN-1001 Create message. The messages will contain the prefetch details about this new Channel.

Input:

1. Running Broker
2. New JMS Session/Channel creation

Output:

```
<date> CHN-1001 : Create : Prefetch <count>
```

Validation Steps:

3. The CHN ID is correct
4. The prefetch value matches that defined by the requesting client.

testChannelConsumerFlowStopped

Description:

The Java Broker implements consumer flow control for all ack modes except No-Ack. When the client fills the prefetch then a CHN-1002 Flow Stopped messasge will be issued in the log.

Input:

1. Running broker
2. Message Producer to put more data on the queue than the client's prefetch
3. Client that ensures that its prefetch becomes full

Output:

```
<date> CHN-1002 : Flow Stopped
```

Validation Steps:

4. The CHN ID is correct

testChannelConsumerFlowStarted

Description:

The Java Broker implements consumer flow control for all ack modes except No-Ack. When the client fills the prefetch. As soon as the client starts to consume the messages (and ack them) the broker will resume the flow issuing a *CHN-1002 Flow Started* message to the log

Input:

1. Running broker
2. Message Producer to put more data on the queue than the client's prefetch
3. Client that ensures that its prefetch becomes full
4. The client then consumes from the prefetch to remove the flow status.

Output:

```
<date> CHN-1002 : Flow Started
```

Validation Steps:

5. The *MST ID* is correct

testChannelCloseViaConnectionClose

Description:

When the client gracefully closes the Connection then a *CHN-1003 Close* message will be issued. This must be the last message logged for this Channel.

Input:

1. Running Broker
2. Connected Client
3. Client then requests that the Connection is closed

Output:

```
<date> CHN-1003 : Close
```

Validation Steps:

4. The *MST ID* is correct
5. This must be the last message logged for this Channel.

testChannelCloseViaChannelClose

Description:

When the client requests that the Channel (JMS Session) be closed then a *CHN-1003 Close* message will be issued. This must be the last message logged for this Channel.

Input:

1. Running Broker
2. Connected Client
3. Client then requests that the Channel is closed

Output:

```
<date> CHN-1003 : Close
```

Validation Steps:

4. The *MST ID* is correct
5. This must be the last message logged for this Channel.

testChannelCloseViaError

Description:

If a Connection becomes interrupted and then a *CHN-1003 Close* message will still be issued to signify that the Channel has been closed. This must be the last message logged for this Channel.

Input:

1. Running Broker
2. Connected Client
3. Client then requests that the Channel is closed

Output:

```
<date> CHN-1003 : Close
```

Validation Steps:

4. The *MST ID* is correct
5. This must be the last message logged for this Channel.

Queue

The Queue test suite validates that the follow log messages as specified in the [Functional Specification](#).

This suite of tests validate that the Queue messages occur correctly and according to the following format:

```
QUE-1001 : Create : [AutoDelete] [Durable|Transient] [Priority:<levels>] [Owner:<name>]
QUE-1002 : Deleted
```

testQueueCreatePersistent

Description:

When a simple persistent queue is created then a *QUE-1001* create message is expected to be logged.

Input:

1. Running broker
2. Persistent Queue is created from a client

Output:

```
<date> QUE-1001 : Create : Persistent Owner:<name>
```

Validation Steps:

3. The *QUE* ID is correct
4. The Persistent tag is present in the message
5. The Owner is as expected

testQueueCreatePersistentAutoDelete

Description:

When an autodelete persistent queue is created then a *QUE-1001* create message is expected to be logged.

Input:

1. Running broker
2. AutoDelete Persistent Queue is created from a client

Output:

```
<date> QUE-1001 : Create : AutoDelete Persistent Owner:<name>
```

Validation Steps:

3. The *QUE* ID is correct
4. The Persistent tag is present in the message
5. The Owner is as expected
6. The AutoDelete tag is present in the message

testCreateQueuePersistentPriority

Description:

When a persistent queue is created with a priority level then a *QUE-1001* create message is expected to be logged.

Input:

1. Running broker
2. Persistent Queue is created from a client with a priority level

Output:

```
<date> QUE-1001 : Create : Persistent Priority:<levels> Owner:<name>
```

Validation Steps:

3. The *QUE* ID is correct
4. The Persistent tag is present in the message
5. The Owner is as expected
6. The Priority level is correctly set

testCreateQueuePersistentAutoDeletePriority

Description:

When an autodelete persistent queue is created with a priority level then a *QUE-1001* create message is expected to be logged.

Input:

1. Running broker
2. An AutoDelete Persistent Queue is created from a client with priority

Output:

```
<date> QUE-1001 : Create : AutoDelete Persistent Priority:<levels> Owner:<name>
```

Validation Steps:

3. The *QUE* ID is correct
4. The Persistent tag is present in the message
5. The Owner is as expected
6. The AutoDelete tag is present in the message
7. The Priority level is correctly set

testQueueCreateTransient

Description:

When a simple transient queue is created then a *QUE-1001* create message is expected to be logged.

Input:

1. Running broker
2. Transient Queue is created from a client

Output:

```
<date> QUE-1001 : Create : Transient Owner:<name>
```

Validation Steps:

3. The *QUE* ID is correct
4. The Transient tag is present in the message
5. The Owner is as expected

testQueueCreateTransientAutoDelete

Description:

When an autodelete transient queue is created then a *QUE-1001* create message is expected to be logged.

Input:

1. Running broker
2. AutoDelete Transient Queue is created from a client

Output:

```
<date> QUE-1001 : Create : AutoDelete Transient Owner:<name>
```

Validation Steps:

3. The *QUE* ID is correct
4. The Transient tag is present in the message
5. The Owner is as expected
6. The AutoDelete tag is present in the message

testCreateQueueTransientPriority

Description:

When a transient queue is created with a priority level then a *QUE-1001* create message is expected to be logged.

Input:

1. Running broker
2. Transient Queue is created from a client with a priority level

Output:

```
<date> QUE-1001 : Create : Transient Priority:<levels> Owner:<name>
```

Validation Steps:

3. The *QUE* ID is correct
4. The Transient tag is present in the message
5. The Owner is as expected
6. The Priority level is correctly set

testCreateQueueTransientAutoDeletePriority

Description:

When an autodelete transient queue is created with a priority level then a *QUE-1001* create message is expected to be logged.

Input:

1. Running broker
2. An autodelete Transient Queue is created from a client with a priority level

Output:

```
<date> QUE-1001 : Create : AutoDelete Transient Priority:<levels> Owner:<name>
```

Validation Steps:

3. The *QUE* ID is correct
4. The Transient tag is present in the message
5. The Owner is as expected
6. The AutoDelete tag is present in the message
7. The Priority level is correctly set

testCreateQueueTransientViaManagementConsole

Description:

Queue creation is possible from the Management Console. When a queue is created in this way then a *QUE-1001* create message is expected to be logged.

Input:

1. Running broker
2. Connected Management Console
3. Queue Created via Management Console

Output:

```
<date> QUE-1001 : Create : Transient Owner:<name>
```

Validation Steps:

4. The *QUE* ID is correct
5. The correct tags are present in the message based on the create options

testQueueDelete

Description:

An explicit QueueDelete request must result in a *QUE-1002* Deleted message being logged. This can be done via an explicit AMQP QueueDelete method.

Input:

1. Running Broker
2. Queue created on the broker with no subscribers
3. Client requests the queue be deleted via a QueueDelete

Output:

```
<date> QUE-1002 : Deleted
```

Validation Steps:

4. The *QUE* ID is correct

testQueueAutoDelete

Description:

When a Client requests a temporary queue then this is represented in the Java Broker as an autodelete exclusive queue. When the client disconnects the queue will automatically deleted. This can be seen as a *QUE-1002* Deleted message will be logged.

Input:

1. Running Broker
2. Client creates a temporary queue then disconnects

Output:

```
<date> QUE-1002 : Deleted
```

Validation Steps:

3. The *QUE* ID is correct

testQueueDeleteViaManagementConsole

Description:

The ManagementConsole can be used to delete a queue. When this is done a *QUE-1002* Deleted message must be logged.

Input:

1. Running Broker
2. Queue created on the broker with no subscribers
3. Management Console connected
4. Queue is deleted via Management Console

Output:

```
<date> QUE-1002 : Deleted
```

Validation Steps:

5. The *QUE* ID is correct

Exchange

The Exchange test suite validates that the follow log messages as specified in the [Functional Specification](#).

This suite of tests validate that the Exchange messages occur correctly and according to the following format:

```
EXH-1001 : Create : [Durable] Type:<value> Name:<value>
EXH-1002 : Deleted
```

testExchangeCreateDurable

Description:

When a durable exchange is created an *EXH-1001* message is logged with the Durable tag. This will be the first message from this exchange.

Input:

1. Running broker
2. Client requests a durable exchange be created.

Output:

```
<date> EXH-1001 : Create : Durable Type:<value> Name:<value>
```

Validation Steps:

3. The *EXH* ID is correct
4. The Durable tag is present in the message

testExchangeCreate

Description:

When an exchange is created an *EXH-1001* message is logged. This will be the first message from this exchange.

Input:

1. Running broker
2. Client requests an exchange be created.

Output:

```
<date> EXH-1001 : Create : Type:<value> Name:<value>
```

Validation Steps:

3. The *EXH* ID is correct

testExchangeDelete

Description:

An Exchange can be deleted through an AMQP ExchangeDelete method. When this is successful an *EXH-1002* Delete message will be logged. This will be the last message from this exchange.

Input:

1. Running broker
2. A new Exchange has been created
3. Client requests that the new exchange be deleted.

Output:

```
<date> EXH-1002 : Deleted
```

Validation Steps:

4. The *EXH* ID is correct
5. There is a corresponding *EXH-1001* Create message logged.

Binding

The Binding test suite validates that the follow log messages as specified in the [Functional Specification](#).

This suite of tests validate that the Binding messages occur correctly and according to the following format:

```
BND-1001 : Create [: Arguments : <key=value>]  
BND-1002 : Deleted
```

testBindingCreate

Description:

The binding of a Queue and an Exchange is done via a Binding. When this Binding is created a *BND-1001* Create message will be logged.

Input:

1. Running Broker
2. New Client requests that a Queue is bound to a new exchange.

Output:

```
<date> BND-1001 : Create
```

Validation Steps:

3. The *BND* ID is correct
4. This will be the **first** message for the given binding

testBindingCreateWithArguments

Description:

A Binding can be made with a set of arguments. When this occurs we log the key,value pairs as part of the Binding log message. When the subscriber with a JMS Selector consumes from an exclusive queue such as a topic. The binding is made with the JMS Selector as an argument.

Input:

1. Running Broker
2. Java Client consumes from a topic with a JMS selector.

Output:

```
<date> BND-1001 : Create : Arguments : <key=value>
```

Validation Steps:

3. The *BND* ID is correct
4. The JMS Selector argument is present in the message
5. This will be the **first** message for the given binding

testBindingCreateViaManagementConsole

Description:

The binding of a Queue and an Exchange is done via a Binding. When this Binding is created via the Management Console a *BND-1001* Create message will be logged.

Input:

1. Running Broker
2. Connected Management Console
3. Use Management Console to perform binding

Output:

```
<date> BND-1001 : Create
```

Validation Steps:

4. The *BND* ID is correct
5. This will be the **first** message for the given binding

testBindingDelete

Description:

Bindings can be deleted so that a queue can be rebound with a different set of values.

Input:

1. Running Broker
2. AMQP UnBind Request is made

Output:

```
<date> BND-1002 : Deleted
```

Validation Steps:

3. The *BND* ID is correct
4. There must have been a *BND-1001* Create message first.
5. This will be the **last** message for the given binding

testBindingDeleteViaManagementConsole

Description:

Bindings can be deleted so that a queue can be rebound with a different set of values. This can be performed via the Management Console

Input:

1. Running Broker
2. Management Console connected
3. Management Console is used to perform unbind.

Output:

```
<date> BND-1002 : Deleted
```

Validation Steps:

4. The *BND* ID is correct
5. There must have been a *BND-1001* Create message first.
6. This will be the **last** message for the given binding

Subscription

The Subscription test suite validates that the follow log messages as specified in the [Functional Specification](#).

This suite of tests validate that the Subscription messages occur correctly and according to the following format:

```
SUB-1001 : Create : [Durable] [Arguments : <key=value>]
SUB-1002 : Close
```

testSubscriptionCreate

Description:

When a Subscription is created it will be logged. This test validates that Subscribing to a transient queue is correctly logged.

Input:

1. Running Broker
2. Create a new Subscription to a transient queue/topic.

Output:

```
<date> SUB-1001 : Create
```

Validation Steps:

3. The *SUB* ID is correct

testSubscriptionCreateDurable

Description:

The creation of a Durable Subscription, such as a JMS DurableTopicSubscriber will result in an extra Durable tag being included in the Create log message

Input:

1. Running Broker
2. Creation of a JMS DurableTopicSubscriber

Output:

```
<date> SUB-1001 : Create : Durable
```

Validation Steps:

3. The *SUB* ID is correct
4. The Durable tag is present in the message

testSubscriptionCreateWithArguments

Description:

The creation of a Subscriber with a JMS Selector will result in the Argument field being populated. These argument key/value pairs are then shown in the log message.

Input:

1. Running Broker
2. Subscriber created with a JMS Selector.

Output:

```
<date> SUB-1001 : Create : Arguments : <key=value>
```

Validation Steps:

3. The *SUB ID* is correct
4. Argument tag is present in the message

testSubscriptionCreateDurableWithArguments

Description:

The final combination of *SUB-1001* Create messages involves the creation of a Durable Subscription that also contains a set of Arguments, such as those provided via a JMS Selector.

Input:

1. Running Broker
2. Java Client creates a Durable Subscription with Selector

Output:

```
<date> SUB-1001 : Create : Durable Arguments : <key=value>
```

Validation Steps:

3. The *SUB ID* is correct
4. The tag *Durable* is present in the message
5. The Arguments are present in the message

testSubscriptionCreateQueueBrowser

Description:

The creation of a QueueBrowser will provides a number arguments and so should form part of the *SUB-1001* Create message.

Input:

1. Running Broker
2. Java Client creates a QueueBroweser

Output:

```
<date> SUB-1001 : Create : Arguments : <key=value>
```

Validation Steps:

3. The *SUB ID* is correct
4. The Arguments are present in the message
5. Arguments keys include AutoClose and Browser.

testSubscriptionClose

Description:

When a Subscription is closed it will log this so that it can be correlated with the Create.

Input:

1. Running Broker
2. Client with a subscription.
3. The subscription is then closed.

Output:

```
<date> SUB-1002 : Close
```

Validation Steps:

4. The *SUB ID* is correct
5. There must be a *SUB-1001* Create message preceding this message
6. This must be the **last** message from the given Subscription

Performance Test Case

In addition to the performance test suite an additional performance test needs to be written that can be run with this new logging enabled and disabled so that an attempt at quantifying any impact can be made.

Test Structure

The test should perform the following actions:

1. Connect a client
2. Create a channel/JMS Session
3. Create an exchange
4. Create a queue

5. Bind the exchange and queue
6. Create a subscriber on the queue
7. Close the Subscriber
8. Close the Session
9. Close the Connection

This will ensure that we hit as many of the new logging routines as possible.
If this test should also be run prior to any code changes so that our current performance can be recorded.

Risks

Testing of this nature is dependant on a lot of items that are out of the tests control such as:

1. CPU scheduling
2. CPU performance
3. Load
4. GC

As a result the test cannot be guaranteed to produce the same results each time. To mitigate this risk running the test in a loop an reporting an average value of 10-20 runs should provide a more stable response.
Leaving the broker startup/shutdown out of the test loop will help improve the tests performance and repeatability.

Status Update Design - Logging Configuration

Overview

To better address this and to allow us to more easily move between logging implementations an abstraction is recommended. This approach will allow the simplification of the logic around determining if logging should be performed in the main code base. For example a Binding logging statement can be controlled via Exchange and Queue, each of which may be limited to a specific instance.

Logging Control

Initially the use of a properties file will be used to enable/disable various logging options. The property value will be one of a set list. This will help avoid the ambiguous parsing of Java Properties files.

Control File Structure

```
# Comment Line
<logger> = <value [OFF|TRACE|DEBUG|INFO|WARN|ERROR|FATAL|ALL]>
```

The values for the <logger> are based on navigation through the hierarchy.

Hierarchy Refinement

Refining the hierarchy presented in the [high level design](#) such that we can implement it in terms of an easily parseable structure for use as the <logger> values.

```
qpidd broker

    [<username>]
    [<ip/hostname>]
connection [<id>] channel [<id>] subscription [<id>]

virtualHost <name>           binding

                    exchange <name>
                    queue  <name>
                    plugin <name>
                    messagestore
```

This hierarchy has a number of paths which make it more difficult to process.

1. Connection can be optionally followed with up to one of the following: username, ip/host, id
2. Subscription can be reached through Connection and Virtualhost
3. Exchange and Queue can be added in either order
4. Binding can be routed through Queue and Exchange, or both in either order.

It is these difficulties that make it best to provide an abstraction layer so that a simple interface can be used at the site of the logging. These difficulties can be distilled to:

1. Processing values for all values of '<name>'
2. Overlapping configuration resolution

3. Presentation of entities with multiple paths

Processing values for all values of '<name>'

To ensure that the logger path is processable the '<name>' must be present to make it easier to understand when we have an identifier or a <name> value. i.e. Is this logging all events for exchanges on Virtualhost called 'queue' or logging all exchange events that occur in relation to queues.

```
qpid.virtualhost.queue.exchange = INFO
```

By introducing the use of the '*' wild card we can make these two situations easier to read:

```
qpid.virtualhost.queue.exchange = INFO  
qpid.virtualhost.*.queue.exchange = INFO
```

The '*' can then be used at the end of the logger to ensure all logging from this point in the hierarchy will be logged, rather than just events to the specified entity.

Overlapping configuration resolution

The loops in the graph will be handled by the logger configurator so that only one log entry is created for each event, even if there are multiple matching entries in the configuration. For example, the follow two entries are equivalent and will both enable all logging for bindings. If the user wishes to log all binding operations then only one entry is necessary but the presence of both should not cause duplicate messages.

Multiple equivalent entries doesn't result in multiple logging

```
qpid.virtualhost.*.exchange.*.binding = INFO  
qpid.virtualhost.*.queue.*.binding = INFO
```

The overlapping of configuration such as logging all details of connection from user guest and from ip 127.0.0.1 will not result in a duplicated logging if guest connects from 127.0.0.1.

Overlapping configuration doesn't result in multiple logging

```
qpid.connection.guest.* = INFO  
qpid.connection.<127.0.0.1>.* = INFO
```

Presentation of entities with multiple paths

Each entity will have a prescribed log format which will be designed to take into consideration its place in the hierarchy, see [Logging Format Design](#) for further details.

Log Configuration processing

The configuration will be processed top-to-bottom. This allows for defaults to be set and specific cases to have further logging enabled. The example below enables *INFO* logging for all events and specifically adds *DEBUG* logging for all activity related to the user 'guest'.

Example Control File

```
qpid.* = INFO  
qpid.connection.guest,* = DEBUG
```

Qpid Design - Queue Implementation

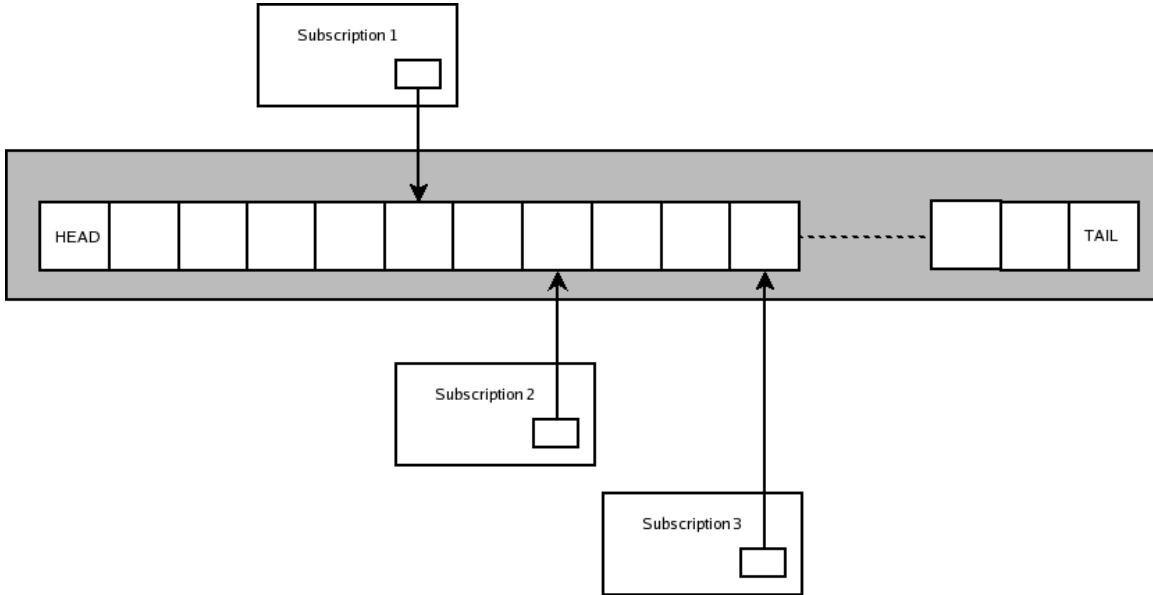
- Strict Ordering
- Enqueuing
- Priority Queues

Strict Ordering

The fundamental principle of the Queuing model is that the queue provides a strict order on the messages being enqueued. Furthermore that order is maintained through the lifetime of the entries on the queue: thus if a message is returned (e.g. the prefetched messages being released upon the consumer closing) the order of that message with respect to other messages on the queue is maintained.

The strict ordering is enforced by the use of a queue data-structure. In order for this to be performant, the data structure uses a lockless thread-safe designed based around the same algorithm used in the java.util.concurrent.ConcurrentLinkedList (more precisely it is based on the public domain implementation in the backport util concurrent project). See the section on Concurrent List implementations for more details.

Each subscription keeps a "pointer" into the list denoting the point at which that particular subscription has reached. A particular subscription will only deliver a message if it is the next AVAILABLE entry on the queue after the pointer which it maintains which matches any selection criteria the subscription may have.



Thread safety is maintained by using the thread-safe atomic compare-and-swap operations for maintaining queue entry state (as described above) and also for updating the pointer on the subscription. The queue is written so that many threads may be simultaneously attempting to perform deliveries simultaneously on the same messages and/or subscriptions.

Enqueuing

When a message is enqueued (using the enqueue() method on the AMQQueue implementation) it is first added to the tail of the list. Then the code iterates over the subscriptions (starting at the last subscription the queue was known to have delivered for reasons of fairness). For each subscription found it attempts delivery (details describe below). If the message cannot be delivered to any subscription then the "immediate" flag on the message is inspected. If the message required immediate delivery then the message is immediately dequeued, otherwise an asynchronous job is created to attempt delivery at a later point.

(Note there is a "shortcut" path for queues which have an exclusive subscriber. In this case we know there is one and only one subscriber and so we can go directly to trying to deliver to it without worrying about iterators, etc.)

Potential Issue: Looking at the code which performs the check of the immediate flag I believe there is a race condition:

```
if (entry.immediateAndNotDelivered())
{
    dequeue(storeContext, entry);
    entry.dispose(storeContext);
}
```

This does not look to be safe in the case where there is a simultaneous execution of an asynchronous deliver which may acquire the message between the check of immediateAndDelivered and dequeue. Instead of calling dequeue directly we should instead do a safe compare-and-swap test to make sure the entry state is "AVAILABLE" before setting it to DEQUEUED. The implementation of this should probably look much like the implementation of entry.dequeue except for the different expected starting state.

Immediate Delivery

For each subscription to the queue, we call the following code:

```

private void deliverToSubscription(final Subscription sub, final QueueEntry entry)
    throws AMQException
{
    sub.getSendLock();
    try
    {
        if (subscriptionReadyAndHasInterest(sub, entry)
            && !sub.isSuspended())
        {
            if (!sub.wouldSuspend(entry))
            {
                if (!sub.isBrowser() && !entry.acquire(sub))
                {
                    // restore credit here that would have been taken away by wouldSuspend
                    since we didn't manage
                    // to acquire the entry for this subscription
                    sub.restoreCredit(entry);
                }
                else
                {

                    deliverMessage(sub, entry);

                }
            }
        }
        finally
        {
            sub.releaseSendLock();
        }
    }
}

```

This code first takes a lock on the subscriber (this prevents it being removed while we are carrying out this operation). It then tests if the given subscription can take this message at the moment (see below for more details). It then further tests if there is enough flow control credit to send this message to this subscription. If there is credit (and the subscription is not a "browser" then it attempts to acquire the entry (entry.acquire(sub)). If the acquisition is successful (or if the subscription is a browser and thus does not need to acquire the entry) then the entry is delivered to the subscription, else the credit that would have been used by the message if sent is restored.

The most interesting method called in the above is subscriptionReadyAndHasInterest(sub, entry):

```

private boolean subscriptionReadyAndHasInterest(final Subscription sub, final QueueEntry
entry)
{
    // We need to move this subscription on, past entries which are already acquired, or
    deleted or ones it has no
    // interest in.
    QueueEntry node = sub.getLastSeenEntry();
    while (node != null && (node.isAcquired() || node.isDeleted() || !sub.hasInterest(node)))
    {

        QueueEntry newNode = _entries.next(node);
        if (newNode != null)
        {
            sub.setLastSeenEntry(node, newNode);
            node = sub.getLastSeenEntry();
        }
        else
        {
            node = null;
            break;
        }
    }

    if (node == entry)
    {
        // If the first entry that subscription can process is the one we are trying to
        deliver to it, then we are
        // good
        return true;
    }
    else
    {
        return false;
    }
}

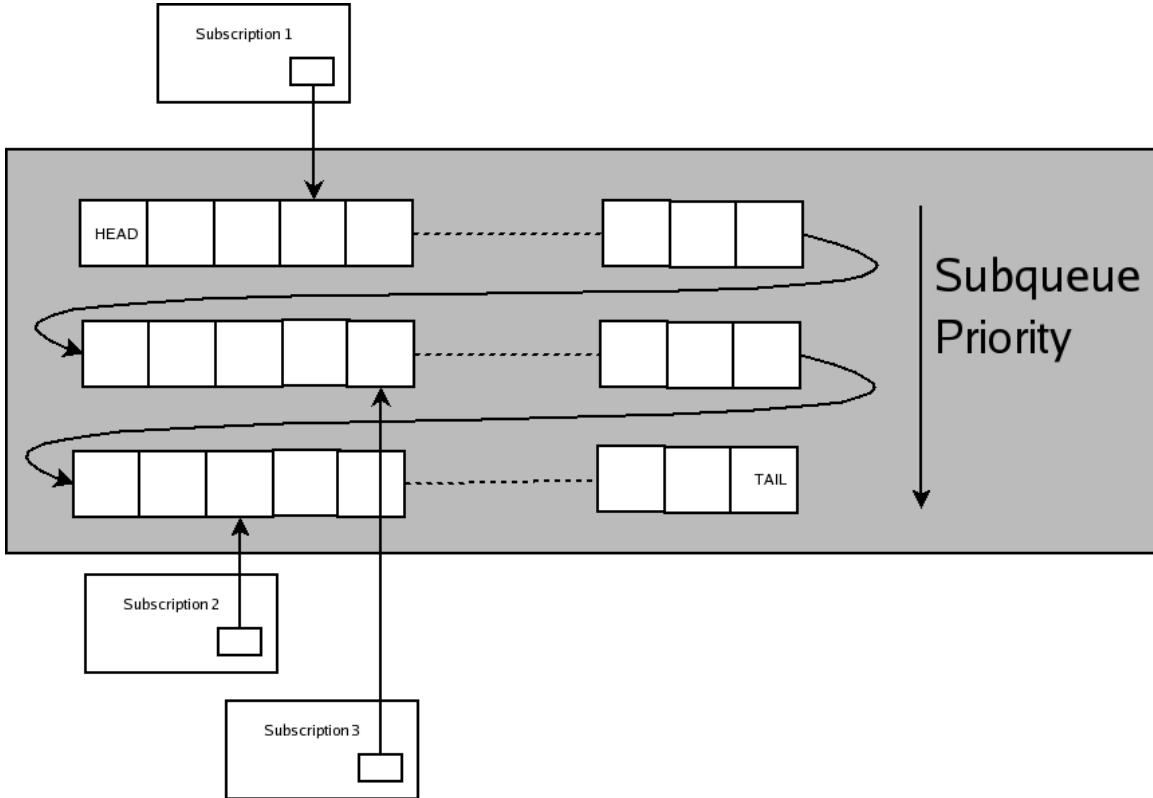
```

Here we see how the subscription is inspected to see where its pointer into the queue (the last seen entry) is in respect to the entry we are trying to deliver. We start from the subscription's current lastSeenEntry and work our way down the list passing over entries which are already acquired by other subscriptions, deleted, or which this subscription has no interest in (e.g. because the node does not meet the subscription's selection criteria); all the while we can update the lastSeenEntry to take it past the entries this subscription has now inspected. Performing this iteration we will eventually arrive at the next entry the subscription is interested in (or just fall off the end of the list). At this point either the next entry that the subscription is interested in is the entry we wish to deliver (success!) or not.

Priority Queues

The fundamental difference between Priority Queues and other Queues is that the strict ordering on the queue is not purely FIFO. Instead the ordering is a combination of FIFO and the priority assigned to the message. To provide strict priority ordering (where a message of higher priority will always be delivered in preference to a message of lower priority) we can implement a priority queue as an ordered list of standard sub-queues with the ordering between them defined such the tail of the highest priority sub-queue is followed by the head of the sub-queue of the next highest priority.

By defining the standard queue implementation such that the methods which determine the ordering between the nodes can be overridden, the implementation of such a strict priority queue is almost trivial.



The interface QueueEntryList provides an extension point for adding a new queue implementation type:

```
public interface QueueEntryList
{
    AMQQueue getQueue();

    QueueEntry add(AMQMessage message);

    QueueEntry next(QueueEntry node);

    QueueEntryIterator iterator();

    QueueEntry getHead();
}
```

The class PriorityQueueList provides the concrete implementation of a strict priority queue as defined above. The constructor takes an argument defining how many priority levels are to be provided.

When a message is added to the list by calling the add() method, the class first works out which sub-queue to add the message to. This is determined by an algorithm identical to that defined in the AMQP0-10 specification and compliant with the JMS requirements. The message is added to the tail of the appropriate sub-queue.

The next() method returns the QueueEntry which logically follows the QueueEntry provided by the caller. First we can simply look at the sub-queue in which the passed QueueEntry is actually in. If there is a subsequent entry in that sub-queue then we use that. If there is no subsequent entry in the sub-queue then we must find the next highest priority subqueue and take the head of that (repeating until we find a subqueue which is non-empty).

The getHead() method iterates over the subqueues to find the highest priority sibqueue which is non-empty and then returns the head of that subqueue.

The iterator() method returns an iterator that respects the ordering defined above.

The only other difference between a PriorityQueue and the standard queue is that new messages arriving may be logically "before" messages that have arrived previously (i.e. a high priority message is always logically prior to a low priority message in the queue). This means that on arrival of a message into the queue all subscriptions need to be inspected to make sure their pointer is not "ahead" of the new arrival.

Thus the entire implementation of AMQPriorityQueue is as follows:

```

public class AMQPriorityQueue extends SimpleAMQQueue
{
    protected AMQPriorityQueue(final AMQShortString name,
                               final boolean durable,
                               final AMQShortString owner,
                               final boolean autoDelete,
                               final VirtualHost virtualHost,
                               int priorities)
        throws AMQException
    {
        super(name, durable, owner, autoDelete, virtualHost, new
PriorityQueueList.Factory(priorities));
    }

    public int getPriorities()
    {
        return ((PriorityQueueList) _entries).getPriorities();
    }

    @Override
    protected void checkSubscriptionsNotAheadOfDelivery(final QueueEntry entry)
    {
        // check that all subscriptions are not in advance of the entry
SubscriptionList.SubscriptionNodeIterator subIter = _subscriptionList.iterator();
        while(subIter.advance() && !entry.isAcquired())
        {
            final Subscription subscription = subIter.getNode().getSubscription();
            QueueEntry subnode = subscription.getLastSeenEntry();
            while(subnode != null && entry.compareTo(subnode) < 0 && !entry.isAcquired())
            {
                if(subscription.setLastSeenEntry(subnode,entry))
                {
                    break;
                }
                else
                {
                    subnode = subscription.getLastSeenEntry();
                }
            }
        }
    }
}

```

The constructor merely ensures passes up the machinery to ensure a PriorityQueueList (as described above) is used for the underlying queueing model. The getPriorities() method is overridden by delegating to the PriorityQueueList and then the algorithm for updating the subscriptions' pointers into the queue is implemented in checkSubscriptionsNotAheadOfDelivery. Thread-safe compare-and-swap operations are used to update the pointer in-case other threads are also trying to move it; and the loop terminates early if the new QueueEntry has already been acquired.

Qpid Design - Message Delivery

- Asynchronous Delivery
- Subscriptions
- Removal
- Flow Control
- Acknowledgement
- Reject and Release

Asynchronous Delivery

If there are no subscriptions that can currently take delivery of a message then we need to schedule an asynchronous delivery. While the code is thread safe and could cope with multiple threads performing asynchronous delivery simultaneously, we limit ourselves to only having one asynchronous delivery job scheduled at any one time, so as not to overwhelm the broker:

```

public void deliverAsync()
{
    _stateChangeCount.incrementAndGet();

    Runner runner = new Runner();

    if (_asynchronousRunner.compareAndSet(null, runner))
    {
        _asyncDelivery.execute(runner);
    }
}

```

Here we first increment our count of "stateChanges". This provides us with a way of knowing between loops of the asynchronous delivery thread whether anything else has happened that makes it worth our while running the asynchronous delivery loop again (in effect it prevents us having to always add another thread to cope with race conditions where we want to start the async delivery just as it is ending). We then create a new instance of the asynchronous delivery "Runner", and attempt to make this instance the current one by means of the ubiquitous compare-and-swap operation. Here we test if we are the thread that moved the queue from having no asynchronous runner to having one; and if so we need to schedule the runner to execute by way of calling `_asyncDelivery.execute(runner)`.

The actual work of the asynchronous delivery is done in the `processQueue(Runnable runner)` method.

```

private void processQueue(Runnable runner) throws AMQException
{
    long stateChangeCount;
    long previousStateChangeCount = Long.MIN_VALUE;
    boolean deliveryIncomplete = true;

    int extraLoops = 1;
    int deliveries = MAX_ASYNC_DELIVERIES;

    _asynchronousRunner.compareAndSet(runner, null);

    while (deliveries != 0 &&
           ((previousStateChangeCount != (stateChangeCount = _stateChangeCount.get())) || deliveryIncomplete)
           && _asynchronousRunner.compareAndSet(null, runner))
    {
        // we want to have one extra loop after every subscription has reached the point where
        // it cannot move
        // further, just in case the advance of one subscription in the last loop allows a different
        // subscription to
        // move forward in the next iteration

        if (previousStateChangeCount != stateChangeCount)
        {
            extraLoops = 1;
        }

        previousStateChangeCount = stateChangeCount;
        deliveryIncomplete = _subscriptionList.size() != 0;
        boolean done = true;
    }
}

```

In this first fragment of the method we see the constraint on how long the asynchronous delivery will keep attempting to deliver more messages.

The first constraint "`deliveries != 0`" is testing a countdown value "deliveries" which is initialised with an initial maximum (currently set to 10): every successful delivery the thread makes decrements this counter. This implements a limit on how long the `processQueue` method will be allowed to run for, stopping this queue from starving other queues of processor time. At the end, if this countdown was the factor to cause the loop to terminate, the asynchronous delivery is scheduled to run again.

The second constraint "`((previousStateChangeCount != (stateChangeCount = _stateChangeCount.get())) || deliveryIncomplete)`" is testing whether there is provably nothing left to do on this queue. The first half tests if there have been any changes since the last iteration that have incremented that state change count (and thus require another loop), the second half says, "even if there haven't been any changes keep looping if last time round we thought there was still more to do".

The final constraint "`_asynchronousRunner.compareAndSet(null, runner)`" is our familiar compare-and-swap operation ensuring that this is the designated instance of the asynchronous processor running.

This loop runs, attempting to deliver one message in each iteration:

```
SubscriptionList.SubscriptionNodeIterator subscriptionIter =
_subscriptionList.iterator();
//iterate over the subscribers and try to advance their pointer
while (subscriptionIter.advance())
{
    boolean closeConsumer = false;
    Subscription sub = subscriptionIter.getNode().getSubscription();
    if (sub != null)
    {
```

Iterate over the subscriptions on the queue...

```
    sub.getSendLock();
```

Lock the subscription so it does not get deleted while attempting to deliver to it.

```
    try
    {
        QueueEntry node = moveSubscriptionToNextNode(sub);
```

Find the next node which the subscription should try to deliver by skipping over already acquired entries, if it is null then this subscription is at the tail of the queue.

```

    if (node != null && sub.isActive())
    {
        Keep a track of whether this subscription is really active and whether we managed to advance the
        pointer on this subscription in this loop (these values go into determining if there is anything
        left to do in a new loop).

```

```

        boolean advanced = false;
        boolean subActive = false;

        if (!(node.isAcquired() || node.isDeleted()))
        {
            if (!sub.isSuspended())
            {

```

The node is not yet acquired or deleted, and we can now be sure the subscription is active.

```

                subActive = true;
                if (sub.hasInterest(node))
                {

```

The following code is similar to that in the deliverToSubscription method described previously.
It should be possible to factor **this** out. The primary difference is the behaviour with a browser
where need to explicitly note that we have advanced.

```

                    if (!sub.wouldSuspend(node))
                    {
                        if (!sub.isBrowser() && !node.acquire(sub))
                        {
                            sub.restoreCredit(node);

                        }
                        else
                        {
                            deliverMessage(sub, node);
                            deliveries--;

                            if (sub.isBrowser())
                            {
                                QueueEntry newNode = _entries.next(node);

                                if (newNode != null)
                                {
                                    sub.setLastSeenEntry(node, newNode);
                                    node = sub.getLastSeenEntry();
                                    advanced = true;
                                }
                            }
                            done = false;
                        }
                    }
                    else // Not enough Credit for message and wouldSuspend
                {

```

This case covers the scenario where we are using bytes based flow control, and the currently available credit is less than the size of the next message. We need to wait either for the credit to be increased (which will cause a state change event) or the entry to be picked off by another subscription (which we capture with the state change listener)

```

        //QPID-1187 - Treat the subscription as suspended for
this message
// and wait for the message to be removed to continue delivery.
subActive = false;

                node.addStateChangeListener( new
QueueEntryListener(sub, node));
            }
        }
    else
    {
        // this subscription is not interested in this node so we
can skip over it
QueueEntry newNode = _entries.next(node);
        if (newNode != null)
        {
            sub.setLastSeenEntry(node, newNode);
        }
    }
}
}

```

Here we calculate if there is anything left to do on this particular subscription. If we are at the tail of the subscription, or the subscription is no longer active, then this subscription can be considered done. If all subscriptions are done then we are truly finished.

```

final boolean atTail = (_entries.next(node) == null);
done = done && (!subActive || atTail);

```

Here calculate if we need to auto-close the subscription - we do this if we are at the tail of the queue, and we didn't advance in this iteration and this is an auto-close subscription.

```

        closeConsumer = (atTail && !advanced && sub.isAutoClose());
    }
}
finally
{
    sub.releaseSendLock();
}

if (closeConsumer)
{
    unregisterSubscription(sub);

    ProtocolOutputConverter converter =
sub.getChannel().getProtocolSession().getProtocolOutputConverter();
    converter.confirmConsumerAutoClose(sub.getChannel().getChannelId(),
sub.getConsumerTag());
}
}

```

This ends the iteration over subscriptions, now we calculate if we believe there we should try iterating over the subscriptions again. We use the value of "done" we calculated while iterating over the subscriptions to determine if we need to loop again. If we believe we are done and we have already used our "extra" loop then we can stop. If we are "done" but we have not yet used the extra loop, then we decrement the extra loop counter (setting it to 0 might be clearer since 0 and 1 are the only valid values) and go round one more time. If we are not done then we restore extraLoops to 1.

```

        if (done)
    {
        if (extraLoops == 0)
        {
            deliveryIncomplete = false;
        }
        else
        {
            extraLoops--;
        }
    }
    else
    {
        extraLoops = 1;
    }
}
_asynchronousRunner.set(null);
}

```

This ends the the "while(...)" loop. The final action in the asynchronous process is to determine if we need to schedule ourselves for another execution:

```

// If deliveries == 0 then the limiting factor was the time-slicing rather than available
messages or credit
// therefore we should schedule this runner again (unless someone beats us to it :-) .
if (deliveries == 0 && _asynchronousRunner.compareAndSet(null, runner))
{
    _asyncDelivery.execute(runner);
}
}

```

Subscriptions

Subscription model the entities created by the receiving of a "Basic.Consume" event in AMQP0-8/0-9. That is they represent a relationship between an AMQP Channel (equivalent to a Java JMS Session) and a queue. As messages are placed on the queue, the queue takes responsibility for as quickly as possible finding a subscriber which is willing to take the message. The subscriber is responsible for delivering the message to the receiving client. As outlined above, a significant change introduced by the refactoring is that the Subscriptions now maintain state representing a pointer into the queue. This pointer represents the current position where the subscription can guarantee that no message prior to that is of interest to it. Generally this pointer only ever moves forward through the queue (see the section on reject and release for the exception to this rule). This is the only dynamic state maintained directly by the subscription.

Different subclasses of SubscriptionImpl are used to model the different behaviour associated with different acknowledgement modes. The subclasses used are AckSubscription, NoAckSubscription, BrowserSubscription and GetNoAckSubscription. The last of these is a special implementation which is used to model a Basic.Get command as a temporary subscription that can only ever receive one message. Modelling Get in this way mirrors how the same semantics are implemented in 0-10 and removes having two separate ways to dequeue messages from the queue.

Adding

When a "Basic.Consume" event is processed the subscription is added to the list of subscriptions on the queue, the "pointer" in the subscription is set to point at the head of the list of queue entries, and then an asynchronous job is kicked off to deliver to that subscription as many messages as can be delivered starting at the head. This uses an algorithm almost identical to that described above to asynchronous message delivery, except it only considers the one subscription. This is found within the "flushSubscription" method of the queue (flushing a subscription is a 0-10 concept where you attempt to send as much as possible to a given subscription and then signal completion when either the subscription's credit runs out or there are no more messages on the queue).

Future Improvement: Factor out the common code between flushSubscription and processQueue.

Removal

A consumer is removed either through the reception of a "Basic.Cancel" event or through the closure of the encapsulating channel. For thread safety, the first action is to remove the subscription from the list of subscriptions that the asynchronous delivery task iterates over. Next the subscription's close() method is called. This takes out a lock on the subscription (to avoid conflicting with any attempt to concurrently send to the subscription) and changes the subscription's state to "closed". The combination of these steps allow us to assert that after that point in time the subscription will not be used by any other threads to attempt to deliver messages. Next the subscription's pointer into the queue is null-ed out in a thread-safe way - this is done to prevent memory leaks due to references being held to points in the queue (due to the way that the concurrent-safe queues work "deleted" elements may not be eligible for garbage collection for some time).

Finally, if the queue is of "auto-delete" type and the subscription being removed is the last subscription attached to the queue, then the queue needs to be deleted.

Flow Control

There are now concrete classes modeling the behaviour of the flow control algorithm. These flow control managers are set at the subscription level. For AMQP 0-8 and 0-9 flow control still happens at a per-channel level, so the same instance of the flow control manager is shared between all subscriptions on a channel. For 0-10 implementations we will be able to use the same code to implement the per-subscription flow-control model that it utilizes.

Acknowledgement

Reject and Release

Messages delivered to a subscription may subsequently be returned to the queue either explicitly (by use of a reject command) or implicitly (by the closure of the channel). In this case the message must be made available again to subscribers to the queue. The issue here is that the pointers held by the subscriptions are likely to be in advance of the point to which the message is being returned. Thus for each message that is returned we must iterate over all subscribers to the queue, and if their current pointer is in advance of the returned message it must be moved back such that the next entry that that subscriber sees is the returned message. We do not reset the pointer for browsing consumers however as doing so would lead to all the browsed messages that are after the returned message in the queue being redelivered to the browsing subscription.

Java authorization plugins

The Qpid Java Server supports pluggable authorization modules through OSGi bundles.

New plugins must implement two classes. One of these should implement the org.apache.qpid.server.security.access.ACPlugin interface. The other should implement the org.apache.qpid.server.security.access.ACPluginFactory interface.

How authorization works

The collection of configured ACPlugins are managed by an ACManager class. This is queried by frame handlers as to whether access should be allowed or not. When this occurs, the manager conducts a vote amongst its plugins. If any plugin votes to deny access, authorization is denied. If a server-level plugin denies access, but a virtualhost level plugin explicitly allows access, the virtualhost vote overrides the server-level plugins and its vote is for access to be allowed. An instance of a plugin may abstain from a vote.

The ACPluginFactory Interface.

This interface has two methods: boolean supportsTag(String) and ACPlugin newInstance(Configuration). If the Factory can produce a plugin which is capable of handling the tag passed into supportsTag it must return true, otherwise it must return false.

If the plugin that the Factory is associated with supports that particular configuration tag, a new instance of that plugin should be created by newInstance and configured with the Configuration instance that is passed in.

The ACPlugin Interface.

This interface has two types of method. setConfiguration is used to pass a Configuration object to the plugin to allow it to access configuration information. This will always be the complete children of one of the <security> sections of the server configuration file (either server-wide or one for a specific virtualhost).

The AuthzResult authorise* methods allow the plugin to restrict or grant access for a particular action. All methods take in an AMQProtocolSession to provide access to the authentication data and the underlying socket. If access should be granted, AuthzResult.ALLOWED should be returned. If access should be denied, AuthzResult.DENIED should be returned. If the plugin has no opinion as to whether access should be permitted, it should return AuthzResult.ABSTAIN.

The AbstractACPlugin class

An abstract ACPlugin is provided that abstains from all votes. It is useful if the plugin you are implementing only cares about a few methods, extend this and you need only implement the authoriseFoo methods the plugin is interested in.

0.6 Broker BasicFlow Synchronisation Design

BasicFlow Synchronisation Design

The recently fixed QPID-1871/0.6 Java Client Dispatcher Changes raised the issue of QPID-2116 which is described here in more detail.

- Problem
- Solution

Problem

The initial problem under investigation(QPID-1871) was the Java client and when used with the Java Broker (0-8/9) protocol was resulting in messages returning out of order after a TxRollback. The initial analysis highlighted the Dispatcher as holding an incoming message and acking it late. The fix for QPID-1871 does not remove the potential for the Dispatcher to hold a message rather than attempt to clear all messages that the Dispatcher has to process so that none can be left behind. The change has fixed the related issue with the 0-10/Java Broker code path but the 0-8/Java broker was still failing. On further investigation it appears that the reason the Dispatcher had a message to reject after completion in the Java case was due to the broker violating the Flow status of the Channel.

The log extracts from the *RollbackOrderTest* highlight that whilst the Client has requested Flow = false and the Broker acknowledges with a FlowOk, one last message is still sent to the Client.

```

// Client Requests Flow = false/suspended
main 2009-09-28 13:09:20,994 DEBUG [apache.qpid.client.AMQSession] Setting channel flow :
suspended
...
// Broker Receives request acknowledges with FlowOk (not logged)
pool-1-thread-1 2009-09-28 13:09:21,020 DEBUG [qpid.server.protocol.AMQProtocolSession] Frame
Received: Frame channelId: 1, bodyFrame: [ChannelFlowBodyImpl: active=false]
pool-1-thread-1 2009-09-28 13:09:21,020 INFO [qpid.message] MESSAGE
[con:0(guest@anonymous(11875256)/test)/ch:1] [con:0(guest@anonymous(11875256)/test)/ch:1] CHN-1002
: Flow Stopped
pool-1-thread-1 2009-09-28 13:09:21,020 DEBUG [qpid.server.handler.ChannelFlowHandler]
Channel.Flow for channel 1, active=false
...
// Asynchronous delivery thread can be seen to be delivering a message to the client
pool-1-thread-2 2009-09-28 13:09:21,046 DEBUG [qpid.server.subscription.SubscriptionImpl]
(25976423) checking filters for message ((HC:32576775 ID:21 Ref:1)
pool-1-thread-2 2009-09-28 13:09:21,046 DEBUG [apache.qpid.server.AMQChannel] 1(8529229) Adding
unacked message(Message[(HC:32576775 ID:21 Ref:1)]: 21; ref count: 1 DT:289) with a
queue(org.apache.qpid.server.queue.SimpleAMQQueue@1c5ddd3) for
[channel=[anonymous(11875256)(guest):1], consumerTag=1, session=anonymous(11875256)]
...
// Client receives FlowOk message
pool-1-thread-1 2009-09-28 13:09:21,086 DEBUG [qpid.client.protocol.AMQProtocolHandler]
(2443549)Method frame received: [ChannelFlowOkBodyImpl: active=false]
pool-1-thread-1 2009-09-28 13:09:21,086 DEBUG [qpid.client.handler.ChannelFlowOkMethodHandler]
Received Channel.Flow-Ok message, active = false
...
// Client receives additional message
pool-1-thread-4 2009-09-28 13:09:21,143 DEBUG [qpid.client.protocol.AMQProtocolHandler]
(2443549)Method frame received: [BasicDeliverBodyImpl: consumerTag=1, deliveryTag=289,
redelivered=true, exchange=amq.direct, routingKey=RollbackOrderTest-testOrderingAfterRollback]
pool-1-thread-4 2009-09-28 13:09:21,144 DEBUG [qpid.client.handler.BasicDeliverMethodHandler] New
JmsDeliver method received:org.apache.qpid.client.protocol.AMQProtocolSession@c22a3b
pool-1-thread-4 2009-09-28 13:09:21,144 DEBUG [apache.qpid.client.AMQSession]
Message[ContentHeader org.apache.qpid.framing.ContentHeaderBody@1e5ba24] received in session

```

The additional message that is received causes the Dispatcher to awake from its `_queue.take()` call and hold until the connection is restarted. At this point the message is rejected as its delivery tag is old. However, the Java Broker has already resent the message with a new delivery tag.

Solution

What needs to happen is that when the the Channel is suspended we must ensure that all Subscriptions have noticed the change in state. This can be done by taking and releasing each of the Subscriptions send locks.

Slow Consumer Disconnect

Slow Consumer Problem Statement

The problem with slow consumers is that the broker must act as a buffer until they can catch up. However the broker does not have infinite resources so it will fail if the consumer does not catch up.

Consumers in the Java Broker

The types of queues where a slow consumer can occur boils down to two properties: durability and bound exchange.

Queues that are bound to the amq.direct exchange, i.e. JMS Queues, are not going to be included in this work.
Queues bound to other exchanges such as amq.match, the Headers exchange will also not be included in this work.

This reduces the queues to consider to just queues bound to the topic exchange.

Topics

In AMQP consumption is always from an AMQP Queue to avoid confusion with JMS Queues in the following discussion the term topic is defined to mean an AMQP Queue bound to the amq.topic exchange.

When a topic reaches a set threshold for message count, size or age the attached consumer session we have three options.

- Flow the producer.
- Disconnect the slow consumer.

Work has already been done to flow producers on queues: [Producer flow control](#).

This leaves us with two options.

Disconnect the Slow Consumer

For non-durable topics this means that it will be deleted so potentially freeing up the memory used by the messages. Remember the messages are shared across all topics so the memory will only be freed up when all the topics no longer require the message.

For durable topics (JMS Durable Subscriptions) disconnecting the consumer will leave the queue bound and receiving messages. This will only make the memory situation worse as we now have a queue with no consumer rather than just a slow consumer. If all the messages on the topic are persistent then they can be evicted from memory if required but there is no guarantee that all the messages will have been sent persistently.

On disconnection the consumer would receive an AMQP error, 506 Resource Limit Exceeded/Resource Error. For non-durable consumers this will always work. However, for a durable subscription it is possible that the consumer has disconnected when the limit is reached. So whilst deleting their subscription would be in line with the configuration it would not be expected by the user. The configuration for enabling slow consumer disconnection should allow for durable subscriptions to be maintained, targeting only transient subscriptions for disconnection.

Design Specification

This work is mainly focused on the broker however the client may also require changes to ensure that the error is correctly reported.

Broker Changes

Extension Point

To enable the broker to monitor the queues and perform the appropriate action we can extend an existing mechanism. That of the VirtualHost housekeeping thread. This is a thread that checks all the queues for alerting purposes. Currently this is done via a single TimerTask however by updating this to utilise a ScheduledThreadPoolExecutor we can run arbitrary processes in the pool and ensure that any error in their operation does not prevent them from running on their defined schedule. This will allow slow consumers to be checked and disconnected on a periodic basis.

Queue Detection

The target queues can easily be identified by checking their bindings. Topics are all bound to the TopicExchange. Once we have identified a topic exchange we can use the queue assigned configuration to determine if we are checking depth, messageCount or messageAge as a means of selecting the subscription for processing.

Processing

We will identify the session/channel that the subscription is on and close it with the appropriate error code. The delete will then ensure that the queue is deleted and all messages released after the session/channel has been closed.

Error Code

The AMQP error code 506 will be used to communicate the failure to the client. This is defined as a Resource Error or Resource Limit Exceeded in 0-8/9/91 and 0-10 respectively. In addition the protocol allows for a textual description to be sent back to the client. In this field we will send 'Consuming too slow.'

Client Changes

Error Processing

Currently a 0-8/9/91 Session will propagate a ChannelCloseException to the client via the JMS ExceptionListener we need to ensure that the 0-10 code path will create the same Exception type and present it to the JMS ExceptionListener.

After Effects

The Exception that is thrown should not be classed as a 'HardError' which would result in Failover starting. After the exception has been received the Consumer and the associated Session should be closed however the Connection will still be operational. This will allow the client to perform recovery without having to reestablish its Connection.

Configuration

Picking up on the [Topic Configuration Design](#) the addition of slow consumer configuration would be done using a 'consumer' element.

The topic currently exposes three properties that we can use to control the client, depth, oldest message, and count. The configuration will provide the option to one or all of these values to apply to the specified topic. In the situation where more than one value is specified they will all be used to trigger the policy. e.g. setting count to 10 and depth to 1024 would allow the 10 messages to exist as long as their total size was not more than 1024.

One additional property that would be of use here would be the consumption rate. If the topic reported the consumption rate this property could be used to define a threshold that the consumer must stay above.

Consumer Element for configuration

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

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

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

    <!-- Policies configuration -->
    <policy name="Delete">
        <options>
            <option name="delete-persistent" value="true"/>
        </options>
    </policy>
</consumer>
```

This `<consumer>` element will be added to the existing queue configuration to allow specific durable subscriptions to be identified and processed. In addition a new `<topic>` element will be added to allow configuration for topics. The resulting section of xml would look like this:

Topic configured for slow consumer disconnection

```
<topic key="stocks.us.*">
    <consumer>
        <!-- The depth before which the policy will be applied-->
        <depth>4235264</depth>

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

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

        <!-- Policies configuration -->
        <policy name="Delete">
            <options>
                <option name="delete-persistent" value="true"/>
            </options>
        </policy>
    </consumer>
</topic>
```

Testing Spec

Testing for this new feature will mainly rely on system testing.

Unit Testing

Configuration changes need to be validated as part of existing Configuration Testing.

System Testing

System testing requires a number of dimensions to be varied.

1. Protocol Version
2. Client Ack Mode
3. Client Consume mode
4. Topic Durability

Protocol Version

Testing should be completed at a minimum on the two sets of protocol 0-8/0-9/0-91 and 0-10. Ideally the test would be run on each protocol version to verify the protocol exception is correctly propagated.

Client Ack Mode

The tests should be run against each client ack mode to validate if there is any difference in the exception handling. Transacted for instance should fail to commit by throwing the expected exception as well as having the exception appear on the ExceptionListener.

The NoAck case has addition issue in that it can overwhelm IO layer in presence of a slow consumer. This should be verified however its resolution is beyond the scope of this work.

Client Consume Mode

The client can consume in one of two ways. Synchronously using receive() or asynchronously using a MessageListener.

Topic Type

Topics can be created as durable or non-durable(transient) both of these configurations should be tested as any exception should be reported in the same way. Additionally as the configuration has the ability to selectively delete durable topics this must also be tested. The required exception should be thrown when enabled but not thrown when the configuration does not control durable topics.

Topic Configuration Design

Topic Configuration Design

Currently we are unable to provide configuration for topics. What follows is a proposal to augment our configuration to provide a 'topic' section that can be validated.

The first important section is the topic definition. Rather than using dynamically named elements that we cannot validate we use a property to name the topic.

```
<topic key="stocks.us.*">
```

This approach allows us to simply use the dotted notation as well as enabling us to validate the xml.

Currently alerting is picked up from the global 'queues' configuration section. Now that we have a 'topic' section we can add an 'alerting' element which can articulate more clearly the alerting values. This will allow configuration on a per topic basis and remove the confusion that has arisen from the 'maximum*' elements in the 'queues' configuration section.

```
<alerting>
    <depth>2117632</depth>
    <messageSize>2117632</messageSize>
    <messageAge>300000</messageAge>
    <messageCount>25</messageCount>
</alerting>
```

To further clarify the use of existing properties a producer element can be used to house configuration that defines how producers to this topic are to be treated. This currently means the [Producer flow control](#) options but it does give some additional clarity to the 'capacity' element that is also mistaken to be a hard limit on the capacity of the queue.

```
<producer>
    <!-- set the topic capacity to 10Mb -->
    <capacity>10485760</capacity>
    <!-- set the resume capacity to 8Mb -->
    <flowResumeCapacity>8388608</flowResumeCapacity>
</producer>
```

The slow consumer design calls for configuration which is defined on that [page](#) and has been included here for completeness.

These 'topic' elements would of course be wrapped in a 'topics' element as is done for 'queues'.

Here is a complete topics example.

New Topic Section

```
<topics>

    <topic key="stocks.us.*">
        <alerting>
            <depth>2117632</depth>
            <messageSize>2117632</messageSize>
            <messageAge>300000</messageAge>
            <messageCount>25</messageCount>
        </alerting>

        <producer>
            <!-- set the topic capacity to 10Mb -->
            <capacity>10485760</capacity>
            <!-- set the resume capacity to 8Mb -->
            <flowResumeCapacity>8388608</flowResumeCapacity>
        </producer>

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

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

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

            <!-- Available Policies : Delete | Cycle -->
            <policy name="Delete">
                <options>
                    <option name="include-persistent" value="true"/>
                </options>
            </policy>
        </consumer>

    </topic>

</topics>
```

Qpid Java Client refactoring

Summary

The goals of the re-factoring are as follows.

1. Provide an AMQP protocol level API.
2. Achieve a clear separation of concerns.
3. Implement proper state and event handling.
4. Provide extensibility to extend the API at any level.
5. Provide a unified interface for configuration through jvm arguments,xml ..etc. (Used commons configuration)

The Prototype is available here https://svn.apache.org/repos/asf/incubator/qpid/branches/client_restructure/java/newclient/

The following document describes the design. [java_amqp_client_design.pdf](#)

Distributed Testing

Testing Proposal.

Use Cases.

The following usage scenarios are covered by this test framework design proposal:

Performance Testing.

- Distributed testing.

Want to be able to distribute performance tests across many machines in parallel, in order to more accurately simulate real usage scenarios and to be able to fully stress test the broker under load.

Want to be able to run performance tests, with the test parameters configurable, so that any reasonable topology can be simulated and performance estimated.

For example:

1. P2P test. On 10 machines, simulating load of 1000 clients. Each machine will run 100 test circuits on 100 connections. Both ends of the test circuit will reside on the same machine, with each client consuming its own messages. Results over all machines to be collated to arrive at total throughput figures.
2. Pub/Sub test. On 10 machines, simulating load of 1000 subscribers, 1 publisher. One machine acts as the sending half of the test circuit. The 1000 subscriber nodes, the receiving end of the circuit, are distributed as evenly as possible across the other 9 machines. The publisher sends messages and collates throughput or latency measurements on the test circuit.

System Testing.

- Thorough testing.
- Functional testing at the product surface; behavioural tests to carry forward as the system evolves.

Configurable framework, capable of exercising every imaginable combination of options, both in-vm broker and standalone, across one client/test circuit up to many clients/test circuits in parallel.

Want to be able to exercise as many different combinations of test configuration parameters as possible in order to generate the most comprehensive testing of the broker and protocol as possible. Exhaustive testing of every combination will discover bugs.

Want to test the system behaviour at its surface. That is, through the JMS API or through a more direct AMQ API where necessary. The test framework will ideally, abstract out the exact details of the API used, in order to allow forward evolution of the AMQ API.

Want to be able to set up each producer or consumer in a test circuit identically by default. More specific tests to be able to produce variations on this theme to test specific scenarios. For example test circuits the send both persistent and transient messages etc.

Build tests out of a standardized construction block.

- Diagram: The test circuit.

Publisher/Receiver pair.

Each end of which is a Producer/Consumer unit.

M producers, N consumers, talking over Z destinations.

The standard construction block for a test, is a test circuit. This consists of a publisher, and a receiver. The publisher and receiver may reside on the same machine, or may be distributed. Will use a standard set of properties to define the desired circuit topology.

Tests are always to be controlled from the publishing side only. The receiving end of the circuit is to be exposed to the test code through an interface, that abstracts as much as possible the receiving end of the test. The interface exposes a set of 'assertions' that may be applied to the receiving end of the test circuit.

In the case where the receiving end of the circuit resides on the same JVM, the assertions will call the receivers code locally. Where the receiving end is distributed across one or more machines, the assertions will be applied to a test report gathered from all of the receivers. Test code will be written to the assertions making as few assumptions as possible about the exact test topology.

A test circuit defines a test topology, M producers, N consumers, Z outgoing routes between them.

The publishing end of each test circuit always resides on a single JVM, even if M > 1. If publishers are to be distributed across many machines, the test framework itself provides the scaling by running the same test circuit many times in parallel. This means that it is possible to have an arbitrary number of message publishers across one or many machines, determined by the test setup.

The receiving half of the circuit may be local, in which case all messages come back to the same machine, or distributed in which case they may be received by many machines.

There are therefore two ways in which tests may be distributed across multiple nodes in a network; many test circuits may be distributed and run in parallel and/or the receiving ends of those circuits may be distributed or local.

Each node in the network can play up to 2 roles in any given test; publisher or receiver. It is possible to play both roles at once, but would like to have a 'single_role' flag, that can be set to ensure that test nodes taking one role, will not participate in the other for the duration of a test. For example, in the pub/sub test want one publisher and the remaining nodes to distribute the receiver role amongst themselves.

Probing for the available test topology.

- Diagram: The available topology.

When the test distribution framework starts up, it should broadcast an 'enlist' request on a known topic. All available nodes in the network to reply in order to make it known that they are available to carry out tests. For the requested test case, C test circuits are to be run in parallel. Each test defines its desired M by N topology for each circuit. The entire network may be available to run both roles, or the test case may have specified a limit on the number of publishing nodes and set the 'single_role' flag. If the number of publishing nodes exhausts the available network and the single role flag is on, then there are no nodes available to run the receiver roles, the test will fail with an error at this point. Suppose there are P nodes available to run the publisher roles, and R nodes available to run the receiver roles. The C test circuits will be divided up as evenly as possible amongst the P nodes. The C * N receivers will be divided up as evenly as possible amongst the R nodes.

A more concrete example. There are 10 test machines available. Want to run a pub/sub test with 2 publishers, publishing to 50 topics, with 250 subscribers, measuring total throughput. The distribution framework probes to find the ten machines. The test parameters specify a concurrency level of 2 circuits, limited to 2 nodes, with the single role flag set, which leaves 8 nodes to play the receiver role. The test parameters specify each circuit as having 25 topics, unique to the circuit, and 125 receivers. The total of 250 receivers are distributed

amongst the 8 available nodes, 31 each, except for two of them which get 32. The test specifies a duration of 10 minutes, sending messages 500 bytes in size using test batches of 10000 messages, as fast as possible. The distribution framework sends a start signal to each of the publishers. The publishers run for 10000 messages. The publishers request a report from each receiver on their circuit. The receivers send back to the publishers a report on the number of messages received in the batch. The publishers assert that the correct number for the batch were indeed received, and log a time sample for the batch. This continues for 10 minutes. At the end of the 10 minutes, the publishers collate all of their timings, failures, errors into a log message. The distribution framework requests the test report from each publishing nodes, and these logs are combined together to produce a single log for the entire run. Some stats, such as total time taken, total messages through the system, total throughput are calculated and added as a summary to the log, along with a record of the requested and actual topology used to run the test.

- Diagram: The requested test applied onto the available topology.

Test Procedures.

A variety of different tests can be written against a standard test circuit, many of these will follow a common pattern. One of the aims of using a common test circuit configured by a number of test parameters, is to be able to automate the generation of all possible test cases that can be produced from the circuit combined with the common testing pattern, and an outline of a procedure for doing this is described here. The typical test sequence is described below:

A typical test sequence.

1. Initialize the test circuit from the default parameters, plus specific settings for the test.
2. Create the test circuit. The requested test parameters are applied to the available topology to produce a live circuit.
3. Send messages.
4. Request a status report.
5. Assert conditions on the publishing end of the circuit.
6. Assert conditions on the receiving end of the circuit.
7. Pass or fail the test.

The thorough test procedure.

The thorough test procedure uses the typical test sequence described above, but generates all of combinations of test parameters and corresponding assertions against the results.

The all_combinations function produces all combinations of test parameters described in Appendix A.

all_combinations : List<Properties>

The expected_results function, produces a list of assertions, given a set of test parameters. For example, mandatory && no_route -> assertions.add(producer.assertMessageReturned), assertions.add(receiver.assertMessageNotReceived).

expected_results: Properties -> List<Assertions>

For parameters : all_combinations
test_circuit = new TestCircuit(parameters).
test_circuit.start.

Send messages.
Request status.

For assertion : expected_results(parameters)
Assert(assertion).

Appendix A - Test Parameters.

	Possible Values	Default Value
Connection properties.		
broker	tcp, vm	tcp://localhost
vhost		<empty>
username		guest
password		guest
Topology properties.		
max_publishing_node		1
single_role	true, false	true
Circuit properties.	Total: $2^2 = 4$ combinations.	
num_publishers		1
num_consumers		1

num_destinations		1
base_out_route_name		ping
base_in_route_name		pong
bind_out_route	true, false	true
bind_in_route	true, false	false
consumer_out_active	true, false	true
consumer_in_active	true, false	false
JMS flags and options.	Total: $2 * 2 * 2 * 6 = 48$ combinations.	
transactional	true, false	false
persistent	true, false	false
no_local	true, false	false
ack_mode	tx, auto, client, dups_ok, no_ack, pre_ack	auto
AMQP/Qpid flags and options.	Total: $2^4 = 16$ combinations.	
exclusive	true, false	false
immediate	true, false	false
mandatory	true, false	false
durable	true, false	false
prefetch_size		
header_fields		
Standard test parameters.	Total: 3 combinations.	
message_size	no_body, one_body, multi_body	one_body
num_messages		100
outgoing_rate		
inbound_rate		
timeout		30 seconds
tx_batch_size		100
max_pending_data		

Total combinations over all test parameters: $4 * 48 * 16 * 3 = 9216$ combinations.

Defaults give an in-VM broker, 1:1 P2P topology, no tx, auto ack, no flags, publisher -> receiver route configured, no return route.

Appendix B - Clock Synchronization Algorithm.

On connection/initialization of the framework, synch clocks between all nodes in the available topology. For in vm tests, the clock delta and error will automatically be zero. For throughput measurements, the overall test times will be long enough that the error does not need to be particularly small. For latency measurements, want to get accurate clock synchronization. This should not be too hard to achieve over a quiet local network.

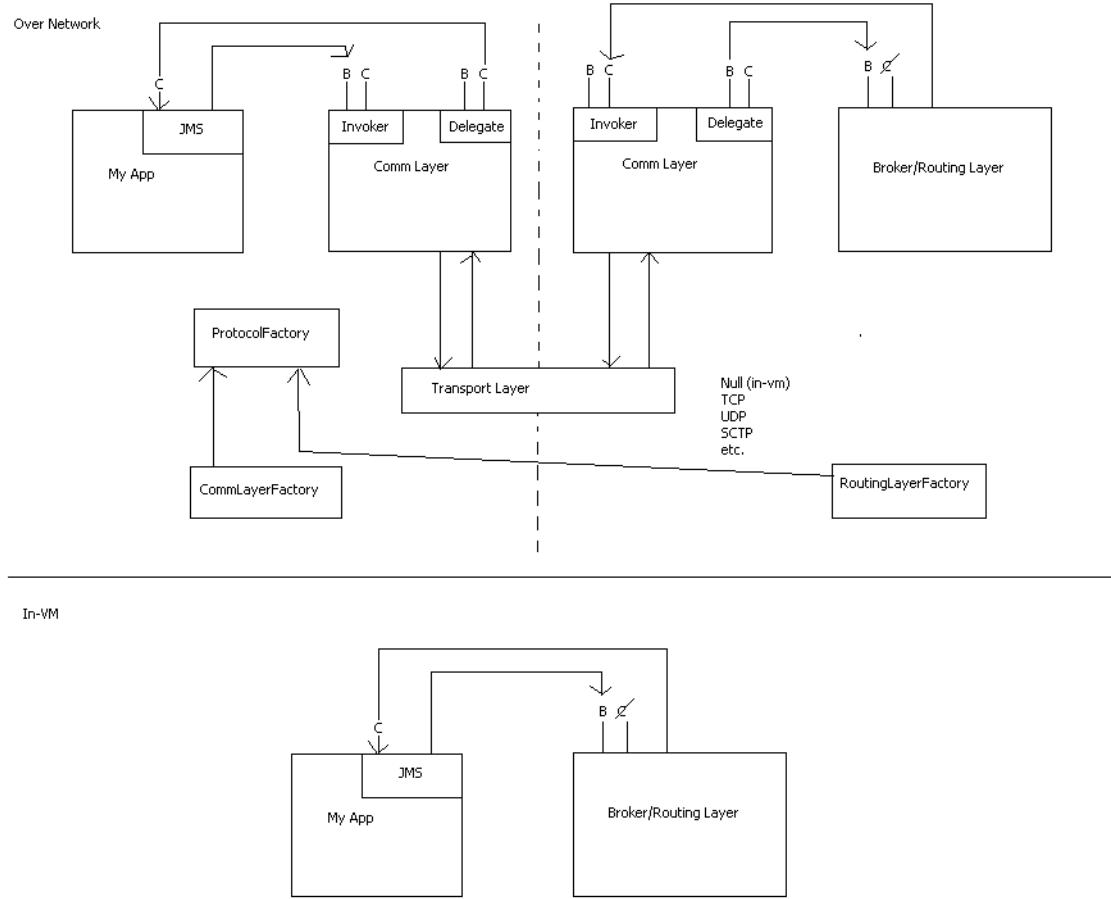
After determining the list of clients available to conduct tests against, the Coordinator synchronizes the clocks of each in turn. The synchronization is done against one client at a time, at a fairly low messaging rate over the Qpid broker. If needed, a more accurate mechanism, using something like NTP over UDP could be used. Ensure the clock synchronization is captured by an interface, to allow better solutions to be added at a later date. Here is a simple algorithm to get started with:

1. Coordinator tells client to synchronize its clock with the coordinators time.
2. Client stamps current local time on a "time request" message and sends to Coordinator.
3. Upon receipt by Coordinator, Coordinator stamps Coordinator-time and returns.
4. Upon receipt by Client, Client subtracts current time from sent time and divides by two to compute latency. It subtracts current time from Coordinator time to determine Client-Coordinator time delta and adds in the half-latency to get the correct clock delta.
5. The first result should immediately be used to update the clock since it will get the local clock into at least the right ballpark.
6. The Client repeats steps 1 through 3, 25 or more times, pausing a few tens of milliseconds each time.
7. The results of the packet receipts are accumulated and sorted in lowest-latency to highest-latency order. The median latency is determined by picking the mid-point sample from this ordered list.
8. All samples above approximately 1 standard-deviation from the median are discarded and the remaining samples are averaged using an arithmetic mean.

The above algorithm includes broker latency, two network hops each way, plus possible effects of buffering/resends on the TCP protocol. A fairly easy improvement on it might be:

1. Coordinator tells client to synchronize its clock with the coordinators time, provides a port/address to synchronize against.
2. Clients sends UDP packets to the Coordinators address and performs the same procedure as outlined above.

Low-Level API Diagram



Weekly QPID Developer Meetings

Qpid Java Meeting Minutes 04-04-2008

Agenda

- ApacheCon Europe
- Update on M2.1 release process
- Review of code commits
- Review of new JIRAs
- Update on GSoC projects

Attendees

ApacheCon Europe

No-one available to travel on those dates

Update on M2.1 release process

Need to update the release notes in the release

Taking out the .svn directories

May cause the release to slip by a couple of days

Review of code commits

Revision	Committer	Date	Comment (1st line)	
r642346	gsm	2008-03-28	Prefer binding key for unbind if specified.	
r642375	nsantos	2008-03-28	QPID-885: patch from Ted Ross	
r642959	gsm	2008-03-31	Prevent broker exit on receiving connection with invalid protocol version.	
r642981	gsm	2008-03-31	Re-introduced old 'no-local' behaviour for exclusive queues via a proprietary arg to queue.declare.	
r643032	ritchiem	2008-03-31	QPID-890 : Removed old references to VHostPrincipalDatabase and an errant old.PrincipalDatabaseAccessManager change.	
r643067	gsm	2008-03-31	Updated xml fragment to reflect correct types for connection.start.mechanisms, connection.start.locales and connection.open.capabilities	
r643086	gsm	2008-03-31	Allow zero sized arrays (with no typecode or count)	
r643153	aidan	2008-03-31	Add licences	
r643154	aidan	2008-03-31	Created prematurely, will recreate from release branch	
r643155	aidan	2008-03-31	Branch for M2.1 release	
r643162	aidan	2008-03-31	Update version in poms	Some versions are wrong in the poms, need correcting (fixed later)
r643165	aidan	2008-03-31	Tag RC1	
r643442	nsantos	2008-04-01	QPID-892: Make qpidd daemon not run as root (rpm install)	
r643472	gsm	2008-04-01	Fix some erroneous definitions in the transitional xml fragment for 0-10.	
r643478	gsm	2008-04-01	Added a dump method to buffer for debugging io (patch from rafael.s@redhat.com)	
r643482	gsm	2008-04-01	Further correction to transitional xml def for final 0-10 (using old schema)	
r643582	aidan	2008-04-01	Add my gpg key	
r643597	nsantos	2008-04-01	QPID-892 - use daemon params instead of runuser; store pid of qpidd daemon to kill single instance	
r643613	aidan	2008-04-01	Set version to M2.1 for all, it's release time	
r643624	aidan	2008-04-01	Tag the first M2.1 to get voted on	
r643822	arnaudsimon	2008-04-02	QPID-829 Remove 0.10 specific URL. The code path is now selected based on broker response. We first try the highest protocol version and update the handler if the broker replies with a different protocol version. NOTE that we need to update the current java broker and 0.8 client for handling protocol headers. This should happen with the M2.1 merge. For the moment we only support an in VM 0.8 broker. Moreover, we'll need to migrate to a 0.10 vs 99.0 protocol version.	Question coding standards on import statements (amend to allow .* ; remove unneeded imports) ; Should probably add ability to set explicit AMQP version
r643891	aconway	2008-04-02	Fix gcc 4.3 warnings.	
r643894	arnaudsimon	2008-04-02	QPID-884 Updated ant for using a profile. I have created a default profile that runs the tests against an 0.8 in VM broker and cpp-async and cpp-sync that respectively runs the test against an 0.10 cpp broker with async store and with sync store.	Update wiki documents to discuss the available stores ; document where to put them ; make another test which refers to no store
r643900	aconway	2008-04-02	Fix doxygen warnings.	
r643914	aconway	2008-04-02	Fix gcc 4.3 warnings.	
r643924	arnaudsimon	2008-04-02	QPID-884 made ant task test alton/error/failure configurable from profile file	
r643957	aconway	2008-04-02	Fixed logger warning on F9.	

r643995	aconway	2008-04-02	Encoding/decoding for new types: amqp_0_10::Map, amqp_0_10::UnknownType	
r644005	aconway	2008-04-02	Fix compile error on rhel5.	
r644125	aconway	2008-04-03	Fix serialize test failure on 64 bit architectures.	
r644245	arnaudsimon	2008-04-03	QPID-897 this test was intermittently failing because of too short timeouts. This fix is a temporary measure until we agree about using a configurable receive timeout.	hold off on permanent fix until after merge
r644287	kpvdr	2008-04-03	Patch from Ted Ross (see QPID-893): This patch enables management of plugged-in store modules.	
r644308	aconway	2008-04-03	amqp_0_10/built_in_types.h	
r644413	aconway	2008-04-03	src/qpid/amqp_0_10/Map.h,.cpp: use preview encoding temporarily.	
r644461	aconway	2008-04-03	rubigen/0-10/exceptions.rb:	
r644533	aconway	2008-04-03	qpid/Serializer.h, qpid/amqp_0_10/Codec.h:	
r644688	arnaudsimon	2008-04-04	QPID-796: Added ability to enable/disable message prefetching. Prefetching is controlled through the property max_prefetch, it is turned off when max_prefetch =0. (this is 0.10 code path change)	'Rause JIRA for spelling mistakes in classes (e.g. BasicMessageConsumer_0_10 strated() BasicMessageConsumer, etc ; startDispatcher ... Replace ClientProperties.MAX_PREFETCH == 0 with method e.g. disallowMessagePrefetch() ... maybe make a connection URL property
r644689	arnaudsimon	2008-04-04	QPID-798 Added boolean property fully_sync when true a sync is sent after a persistent message is transferred.	Make property per connection, rather than system wide; maybe change name to SYNC_PERSISTENT to denote only used on persistent messages

Review of new JIRAs

Key	Component/s	Affects Version/s	Summary	Status	Assignee	Reporter
QPID-902	C++ Broker, Python Client		Management Improvements for C++ Broker	Open	Unassigned	Ted Ross
QPID-901	Java Client		update the java client to the 0-10 final spec	Open	Rafael H. Schloeming	Rafael H. Schloeming
QPID-900	Java Common	M2.1	[Java Common] AMQShortString should not create new byte[] backing when based off a heap buffer	Open	Rob Godfrey	Rob Godfrey
QPID-899	Java Common	M2.1	[Java Common] Bug in AMQShortString on tokenized substrings	Open	Rob Godfrey	Rob Godfrey
QPID-898	Ant Build System		remove .tar.bz2 from release target	Open	Unassigned	Nuno Santos
QPID-897	Java Tests	M3	Configurable receive timeout	Open	Unassigned	Arnaud Simon
QPID-896	Java Client		Provide Simple Pub/Sub examples that do not use extends	Open	Martin Ritchie	Martin Ritchie
QPID-895			FailoverSingleServer delays on first connection.	Open	Unassigned	Martin Ritchie
QPID-894	Java Common		Dependency on commons-lang: real need for 2.2, or can use 2.1?	Open	Rafael H. Schloeming	Nuno Santos
QPID-893	C++ Broker		Enable management of plugged-in store module	Closed	Unassigned	Ted Ross
QPID-892	C++ Broker		Make qpidd daemon not run as root (rpm install)	Resolved	Unassigned	Nuno Santos
QPID-891	Java Broker		QueueDeclare reports incorrect message count on queue	Open	Unassigned	Martin Ritchie

QPID-890		M2.1	Broker transient_config.xml still contains reference to outdated principal database :PlainPasswordVhostFilePrincipalDatabase	Resolved	Martin Ritchie	Martin Ritchie
QPID-889	Java Broker	M2, M2.1	Requirement for _reapingStoreContext should be removed.	Open	Martin Ritchie	Martin Ritchie
QPID-888		M2, M2.1	Management methods don't correctly protect _lock from being lost on error	Open	Unassigned	Martin Ritchie
QPID-887	Java Broker	M2, M2.1	QueueHousekeeping threads are poorly named	Open	Unassigned	Martin Ritchie
QPID-886	Java Broker	M2, M2.1	CSDM.removeExpired() doesn't relinquish lock causing broker hang.	Open	Martin Ritchie	Martin Ritchie
QPID-885	Python Client		Broker configuration utility	Closed	Unassigned	Ted Ross
QPID-884	Java Tests	M3	Improve ant task test configuration	Closed	Arnaud Simon	Arnaud Simon
QPID-883	Python Client		Synchronous API in management.py	Closed	Unassigned	Ted Ross
QPID-882	Java Client	M3	XAResource does not check pre and post-conditions	Open	Arnaud Simon	Arnaud Simon
QPID-881	Java Client	M3	Commands waiting on a future are not notified of a connection closed event	Open	Rafael H. Schloeming	Arnaud Simon
QPID-880	Java Client	M2,M2.1	Client is still using broker set temporary queue names	Open	Unassigned	Martin Ritchie
QPID-879	C++ Broker		Adding support for XML-based routing	Open	Unassigned	Jonathan Robie
QPID-878	Java Tests	M3	Some tests are not run by ant as their class name does not end by Test	Closed	Arnaud Simon	Arnaud Simon
QPID-877	C++ Broker, Python Client		Management protocol and API improvements	Closed	Unassigned	Ted Ross

Update on GSoC projects

Had some interest, not all have been submitted.

Qpid Java Meeting Minutes 11-04-2008

Agenda

- Update on M2.1 release process
- Review of code commits
- Review of new JIRAs
- Update on GSoC projects

Attendees

Rob Godfrey
Aidan Skinner
Marnie McCormack
Martin Ritchie
Carl Trieloff
Rajith Attapatu
Rafael Schloeming
Gordon Sim

Update on M2.1 release process

RC3 is out, release instructions need to be updated.
C++ documentation should exist

Review of code commits

revision	committer	date	comment

r644689	arnaudsimon	2008-04-04 13:11:38 +0100 (Fri, 04 Apr 2008)	QPID-798 Added boolean property fully_sync when true a sync is sent after a persistent message is transferred. .
r644727	aconway	2008-04-04 15:42:36 +0100 (Fri, 04 Apr 2008)	src/qpid/amqp_0_10/Assembly.cpp,.h:
r644732	nsantos	2008-04-04 15:58:01 +0100 (Fri, 04 Apr 2008)	QPID-898: move bz2 generation from the release target to a new release-all target
r644806	kpvdr	2008-04-04 19:14:42 +0100 (Fri, 04 Apr 2008)	Patch from Ted Ross (see QPID-902): This patch contains the following improvements for management:\n1) Schema display cleaned up in the python mgmt-cl\n2) Locking added automatically to management object accessors (manual locking removed from broker/Queue.cpp)\n3) Schemas are now pre-registered with the management agent using a package initializer. This allows management consoles to get schema information for a class even if no instances of the class exist.
r644812	kpvdr	2008-04-04 19:25:08 +0100 (Fri, 04 Apr 2008)	Additional files for Ted Ross's checkin
r644845	aconway	2008-04-04 20:35:14 +0100 (Fri, 04 Apr 2008)	Minor cleanup of base Exception and python_tests script.
r644917	aconway	2008-04-04 22:00:40 +0100 (Fri, 04 Apr 2008)	src/qpid/amqp_0_10/Exception.h
r645470	gsim	2008-04-07 12:51:07 +0100 (Mon, 07 Apr 2008)	AsynchIoAcceptor.cpp: Limit output from codec to one buffer per 'idle' call.
r645593	aidan	2008-04-07 17:27:20 +0100 (Mon, 07 Apr 2008)	Add toplevel NOTICE and LICENSE files, add Felix to java/resources/NOTICE, make sure all spec files are included in java source distribution
r645663	aconway	2008-04-07 21:12:31 +0100 (Mon, 07 Apr 2008)	Encoding/decoding for packed structs and optional members.
r645664	aconway	2008-04-07 21:13:59 +0100 (Mon, 07 Apr 2008)	Missing from last commit.
r645670	aconway	2008-04-07 21:42:28 +0100 (Mon, 07 Apr 2008)	Fix rhel5 build errors.
r645685	astitcher	2008-04-07 21:59:02 +0100 (Mon, 07 Apr 2008)	Fixed time classes for some changes that misunderstood how they are supposed
r645699	aconway	2008-04-07 22:16:48 +0100 (Mon, 07 Apr 2008)	rubygen/0-10/specification.rb
r645731	aidan	2008-04-08 00:05:28 +0100 (Tue, 08 Apr 2008)	Update release notes
r645732	aidan	2008-04-08 00:07:50 +0100 (Tue, 08 Apr 2008)	Tag M2.1 RC3

r645733	aconway	2008-04-08 00:22:36 +0100 (Tue, 08 Apr 2008)	src/qpid/amqp_0_10/Body.h, Header.h: placeholders.
r645804	gsim	2008-04-08 10:18:10 +0100 (Tue, 08 Apr 2008)	Fixed compilation error from ignored qualifier on function return type.
r645826	gsim	2008-04-08 11:16:32 +0100 (Tue, 08 Apr 2008)	Removed out-of-date and misleading comment.
r645951	gsim	2008-04-08 15:48:25 +0100 (Tue, 08 Apr 2008)	QPID-903: changed read-write lock to mutex (currently recursive) to avoid deadlocking when adding bridge.
r646045	kpvdr	2008-04-08 20:29:08 +0100 (Tue, 08 Apr 2008)	Patch from Ted Ross: QPID-907: Management Improvements for C++ Broker and Store
r646054	aconway	2008-04-08 20:53:07 +0100 (Tue, 08 Apr 2008)	Summary: added 0-10 Array encoding and decoding.
r646071	aconway	2008-04-08 22:02:14 +0100 (Tue, 08 Apr 2008)	Touched existing template so new template allSegmentTypes.rb will be noticed.
r646093	cctrielloff	2008-04-08 22:51:17 +0100 (Tue, 08 Apr 2008)	QPID-908 from tross
r646099	aconway	2008-04-08 23:11:40 +0100 (Tue, 08 Apr 2008)	Fix packaing problem with generated file allSegmentTypes.h
r646113	aidan	2008-04-08 23:46:06 +0100 (Tue, 08 Apr 2008)	Nuke
r646114	aidan	2008-04-08 23:46:10 +0100 (Tue, 08 Apr 2008)	use svnexe for uploading, generate source jars
r646115	aidan	2008-04-08 23:46:26 +0100 (Tue, 08 Apr 2008)	Tag M2.1 RC3
r646376	aconway	2008-04-09 15:25:09 +0100 (Wed, 09 Apr 2008)	Improved diagnostics in allSegmentTypes test.
r646452	gsim	2008-04-09 18:59:38 +0100 (Wed, 09 Apr 2008)	Handle the set-redelivered flag on the final version of the message.release command.
r646505	gsim	2008-04-09 20:52:44 +0100 (Wed, 09 Apr 2008)	Fixes and automated tests for federation function.
r646519	rajith	2008-04-09 21:31:28 +0100 (Wed, 09 Apr 2008)	This is a fix for QPID-911. When the message id is set, _hasBeenUpdated will be set to true.

r646778	aconway	2008-04-10 13:36:58 +0100 (Thu, 10 Apr 2008)	Invocation handlers, see src/tests/amqp_0_10/handlers.cpp for example.
r646783	aconway	2008-04-10 14:00:04 +0100 (Thu, 10 Apr 2008)	Use "Exception" instead of typeid.name() as default exception name. Mangled type names are too confusing.
r646791	kpvdr	2008-04-10 14:17:44 +0100 (Thu, 10 Apr 2008)	Minor change to format of log message when exception is thrown
r646943	aconway	2008-04-10 21:15:08 +0100 (Thu, 10 Apr 2008)	amqp_0_10: Encoding for packed structs.
r647099	gsim	2008-04-11 11:02:49 +0100 (Fri, 11 Apr 2008)	QPID-913: committed patch from tross@redhat.com
r647123	gsim	2008-04-11 12:44:12 +0100 (Fri, 11 Apr 2008)	Set executable property for commands

- Attempt to achieve consensus on qpid-dev that all (C++) commits have a JIRA at the start
- AS to fix release notes, again
- r646519 should use updated(), check if QPID-911 affects M2.1, audit class for other usages of hasUpdated directly

Review of new JIRAs

Key	Component/s	Affects Version/s	Summary	Status	Assignee	Reporter
QPID-913	Python Client		Reorganization of python utilities + a bug fix	Open	Unassigned	Ted Ross
QPID-912	Java Client	M2,M2.1	AMQSession.getQueueDepth should better describe its failure conditions.	Open	Unassigned	Martin Ritchie
QPID-911	Java Client	M3	MessageID is not set for outgoing messages if that is the only property set	Resolved	Rajith Attapattu	Rajith Attapattu
QPID-910	Java Broker,Java Client	M2.1	Some poms reference inappropriate mvn repos	Open	Aidan Skinner	Aidan Skinner
QPID-909	Java Tests	M2.1	DurationTestDecorator doesn't work in conjunction with the ScaledTestDecorator	Open	Unassigned	Martin Ritchie
QPID-908	Python Client		Python command utility for federation routes	Closed	Unassigned	Ted Ross
QPID-907	C++ Broker,Python Client		Management Improvements for C++ Broker and Store	Closed	Unassigned	Ted Ross
QPID-906	Java Client	M2.1	java.lang.NumberFormatException silently ignored in MessageListener.onMessage(Message message)	Open	Unassigned	Alasdair MacLeod
QPID-905	Java Client		org.apache.qpid.client.ClientProperties dies ungracefully when max_prefetch is not set to a valid number	Open	Unassigned	Rafael H. Schlorning
QPID-904			broker.clean is not set properly in any of the test profiles	Open	Unassigned	Rafael H. Schlorning
QPID-903	C++ Broker		Federation (inter-broker) links cause C++ broker to hang	Closed	Unassigned	Ted Ross
QPID-902	C++ Broker,Python Client		Management Improvements for C++ Broker	Closed	Unassigned	Ted Ross
QPID-901	Java Client		update the java client to the 0-10 final spec	Open	Rafael H. Schlorning	Rafael H. Schlorning
QPID-900	Java Common	M2.1	[Java Common] AMQShortString should not create new byte[] backing when based off a heap buffer	Open	Rob Godfrey	Rob Godfrey

QPID-899	Java Common	M2.1	[Java Common] Bug in AMQShortString on tokenized substrings	Open	Rob Godfrey	Rob Godfrey
QPID-898	Ant Build System		remove .tar.bz2 from release target	Resolved	Unassigned	Nuno Santos
QPID-897	Java Tests	M3	Configurable receive timeout	Open	Unassigned	Arnaud Simon
QPID-896	Java Client		Provide Simple Pub/Sub examples that do not use extends	Open	Martin Ritchie	Martin Ritchie
QPID-895	Java Client		FailoverSingleServer delays on first connection.	Open	Unassigned	Martin Ritchie
QPID-894	Java Common		"Dependency on commons-lang: real need for 2.2, or can use 2.1?"	Open	Rafael H. Scholomig	Nuno Santos

- RS - new JIRA to moving try catch to just around onMessage()
- RS - Take QPID-906 and check that it is JMS Spec compliant
- CT - find cruise control files
- CT - JIRA time tracking is broken, normalises 1d is 24h should be 8h

Update on GSoC projects

We discussed our project proposals

Qpid Java Meeting Minutes 28-03-2008

Attendees

Rob Godfrey
 Arnaud Simon
 Aidan Skinner
 Marnie McCormack
 Rafi Scholomig

Update on M2.1 release process

Tagged RC1 yesterday
 Going to run RAT, push out over the weekend
 Need to schedule updates to documentation

Update on GSoC projects

Had a large number of students, need to get mentors
 Need to distribute the load

Also have interest from external parties helping with management bridge

Need to follow the timeline; process

Apache Conn Europe

Qpid has a 15 min slot Amsterdam, Wednesday 9th April. Any volunteers?

Review of the previous week's commits

<http://markmail.org/search/?q=type%3Acheckins+list%3Aorg.apache.incubator.qpid-commits+order%3Adate-forward+date%3A200803+#que>
 r639619 : Rafi comments that there is a base class on trunk which should close all connections
 r640117 : We should standardize the comments on checkin to have QPID-xxx first, ensure case is correct
 r640422 : Should make timeouts a configuration parameter rather than picking arbitrary values; also allow non-timeout path coverage;
 r640434 : Need to look at logging levels for M3
 r640503 : no JIRA number associated with the checkin!
 r641212 : no need to be so generic, only need 32 bit serial numbers
 r641232 : incorrect capitalization

Review newly raised JIRAs

<https://issues.apache.org/jira/secure/IssueNavigator.jspa?reset=true&&type=-2&pid=12310520&created%3Aprevious=-1w+3d&sorter/field=id>

Update on what people are currently working on

MM: Looking at priorities for M3

AS: M2.1 and Merge

RG: Broker Refactoring
Headers/Exchange

ASimon: Adding DTX tests

See QPID-884

Fix the URL for backwards compatibility - check version in protocol initiation

Configuration for prefetch in 0-10

RS: Updating 0-10 client to use correct 0-10 spec
large commit next week

Update on the merge from M2.1 to trunk (Aidan)

Will resynch broker / client after M2.1 finalised

Testing

Marnie will put up test spec

Need to review to make sure all use cases covered.

Need stress tests and soak tests

Arnaud - should look at the Sonic test suite

Meeting closed 15:30 UK

Qpid Java Meeting Minutes 2008 05 02

revision	committer	date	comment
r649547	aconway	2008-04-18	Uncommented tests.
r649554	aconway	2008-04-18	From Ted Ross: QPID-934 " title="Visit page outside Confluence" rel="nofollow"linktype="raw" linktext="https://issues.apache.org/jira/browse/QPID-934 "> QPID-934 " title="Visit page outside Confluence" rel="nofollow"linktype="raw" linktext="https://issues.apache.org/jira/browse/QPID-934 "> QPID-934 ">https://issues.apache.org/jira/browse/QPID-934 This patch fixes a problem related to multiple management sessions run over the same AMQP session (typically seen in test environments). ...
r649571	aconway	2008-04-18	Fix build problem.
r649585	aidan	2008-04-18	Merged revisions 648217-649481 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/trunk r648272 - aconway - 2008-04-15 15:54:46 +0100 (Tue, 15 Apr 2008) - 1 line Fix b ...
r649642	kpvdr	2008-04-18	fix ambiguity problem found on gcc 3.4 compilers
r649648	rhs	2008-04-18	QPID-901 : update pom to work with new codegen
r649661	aidan	2008-04-18	Merged revisions 649482-649660 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/trunk r649547 - aconway - 2008-04-18 15:12:36 +0100 (Fri, 18 Apr 2008) - 2 lines Unco ...
r649666	astitcher	2008-04-18	Split AsynchIOAcceptor into IOHandler and connection control code
r649671	aconway	2008-04-18	Fix test failure.
r649689	astitcher	2008-04-18	Refactored Acceptor code to allow multiple acceptors to be present in the broker
r649691	astitcher	2008-04-18	Added missed new include file
r649711	aidan	2008-04-18	QPID-832 fix some compile errors
r649790	aidan	2008-04-19	QPID-832 ahem
r649905	aidan	2008-04-20	QPID-832 now 95% bug free
r649907	aidan	2008-04-20	QPID-832 remove log4j spewage
r649909	aidan	2008-04-20	Merged revisions 649661-649908 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/trunk r649666 - astitcher - 2008-04-18 20:44:25 +0100 (Fri, 18 Apr 2008) - 2 lines Sp ...
r649915	gsim	2008-04-20	QPID-920 : converted c++ client to use final 0-10 protocol * connection handler converted to using invoker & proxy and updated to final method defs * SessionCore & ExecutionHandler replace by SessionIm ...
r650099	aidan	2008-04-21	QPID-832 Make sure that we lock early enough to avoid deadlocks when closing
r650108	aidan	2008-04-21	QPID-832 revert last commit, 650099

r650122	aidan	2008-04-21	QPID-832 pass a genuine connection so that it doesn't NPE
r650127	arnaudsimon	2008-04-21	QPID-939 Added "ID:" to message ID
r650136	rgodfrey	2008-04-21	
r650145	rgodfrey	2008-04-21	Wrong revision
r650146	rgodfrey	2008-04-21	create branch for broker refactoring
r650148	rgodfrey	2008-04-21	Initial checkpoint of queue refactoring work
r650159	gsim	2008-04-21	QPID-920 : send message-accept for acks (as well as completion) * AckPolicy now maintains a set of transferred messages for cumulative accepts
r650179	rgodfrey	2008-04-21	remove duplicate check of interest in enqueue, enable new Queue by default
r650193	aidan	2008-04-21	QPID-832 catch dodgy mock generated exception, just like M2.x
r650198	aconway	2008-04-21	src/qpid/RangeSet.h: generic set implementation using ranges. - no heap allocation for simple sets (<= 3 ranges) - binary searches for $O(\log n)$ performance in complex sets
r650205	aidan	2008-04-21	QPID-832 handle multiple brokers properly
r650210	aconway	2008-04-21	Fix compile error on rhel5.
r650221	aconway	2008-04-21	Better workaround for boost::ptr_map incompatibility between boost 1.33 and 1.34, based on public properties of ptr::map types rather than version numbers.
r650227	aconway	2008-04-21	Disable compilation of amqp_0_10 codec until ready for integration.
r650250	gsim	2008-04-21	<ul style="list-style-type: none"> raise error when controls other than attached are received on unattached channel * corrected exception handling in client and on broker (broker to issue detach)
r650273	aconway	2008-04-21	Fix packaging problems for rpmbuild.
r650359	rhs	2008-04-22	QPID-832 : fixed ant build system
r650439	gsim	2008-04-22	QPID-648 : (based on patch from mfarrellee@redhat.com) * apply authentication to final 0-10 codepath * consolidate conditional compilation of sasl-related code * improved handling of connection close ...
r650440	rhs	2008-04-22	QPID-823 : add junit to the libs for junit-toolkit
r650447	aidan	2008-04-22	create forrest site
r650450	gsim	2008-04-22	QPID-944 : do no-local checking where requested when there is an exclusive subscription active
r650478	aidan	2008-04-22	Merged revisions 649909-650455 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/trunk r649915 - gsim - 2008-04-20 13:10:37 +0100 (Sun, 20 Apr 2008) - 6 lines QPID-92 ...
r650565	rhs	2008-04-22	QPID-947 : update cpp and python management to 0-10 final
r650579	rhs	2008-04-22	QPID-832 : updated build order
r650581	rhs	2008-04-22	QPID-832 : moved 0-8 specific code into 0-8 subclass of session
r650598	rhs	2008-04-22	QPID-832 : moved more 0-8 specific code to 0-8 subclasses
r650604	rhs	2008-04-22	QPID-948 : patch from Ted Ross for updated management utilities to 0-10 final
r650617	rhs	2008-04-22	QPID-832 : moved more 0-8 specific code into 0-8 subclasses
r650620	cctrielloff	2008-04-22	QPID-945 from Ted Ross
r650635	gsim	2008-04-22	Moved federation to final 0-10 codepath
r650640	gsim	2008-04-22	QPID-920 : allow applications to trigger the sending of a flush to server
r650657	astitcher	2008-04-22	<ul style="list-style-type: none"> Renamed the Acceptor class to be the ProtocolFactory class which better approximates its current behaviour * Slightly refactored TCPIOPlugin to better approximate how it would look when we implement ...
r650658	aidan	2008-04-22	QPID-832 make tests use new QpidTestCase magic for getting connections

r650795	gsim	2008-04-23	Added to the no-local tests, cleaned up comments (and highlighted non-standard nature of these tests)
r650813	aidan	2008-04-23	QPID-832 remember to pass username to getConnection
r650850	rgodfrey	2008-04-23	QPID-832 : Quoted identifiers grammar fix for client side selectors
r650855	rhs	2008-04-23	QPID-832 : switched from execing javacc to using the javacc task; this should fix the build on cygwin
r650858	rhs	2008-04-23	QPID-832 : moved CloseTests and DurableSubscriberTests -> CloseTest and DurableSubscriberTest
r650875	gsim	2008-04-23	Add support for reading 0-10 arrays; Set sync bit on session header for commands sent with auto_sync on.
r650876	rgodfrey	2008-04-23	QPID-832 : connection should be set to started before sessions are started
r650887	rgodfrey	2008-04-23	QPID-832 : Fixed AMQSession_0_10 so that it takes the acknowledge mode from the session not a system definition
r650890	aidan	2008-04-23	QPID-832 fix failover detection, rename startDispatcher
r650901	rhs	2008-04-23	QPID-832 : fixed override of notifyMessage, this re-enables selectors for 0-10
r650906	aconway	2008-04-23	<ul style="list-style-type: none"> SequenceSet implemented on RangeSet. - Reduced #include dependencides on SequenceSet.h
r650922	astitcher	2008-04-23	Make python tests work with VPATH builds
r650970	astitcher	2008-04-23	Patch from Mick Goulish: Fixes to previous improved portability patch
r650997	aconway	2008-04-23	src/tests/ClientSessionTest.cpp: uncommented tests for session resume as EXPECTED_FAILURES tests. src/tests/unit_test.h: workarounds for broken EXPECTED_FAILURES tests in boost <= 1.34
r650999	astitcher	2008-04-23	Reverted earlier change to valgrind suppressions
r651088	aconway	2008-04-23	Fix build error introduced by earlier commit.
r651107	aidan	2008-04-23	QPID-832 copy trunk before performing some major operations
r651111	aidan	2008-04-23	Delete stuff that's just going to get synced from M2.x
r651112	aidan	2008-04-23	QPID-832 copy the M2.x broker
r651113	aidan	2008-04-23	QPID-832 sync from M2.x
r651115	aidan	2008-04-24	QPID-832 copy from M2.x
r651116	aidan	2008-04-24	QPID-832 copy from M2.x
r651117	aidan	2008-04-24	QPID-832 copy from M2.x
r651118	aidan	2008-04-24	QPID-832 copy from M2.x
r651119	aidan	2008-04-24	QPID-832 copy from M2.x
r651120	aidan	2008-04-24	QPID-832 copy from M2.x
r651124	aidan	2008-04-24	QPID-832 sync build stuff from thegreatmerge
r651125	aidan	2008-04-24	QPID-832 sync build stuff from thegreatmerge
r651126	aidan	2008-04-24	QPID-832 sync build stuff from thegreatmerge
r651133	aidan	2008-04-24	QPID-832 merge M2.x
r651134	aidan	2008-04-24	QPID-832 nuke some obsolete stuff
r651211	aidan	2008-04-24	QPID-832 update pom version
r651276	rhs	2008-04-24	QPID-832 : fixed DerbyMessageStore to compile on Java 1.5 (hopefully)
r651287	aconway	2008-04-24	emacs/qpid-c+-mode.el: qpid-c+-mode for Emacs. qpid-style indentation plus some useful commands for inserting copyrights etc. Feel free to improve it, there's lots of room.
r651290	cctrielloff	2008-04-24	QPID-953 from tross
r651301	aidan	2008-04-24	QPID-832 make systests run

r651321	aconway	2008-04-24	Edits to doxygen comments for user documentation.
r651325	rgodfrey	2008-04-24	QPID-832 : Fix eol-style
r651346	nsantos	2008-04-24	add include for list, for gcc4.3
r651375	rgodfrey	2008-04-24	QPID-832 : Fix CRLF line endings
r651421	gsim	2008-04-24	Correct expected error codes for final 0-10 spec
r651423	gsim	2008-04-24	Generate c++ code from final 0-10 spec
r651425	cctrielloff	2008-04-24	QPID-958 by Danushka Menikkumbura
r651431	rhs	2008-04-24	QPID-832 : fixed merge artifact affecting replyTo
r651474	astitcher	2008-04-25	Fix building examples when using a VPATH build
r651531	gsim	2008-04-25	Fixed caught exception type in recovery
r651997	aconway	2008-04-27	Session state as per AMQP 0-10 specification.
r652037	cctrielloff	2008-04-28	QPID-918 patch from Senaka
r652041	cctrielloff	2008-04-28	QPID-971 - Senaka Fernando, I also edited the README in a few places.
r652053	astitcher	2008-04-28	Work In Progress: Added initial rdma code including test server and client Turn off rdma support by default but autoconf should now detect whether necessary rdma/ibverbs libs and headers are pre ...
r652075	gsim	2008-04-29	QPID-977 : shutdown mgmt client cleanly in federation tests (patch from tross@redhat.com) QPID-981 : added custom options to queue declare to tag each message as it goes through a bridge queue and allow ...
r652076	gsim	2008-04-29	QPID-981 : management schema change missed from previous commit
r652083	gsim	2008-04-29	QPID-974 : allow the size of the queue of outgoing frames to be restricted QPID-544 : tidy up configuration (ensuring desired settings are used correctly, allowing tcp socket options to be se ...
r652086	rhs	2008-04-29	QPID-979 : added backwards compatible uuid to qpid.datatypes
r652114	gsim	2008-04-29	QPID-981 : allow id and exclude list to be passed through when creating a bridge with qpid-route
r652120	astitcher	2008-04-29	Removed some unnecessary #includes
r652170	rhs	2008-04-29	QPID-984 : override MINA's IoServiceListenerSupport class in order to fix infinite loop
r652173	rhs	2008-04-29	QPID-983 : fixed deadlock between AMQConnection.close and FailoverHandler
r652180	astitcher	2008-04-29	More RDMA Work in Progress Changes to client buffering Buffering improvement to server Removed unused state machine from RdmaIO code Move the write throttling due to limited write buff ...
r652386	gsim	2008-04-30	QPID-988 and QPID-989 : fixes to framing for final 0-10 spec
r652388	ritchiem	2008-04-30	QPID-889 : Removed _reapingStoreContext from CSDM replaced with local StoreContext()s so they are not reused by different threads.
r652389	ritchiem	2008-04-30	QPID-887 : Renamed QueueHouseKeeping threads so they can be identified in thread dump. Named Queue-housekeeping-<virtualhost name>
r652399	ritchiem	2008-04-30	QPID-888 , QPID-886 : Fixed all management uses of _lock.lock / _lock.unlock so that they correctly call unlock from a finally block in the CSDM. There are two issues that cover that. QPID-888 - Fix the ...
r652409	rajith	2008-05-01	This commit is for QPID-992 and QPID-993 build.xml ===== I added the broker-plugins as a module to the ant build system. This was nessacery to get the plugins jar generated for QPID-993 . In gene ...
r652411	rajith	2008-05-01	Added the following tests to 0-10 exclude lists as these tests are for the java broker. However when the java broker gets to 0-10 we should also have exclude lists per broker as well. For the time bei ...
r652451	gsim	2008-05-01	Cleanup: Re-enable tests that now pass; delete unused templates directory.
r652469	gsim	2008-05-01	QPID-966 : applied patch from rajith; altered to use uuid as session name; updated verify scripts for automated testing; re-enabled automated testing in c++ build
r652506	gsim	2008-05-01	Remove preview tests (no longer required)

r652535	nsantos	2008-05-01	applying Ted Ross's patch to handle unicode encoding and management message ordering issues					
r652548	rhs	2008-05-01	QPID-965 : added an option to turn on deprecation during compile					
r652558	gsim	2008-05-01	QPID-989 : fix decode of zero sized map					
r652567	aidan	2008-05-01	QPID-994 Dont wait for attain state as connection is closed by we get CloseOk					
r652568	aidan	2008-05-01	QPID-1001 dont set the expiration time if TTL is 0					
r652591	gsim	2008-05-01	Turn auth back on by default for c++ broker (only if SASL libs are available)					
r652670	rhs	2008-05-01	QPID-987 : reduced message count in DupsOKTest					
r652672	rhs	2008-05-01	QPID-993 : added an osgi manifest to broker-plugins jar					
r652680	rhs	2008-05-01	QPID-1002 : applied patch from Senaka to make systests run using normal junit testrunner					
r652689	gsim	2008-05-01	Boost's string split function causes problems on older versions of the library. Replaced with homegrown equivalent.					
r652705	rhs	2008-05-01	QPID-1003 : added excludes for framework test classes					
r652779	gsim	2008-05-02	QPID-986 : Patch from Danushka Menikkumbura.					
r652783	gsim	2008-05-02	QPID-980 : Patch from Danushka Menikkumbura revising installation notes.					

JIRAs

Key	Component(s)	Affects Version/s	Summary	Status	Assignee	Reporter	Review Commencement
QPID-965	Ant Build System	M3	(QPID-965) ant build system does not completely replace maven	Open	Unassigned	Aidan Skinner	
QPID-966	Python Client	M3	(QPID-966) Update python examples to use the new API	Open	Rajith Attapattu	Rajith Attapattu	
QPID-967	Java Client		(QPID-967) Binding arguments in Java JMS	Open	Unassigned	Jonathan Robie	
QPID-968	Java Client		(QPID-968) runSamples script uses incorrect log4j.xml file location	Open	Unassigned	Suran Jayathilaka	
QPID-969		M2	(QPID-969) Qpid/C++ M2 build failure in man file generation	Open	Unassigned	Danushka Menikkumbura	
QPID-970	Java Management Console	M3	(QPID-970) Wrong url in README file of Eclipse-plugin	Open	Unassigned	Lahiru Gunathilake	
QPID-971	C++ Client		(QPID-971) README for C++ examples	Open	Unassigned	Senaka Fernando	
QPID-972	Java Broker, Java Client		(QPID-972) Add runSamples.bat script for running samples in Windows	Open	Unassigned	Suran Jayathilaka	AS or MI to review patch
QPID-974	C++ Client	M3	(QPID-974) Need to allow the size of the queue of outgoing frames to be limited	Resolved	Gordon Sim	Gordon Sim	
QPID-975	Dot Net Client		(QPID-975) Prefetched messages can cause problems with clients	Open	Unassigned	Martin Ritchie	
QPID-976	C++ Client	M3	(QPID-976) Add wider SASL support to c++ client	Open	Unassigned	Gordon Sim	
QPID-977	C++ Broker	M3	(QPID-977) Federation tests raise exceptions after successful completion	Closed	Gordon Sim	Ted Ross	
QPID-978	C++ Broker	M2, M2.1	(QPID-978) memory leak in C++ broker	Open	Gordon Sim	Rudolf Polzer	
QPID-979	Python Client	M3	(QPID-979) Improvements for the python client API	Open	Rafael H. Schloeming	Rajith Attapattu	
QPID-980		M2	(QPID-980) Amend Qpid/C++ INSTALL file to show how to build/install Boost to work with the current build system	Closed	Unassigned	Danushka Menikkumbura	

QPID-981	C++ Broker	M3	(QPID-981) Prevent looping in inter-broker routing	Resolved	Gordon Sim	Gordon Sim	
QPID-982	C++ Broker, C++ Client, Code Generator		(QPID-982) C++ rubygen code generator sensitive to ruby version.	Open	Unassigned	Alan Conway	
QPID-983	Java Client	M3	(QPID-983) Deadlock between AMQConnection.close and FailoverHandler	Resolved	Unassigned	Rafael H. Schloeming	Want to check fo M2.x
QPID-984	Java Client	M3	(QPID-984) FailoverTest hangs	Open	Unassigned	Rafael H. Schloeming	
QPID-985			(QPID-985) Build break in C++ repo	Closed	Unassigned	Danushka Menikkumbura	
QPID-986	C++ Broker, C++ Client		(QPID-986) Need to link libuuid.a to libqpidbroker.so and libqpidclient.so in Qpid/C++	Closed	Unassigned	Danushka Menikkumbura	
QPID-987	Java Tests		(QPID-987) org.apache.qpid.test.client.DupsOkTest has high memory requirements	Resolved	Unassigned	Senaka Fernando	
QPID-988	C++ Broker, C++ Client, Java Client, Python Client	M3	(QPID-988) Final 0-10 spec does not include a frame-end marker between frames	Resolved	Gordon Sim	Gordon Sim	
QPID-989	C++ Broker, C++ Client, Java Client, Python Client	M3	(QPID-989) Final 0-10 spec has a count filed in maps	Resolved	Gordon Sim	Gordon Sim	
QPID-990	C++ Broker	M3	(QPID-990) Federation objects need to be persistent	Open	Unassigned	Ted Ross	
QPID-991	Java Tools		(QPID-991) Qpid Junit Toolkit Ant Plugin (Ant Task)	Open	Martin Ritchie	Senaka Fernando	
QPID-992	Java Tests	M3	(QPID-992) SimpleACLTTest fails in trunk	Open	Rajith Attapattu	Rajith Attapattu	Test nee to be reviewed suspect time.
QPID-993	Java Tests	M3	(QPID-993) org.apache.qpid.server.plugins.PluginTest is failing on trunk	Resolved	Rajith Attapattu	Rajith Attapattu	
QPID-994	Java Tests		(QPID-994) Reduce Memory consumed by DupsOkTest	Open	Unassigned	Senaka Fernando	
QPID-995	Dot Net Client	M2.1, M3	(QPID-995) StateWaiter can get terribly confused	Open	Aidan Skinner	Aidan Skinner	
QPID-996	Dot Net Client	M2.1, M3	(QPID-996) .Net client SslConnectionTest uses odd port	Open	Aidan Skinner	Aidan Skinner	
QPID-997	Java Tests		(QPID-997) AMQBrokerManagerMBeanTest takes more than 60seconds to run	Open	Unassigned	Martin Ritchie	
QPID-998	Ant Build System		(QPID-998) Prevent warnings about missing bin / etc directories	Open	Martin Ritchie	Martin Ritchie	
QPID-999	Java Client		(QPID-999) QPID-854 : Commit 637086 on M2.1 is missing from trunk merge	Open	Unassigned	Martin Ritchie	
QPID-1000	Java Tests		(QPID-1000) Deadlock in unit.basic.SelectorTest	Open	Unassigned	Martin Ritchie	Martin to svn annotate and take appropriate action.
QPID-1001	Dot Net Client	M2.1, M3	(QPID-1001) .Net client sets expiration time incorrectly	Open	Aidan Skinner	Aidan Skinner	

QPID-1002	Java Tests		(QPID-1002) Improvements to Test Framework making it possible to run systests	Closed	Martin Ritchie	Senaka Fernando	MR to commen
QPID-1003	Ant Build System, Java Tools		(QPID-1003) Junit treats Classes ending with Test as testcases	Closed	Martin Ritchie	Senaka Fernando	
QPID-1004	Java Tests		(QPID-1004) org.apache.qpid.test.testcases.TTLTest Fails	Open	Unassigned	Senaka Fernando	RHS to explain problem Jira
QPID-1005	Java Client	M3	(QPID-1005) Client ID	Open	Unassigned	Arnaud Simon	
QPID-1006	Java Client	M3	(QPID-1006) (0.10 code path) Message rate is very slow when big messages are sent.	Open	Unassigned	Arnaud Simon	
QPID-1007	Java Client	M3	(QPID-1007) (0.10 code path) Need a way of controlling MINA queue size	Open	Unassigned	Arnaud Simon	

AOB: rhs to get permissions to edit wiki

mar to comment on 1002

Qpid Java Meeting Minutes 2008-04-18

Agenda

- Update on Merge
- Update on M2.1 release process
- Review of code commits
- Review of new JIRAs
- Update on GSoC projects

Attendees

Aidan Skinner
 Arnaud Simon
 Carl Trieloff
 Martin Ritchie
 Robert Godfrey

Apologies

Gordon Sim

Update on M2.1 release process

Review of code commits

Revision	Committer	Date	Commit Comment	Review Points
r647227	rhs	2008-04-11	QPID-901 : temporary workaround for AMQP-218	
r647270	kpvdr	2008-04-11	Patch from Ted Ross: added set methods to hilo types in generated management classes	
r647616	aidan	2008-04-13	QPID-916 Correct release notes	
r647620	aidan	2008-04-13	QPID-916 tag RC4	
r647704	gsm	2008-04-14	QPID-917 : Use PLAIN (rather than the non-standard AMQPLAIN) as the SASL mechanism when authenticating python test clients.	
r647710	gsm	2008-04-14	Correction to release notes (now support AMQP 0-9)	
r647716	gsm	2008-04-14	QPID-648 : Initial support for sasl authentication for c++ broker. From patch submitted by mfarrellee@redhat.com. Authentication is optional at compile time (based on user selection or availability of ...)	
r647726	aidan	2008-04-14	QPID-832 delete for resync with trunk (cpp, ruby) and M2.1 (java/broker)	
r647727	aidan	2008-04-14	QPID-832 sync cpp with trunk	

r647728	aidan	2008-04-14	QPID-832 sync ruby from trunk	
r647730	aidan	2008-04-14	QPID-832 sync broker with M2.1	
r647800	gsm	2008-04-14	QPID-648 : keep the sasl_conn member in the handler to avoid the need for friend declaration	
r647825	aidan	2008-04-14	QPID-916 Tag RC5, with correct C++ release notes	
r647887	rhs	2008-04-14	fixed encode/decode of structs in command/control arguments to include the type code when specified	
r647903	gsm	2008-04-14	Use the errata file for final 0-10 that has a type code for xids without which dtx.recover can't work. Return the indoubt xids as an array of struct32s each of which contains an encoded xid.	
r647937	aconway	2008-04-14	https://bugzilla.redhat.com/show_bug.cgi?id=441080 from Ville Skyttä (ville.skytta@iki.fi) qpidc's build does not use \$RPM_OPT_FLAGS so it misses some compiler security features, and strips installed ...	
r647940	gsm	2008-04-14	QPID-648 : more flexible sasl implementation (patch provided by mfarrellee@redhat.com)	
r647990	gsm	2008-04-14	Fix to struct32 encoding	
r647999	gsm	2008-04-14	<ul style="list-style-type: none"> • Fix interpretation of accept-mode, 0 == EXPLICIT * Ensure accepts are taken into account in command sequence 	
r648013	nsantos	2008-04-14	fix home dir permissions	
r648095	aconway	2008-04-15	Struct32 encoding	
r648194	gsm	2008-04-15	Remove deleted file from distribution list.	
r648196	gsm	2008-04-15	QPID-648 : Get list of supported mechanisms from sasl lib. (Patch from mfarrellee@redhat.com)	
r648207	aidan	2008-04-15	Merged revisions 631979-647797 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/trunk r631987 - aconway - 2008-02-28 14:47:59 +0000 (Thu, 28 Feb 2008) - 2 lines Fixe ...	
r648216	aidan	2008-04-15	Merged revisions 627417-648207 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/branches/M2.1 r627552 - rgodfrey - 2008-02-13 18:10:53 +0000 (Wed, 13 Feb 2008) - 1 line ...	
r648218	aidan	2008-04-15	Merged revisions 647798-648216 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/trunk r647800 - gsm - 2008-04-14 14:57:36 +0100 (Mon, 14 Apr 2008) - 3 lines QPID-64 ...	
r648272	aconway	2008-04-15	Fix build error: MapValue SIZE was too small for Struct32.	
r648288	astitcher	2008-04-15	Refactored the IO framework that sits on top of Poller so that it uses a generalised IOHandle. This means that you can define new classes derived from IOHandle (other than Socket) that can also be add ...	
r648292	nsantos	2008-04-15	add svn revision include to specfile	
r648297	aconway	2008-04-15	Cleanup of size calculations and handling UnknownStruct	
r648308	nsantos	2008-04-15	QPID-921 : applied qpid-patch36.diff on behalf of Ted Ross	
r648314	aidan	2008-04-15	QPID-831 fix pom version	
r648328	aidan	2008-04-15	QPID-831 Remove some broker introspection stuff and transaction bumpf	
r648329	aconway	2008-04-15	Disabled failing tests, working on fixing the issues.	
r648338	aconway	2008-04-15	Comment out failing test, repairing.	

r648362	aconway	2008-04-15	Correct Struct32 encoding: size/code/data. Proper re-encoding for unknown struct codes.	
r648418	aconway	2008-04-15	Fix build error - missing op << for Struct32.	
r648614	rgodfrey	2008-04-16	QPID-899 : Bug in AMQShortString on tokenized substrings	
r648617	rgodfrey	2008-04-16	QPID-900 : Re-use byte[] of buffer for backing of AMQShortString	
r648637	rgodfrey	2008-04-16	QPID-922 : Selectors on header properties should not convert AMQShortStrings to Strings on every evaluation	
r648639	aidan	2008-04-16	QPID-916 add DISCLAIMER to top-level, make sure C++ and .Net archives are named correctly	
r648642	aidan	2008-04-16	QPID-916 tag RC6	
r648645	rgodfrey	2008-04-16	QPID-925 : Only begin store transactions when there is a persistent message to be committed	
r648648	rgodfrey	2008-04-16	QPID-926 : Perform all store operations associated with an acknowledge in a single store transaction	
r648652	rgodfrey	2008-04-16	QPID-927 : Multiple acknowledgements should be coalesced into single multiple ack	
r648658	rgodfrey	2008-04-16	QPID-929 : Exchange.Declare being sent prior to every message when publishing to explicit destination	
r648661	arnaudsimon	2008-04-16	QPID-928 Added a pause period for letting the finalizer a chance to notify all the Mina connector threads before we check for spurious threads.	
r648666	rgodfrey	2008-04-16	QPID-931 : Always use exclusive consumers when subscribing to topics	
r648669	rgodfrey	2008-04-16	QPID-932 : Remove references to unused constructor argument "browsedAcks" of NonTransactionalContext	
r648670	rgodfrey	2008-04-16	QPID-932 : Remove references to unused constructor argument "browsedAcks" of NonTransactionalContext	
r648672	rgodfrey	2008-04-16	QPID-933 : performance tweaks	
r648678	aidan	2008-04-16	QPID-916 update dotnet NOTICE to include log4net, make sure DISCLAIMER is including in c++	
r648679	aidan	2008-04-16	QPID-916 delete, about to recreate	
r648680	aidan	2008-04-16	QPID-916 tag RC6	
r648681	arnaudsimon	2008-04-16	QPID-897 : this test was intermittently timing out when messages are not prefetched. This is a temporary fix until we use a configurable timeout.	
r648692	rhs	2008-04-16	QPID-901 : updates to the java client to use the 0-10 final spec instead of the 0-10 preview spec; this includes improvements to the codegen process as well as some modifications to the shared code pat ...	
r648699	rhs	2008-04-16	QPID-901 : add back ExceptionHelper.java to fix the build	
r648706	aconway	2008-04-16	Fix bug in Blob::assign assigning from an empty blob.	
r648724	aconway	2008-04-16	Fix encoding for sized structs.	
r648726	aconway	2008-04-16	Separate new codec from libqpidcommon to improve link times. To be included in libqpidcommon when we are ready to replace framing codec.	
r648735	aidan	2008-04-16	QPID-832 fix exception constructors	
r648740	ritchiem	2008-04-16	QPID-886 : In order to allow the test to be written that highlights the failure we need to be able to provide a Config object not a file. So made the method public that reads the file and added constr ...	
r648764	aidan	2008-04-16	Rename in case of further M2-based releases	
r648770	aconway	2008-04-16	Removed complex_types.h from Makefile.am.	
r648771	rhs	2008-04-16	QPID-901 : don't depend on constant values matching up when converting between JMS and AMQP delivery modes	
r648782	aconway	2008-04-16	Add missing files to packaging.	

r648784	rhs	2008-04-16	QPID-901 : updated the JMS examples to use legal delivery mode values as they are now checked with the 0-10 final updates	
r648834	rgodfrey	2008-04-16	QPID-156 : Add an Apache licensed store - created an experimental Derby based store	
r648904	astitcher	2008-04-17	Refactored IO Thread creation so that it happens in the Broker class - There is now a single Poller created by the Broker class that is passed to the Acceptor for use in network IO. It can also now ...	
r649016	arnaudsimon	2008-04-17	QPID-919 Changed AMQBrokerDetails to throw an URL exception when the port number is not specified.	
r649055	aidan	2008-04-17	Track changes from M2.x now	
r649057	aidan	2008-04-17	Track changes from M2.x now	
r649058	aidan	2008-04-17	Track changes from M2.x now, with right svn	
r649059	gsim	2008-04-17	Some fixes to the transitional spec defs.	
r649070	arnaudsimon	2008-04-17	QPID-796 Made connection URL property + use session level method	
r649099	arnaudsimon	2008-04-17	QPID-884 Added cpp profile that does not use a store. Also updated profile for taking auth into account and updated broker.clean as per QPID-904	
r649126	nsantos	2008-04-17	add full path to qpidd in init script, as it fails in some environments with just the command name	
r649130	aconway	2008-04-17	Added missing .h files to Makefile.am to fix make rpmbuild. Add non-const Message::data() Make log/Statement.h public.	
r649132	aidan	2008-04-17	Merged revisions 632071-649058 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/branches/M2.x r632363 - ritchiem - 2008-02-29 16:00:19 +0000 (Fri, 29 Feb 2008) - 1 line ...	
r649141	aidan	2008-04-17	QPID-831 fix compile errors	
r649159	aconway	2008-04-17	src/Makefile.am: Fix problems with rpmbuild. src/tests/README: Updated information about boost test.	
r649294	astitcher	2008-04-17	Patch for improved compatibility with gcc 3.4 and boost 1.33	
r649339	aconway	2008-04-18	src/tests/python_tests: fix exit status if QPID_NO_PREVIEW is set. src/qpid/framing/AMQFrame.h: frame header setters/getters with 0-10 naming.	
r649409	arnaudsimon	2008-04-18	QPID-798 Make property per connection and/or system wide; change name to SYNC_PERSISTENT to denote only used on persistent messages	Add accessor to Qpid test Case to allow prefetch to be turned on / off
r649432	aidan	2008-04-18	QPID-832 default protocol version to 0-9	
r649436	arnaudsimon	2008-04-18	QPID-936 : added missing session close op	
r649479	kpvdr	2008-04-18	Fix to prevent possible Timer deadlocks by holding onto mutex while calling fire()	

Review of New JIRAs

Key	Component(s)	Affects Version/s	Summary	Status	Assignee	Reported By
QPID-914	Python Client	M3	(QPID-914) Separate python management component from core python client	Open	Rafael H. Schlorning	Gordon Sim
QPID-915	C++ Broker		(QPID-915) A list of ways to avoid c++ and Python problems on the RHEL4 platform This is HTML, i.e. for a Wiki page.	Closed	Unassigned	michael goulia
QPID-916		M2.1	(QPID-916) Release M2.1	Open	Aidan Skinner	Aidan Skinner
QPID-917	Python Client	M1, M2, M2.1, M3	(QPID-917) python tests use the default, non-standard 'AMQPLAIN' as SASL mechanism	Resolved	Gordon Sim	Gordon Sim

QPID-918	C++ Broker	M3	(QPID-918) Authentication password is logged when "trace" is enabled	Open	Unassigned	Ted Simc
QPID-919	Java Client	M3	(QPID-919) Wrong exception thrown when the connection URL contains a broker name followed by ":" but no port number is specified.	Open	Arnaud Simon	Arnaud Simon
QPID-920	C++ Client	M3	(QPID-920) Convert c++ client to final 0-10 spec	Open	Gordon Sim	Gordon Sim
QPID-921	C++ Broker, Python Client	M3	(QPID-921) Management: persistent object-ids, SASL support, System-ID	Closed	Unassigned	Ted Simc
QPID-922	Java Broker	M2.1	(QPID-922) (Java Broker) Selectors comparing header properties to constants always convert the property from AMQShortString to String, rather than converting the constant once	Resolved	Rob Godfrey	Rob Godfrey
QPID-923	Python Test Suite	M2.1	(QPID-923) Python tests are not spec version aware	Open	Unassigned	Marti Ritch
QPID-924	Ruby Test Suite	M2.1	(QPID-924) Ruby test.rb use of '0.0.0.0' for localhost doesn't work under cygwin	Open	Unassigned	Marti Ritch
QPID-925	Java Broker	M2.1	(QPID-925) (Java Broker) Only begin store transactions when there is a persistent message to be committed	Resolved	Rob Godfrey	Rob Godfrey
QPID-926	Java Broker	M2.1	(QPID-926) (Java Broker) Perform all store operations associated with an acknowledgement in a single store transaction	Open	Rob Godfrey	Rob Godfrey
QPID-927		M2.1	(QPID-927) (Java Client) Multiple acknowledgements should be coalesced into single multiple ack, rather than each being sent individually	Resolved	Rob Godfrey	Rob Godfrey
QPID-928	Java Tests	M3	(QPID-928) Test org.apache.qpid.test.unit.client.connection.ConnectionCloseTest intermittently failing	Open	Unassigned	Arnaud Simon
QPID-929	Java Client		(QPID-929) (Java Client) Exchange.Declare being sent prior to every message when publishing to explicit destination	Resolved	Rob Godfrey	Rob Godfrey
QPID-930	Ruby Client	M1, M2, M2.1, M3	(QPID-930) ruby channel_close test issues	Open	Unassigned	Gordon Sim
QPID-931	Java Client	M2.1	(QPID-931) (Java Client) Always use exclusive consumers when subscribing to topics	Resolved	Rob Godfrey	Rob Godfrey
QPID-932		M2.1	(QPID-932) (Java) Remove references to unused constructor argument "browsedAcks" of NonTransactionalContext	Open	Rob Godfrey	Rob Godfrey
QPID-933	Java Broker, Java Client, Java Common	M2.1	(QPID-933) (Java Performance) reduce memory copies, boxing/unboxing and needless iterating	Resolved	Rob Godfrey	Rob Godfrey
QPID-934	C++ Broker	M3	(QPID-934) Federation tests fail intermittently	Open	Unassigned	Ted Simc
QPID-935			(QPID-935) patch to make trunk code work with old boost and gcc versions	Open	Unassigned	michael goulart
QPID-936	Java Tests	M3	(QPID-936) Test testMigrateDurableSubscriber from org.apache.qpid.test.unit.xa.TopicTest is failing	Open	Unassigned	Arnaud Simon
QPID-937			(QPID-937) Typo in getting-started page of the site	Open	Unassigned	Surya Jaya

Qpid Java Meeting Minutes 2008-05-09

Agenda

Attendees

Rob Godfrey

Rajith Attapattu

Rafi Schlominger

Review of Code Commits

revision	committer	date	comment	review points
r649339	aconway	2008-04-18	src/tests/python_tests: fix exit status if QPID_NO_PREVIEW is set. src/qpid/framing/AMQFrame.h: frame header setters/getters with 0-10 naming.	
r649409	arnaudsimon	2008-04-18	QPID-798 Make property per connection and/or system wide; change name to SYNC_PERSISTENT to denote only used on persistent messages	
r649432	aidan	2008-04-18	QPID-832 default protocol version to 0-9	
r649436	arnaudsimon	2008-04-18	QPID-936 : added missing session close op	
r649479	kpvdr	2008-04-18	Fix to prevent possible Timer deadlocks by holding onto mutex while calling fire()	
r649547	aconway	2008-04-18	Uncommented tests.	
r649554	aconway	2008-04-18	From Ted Ross: QPID-934 " title="Visit page outside Confluence" rel="nofollow"linktype="raw" linktext=" https://issues.apache.org/jira/browse/QPID-934 "> QPID-934 "> https://issues.apache.org/jira/browse/QPID-934 This patch fixes a problem related to multiple management sessions run over the same AMQP session (typically seen in test environments). ...	
r649571	aconway	2008-04-18	Fix build problem.	
r649585	aidan	2008-04-18	Merged revisions 648217-649481 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/trunk r648272 - aconway - 2008-04-15 15:54:46 +0100 (Tue, 15 Apr 2008) - 1 line Fix b ...	
r649642	kpvdr	2008-04-18	fix ambiguity problem found on gcc 3.4 compilers	
r649648	rhs	2008-04-18	QPID-901 : update pom to work with new codegen	
r649661	aidan	2008-04-18	Merged revisions 649482-649660 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/trunk r649547 - aconway - 2008-04-18 15:12:36 +0100 (Fri, 18 Apr 2008) - 2 lines Unco ...	
r649666	astitcher	2008-04-18	Split AsynchIOAcceptor into IOHandler and connection control code	
r649671	aconway	2008-04-18	Fix test failure.	
r649689	astitcher	2008-04-18	Refactored Acceptor code to allow multiple acceptors to be present in the broker	
r649691	astitcher	2008-04-18	Added missed new include file	
r649711	aidan	2008-04-18	QPID-832 fix some compile errors	
r649790	aidan	2008-04-19	QPID-832 ahem	
r649905	aidan	2008-04-20	QPID-832 now 95% bug free	
r649907	aidan	2008-04-20	QPID-832 remove log4j spewage	
r649909	aidan	2008-04-20	Merged revisions 649661-649908 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/trunk r649666 - astitcher - 2008-04-18 20:44:25 +0100 (Fri, 18 Apr 2008) - 2 lines Sp ...	
r649915	gsim	2008-04-20	QPID-920 : converted c++ client to use final 0-10 protocol * connection handler converted to using invoker & proxy and updated to final method defs * SessionCore & ExecutionHandler replace by SessionIm ...	
r650099	aidan	2008-04-21	QPID-832 Make sure that we lock early enough to avoid deadlocks when closing	
r650108	aidan	2008-04-21	QPID-832 revert last commit, 650099	
r650122	aidan	2008-04-21	QPID-832 pass a genuine connection so that it doesn't NPE	
r650127	arnaudsimon	2008-04-21	QPID-939 Added "ID:" to message ID	
r650136	rgodfrey	2008-04-21		

r650145	rgodfrey	2008-04-21	Wrong revision	
r650146	rgodfrey	2008-04-21	create branch for broker refactoring	
r650148	rgodfrey	2008-04-21	Initial checkpoint of queue refactoring work	
r650159	gsim	2008-04-21	QPID-920 : send message-accept for acks (as well as completion) * AckPolicy now maintains a set of transferred messages for cumulative accepts	
r650179	rgodfrey	2008-04-21	remove duplicate check of interest in enqueue, enable new Queue by default	
r650193	aidan	2008-04-21	QPID-832 catch dodgy mock generated exception, just like M2.x	
r650198	aconway	2008-04-21	src/qpid/RangeSet.h: generic set implementation using ranges. - no heap allocation for simple sets (<= 3 ranges) - binary searches for o(log ) performance in complex sets	
r650205	aidan	2008-04-21	QPID-832 handle multiple brokers properly	
r650210	aconway	2008-04-21	Fix compile error on rhel5.	
r650221	aconway	2008-04-21	Better workaround for boost::ptr_map incompatibility between boost 1.33 and 1.34, based on public properties of ptr::map types rather than version numbers.	
r650227	aconway	2008-04-21	Disable compilation of amqp_0_10 codec until ready for integration.	
r650250	gsim	2008-04-21	<ul style="list-style-type: none"> raise error when controls other than attached are received on unattached channel * corrected exception handling in client and on broker (broker to issue detach) 	
r650273	aconway	2008-04-21	Fix packaging problems for rpmbuild.	
r650359	rhs	2008-04-22	QPID-832 : fixed ant build system	
r650439	gsim	2008-04-22	QPID-648 : (based on patch from mfarrellee@redhat.com) * apply authentication to final 0-10 codepath * consolidate conditional compilation of sasl-related code * improved handling of connection close ...	
r650440	rhs	2008-04-22	QPID-823 : add junit to the libs for junit-toolkit	
r650447	aidan	2008-04-22	create forrest site	
r650450	gsim	2008-04-22	QPID-944 : do no-local checking where requested when there is an exclusive subscription active	
r650478	aidan	2008-04-22	Merged revisions 649909-650455 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/trunk r649915 - gsim - 2008-04-20 13:10:37 +0100 (Sun, 20 Apr 2008) - 6 lines QPID-92 ...	
r650565	rhs	2008-04-22	QPID-947 : update cpp and python management to 0-10 final	
r650579	rhs	2008-04-22	QPID-832 : updated build order	
r650581	rhs	2008-04-22	QPID-832 : moved 0-8 specific code into 0-8 subclass of session	
r650598	rhs	2008-04-22	QPID-832 : moved more 0-8 specific code to 0-8 subclasses	
r650604	rhs	2008-04-22	QPID-948 : patch from Ted Ross for updated management utilities to 0-10 final	
r650617	rhs	2008-04-22	QPID-832 : moved more 0-8 specific code into 0-8 subclasses	
r650620	cctrielloff	2008-04-22	QPID-945 from Ted Ross	
r650635	gsim	2008-04-22	Moved federation to final 0-10 codepath	
r650640	gsim	2008-04-22	QPID-920 : allow applications to trigger the sending of a flush to server	

r650657	astitcher	2008-04-22	<ul style="list-style-type: none"> Renamed the Acceptor class to be the ProtocolFactory class which better approximates its current behaviour * Slightly refactored TCPIOPlugin to better approximate how it would look when we imple ... 	
r650658	aidan	2008-04-22	QPID-832 make tests use new QpidTestCase magic for getting connections	
r650795	gsim	2008-04-23	Added to the no-local tests, cleaned up comments (and highlighted non-standard nature of these tests)	
r650813	aidan	2008-04-23	QPID-832 remember to pass username to getConnection	
r650850	rgodfrey	2008-04-23	QPID-832 : Quoted identifiers grammar fix for client side selectors	
r650855	rhs	2008-04-23	QPID-832 : switched from execing javacc to using the javacc task; this should fix the build on cygwin	
r650858	rhs	2008-04-23	QPID-832 : moved CloseTests and DurableSubscriberTests -> CloseTest and DurableSubscriberTest	
r650875	gsim	2008-04-23	Add support for reading 0-10 arrays; Set sync bit on session header for commands sent with auto_sync on.	
r650876	rgodfrey	2008-04-23	QPID-832 : connection should be set to started before sessions are started	
r650887	rgodfrey	2008-04-23	QPID-832 : Fixed AMQSession_0_10 so that it takes the acknowledge mode from the session not a system definition	
r650890	aidan	2008-04-23	QPID-832 fix failover detection, rename startDispatcher	
r650901	rhs	2008-04-23	QPID-832 : fixed override of notifyMessage, this re-enables selectors for 0-10	
r650906	aconway	2008-04-23	<ul style="list-style-type: none"> SequenceSet implemented on RangeSet. - Reduced #include dependencides on SequenceSet.h 	
r650922	astitcher	2008-04-23	Make python tests work with VPATH builds	
r650970	astitcher	2008-04-23	Patch from Mick Goulish: Fixes to previous improved portability patch	
r650997	aconway	2008-04-23	src/tests/ClientSessionTest.cpp: uncommented tests for session resume as EXPECTED_FAILURES tests. src/tests/unit_test.h: workarounds for broken EXPECTED_FAILURES tests in boost <= 1.34	
r650999	astitcher	2008-04-23	Reverted earlier change to valgrind suppressions	
r651088	aconway	2008-04-23	Fix build error introduced by earlier commit.	
r651107	aidan	2008-04-23	QPID-832 copy trunk before performing some major operations	
r651111	aidan	2008-04-23	Delete stuff that's just going to get synced from M2.x	
r651112	aidan	2008-04-23	QPID-832 copy the M2.x broker	
r651113	aidan	2008-04-23	QPID-832 sync from M2.x	
r651115	aidan	2008-04-24	QPID-832 copy from M2.x	
r651116	aidan	2008-04-24	QPID-832 copy from M2.x	
r651117	aidan	2008-04-24	QPID-832 copy from M2.x	
r651118	aidan	2008-04-24	QPID-832 copy from M2.x	
r651119	aidan	2008-04-24	QPID-832 copy from M2.x	
r651120	aidan	2008-04-24	QPID-832 copy from M2.x	
r651124	aidan	2008-04-24	QPID-832 sync build stuff from thegreatmerge	
r651125	aidan	2008-04-24	QPID-832 sync build stuff from thegreatmerge	

r651126	aidan	2008-04-24	QPID-832 sync build stuff from thegreatmerge	
r651133	aidan	2008-04-24	QPID-832 merge M2.x	
r651134	aidan	2008-04-24	QPID-832 nuke some obsolete stuff	
r651211	aidan	2008-04-24	QPID-832 update pom version	
r651276	rhs	2008-04-24	QPID-832 : fixed DerbyMessageStore to compile on Java 1.5 (hopefully)	
r651287	aconway	2008-04-24	emacs/qpid-c+-mode.el: qpid-c+-mode for Emacs. qpid-style indentation plus some useful commands for inserting copyrights etc. Feel free to improve it, there's lots of room.	
r651290	cctrielloff	2008-04-24	QPID-953 from tross	
r651301	aidan	2008-04-24	QPID-832 make systests run	
r651321	aconway	2008-04-24	Edits to doxygen comments for user documentation.	
r651325	rgodfrey	2008-04-24	QPID-832 : Fix eol-style	
r651346	nsantos	2008-04-24	add include for list, for gcc4.3	
r651375	rgodfrey	2008-04-24	QPID-832 : Fix CRLF line endings	
r651421	gsim	2008-04-24	Correct expected error codes for final 0-10 spec	
r651423	gsim	2008-04-24	Generate c++ code from final 0-10 spec	
r651425	cctrielloff	2008-04-24	QPID-958 by Danushka Menikkumbura	
r651431	rhs	2008-04-24	QPID-832 : fixed merge artifact affecting replyTo	
r651474	astitcher	2008-04-25	Fix building examples when using a VPATH build	
r651531	gsim	2008-04-25	Fixed caught exception type in recovery	
r651997	aconway	2008-04-27	Session state as per AMQP 0-10 specification.	
r652037	cctrielloff	2008-04-28	QPID-918 patch from Senaka	
r652041	cctrielloff	2008-04-28	QPID-971 - Senaka Fernando, I also edited the README in a few places.	
r652053	astitcher	2008-04-28	Work In Progress: Added initial rdma code including test server and client Turn off rdma support by default but autoconf should now detect whether necessary rdma/ibverbs libs and headers are pre ...	
r652075	gsim	2008-04-29	QPID-977 : shutdown mgmt client cleanly in federation tests (patch from tross@redhat.com) QPID-981 : added custom options to queue declare to tag each message as it goes through a bridge queue and allow ...	
r652076	gsim	2008-04-29	QPID-981 : management schema change missed from previous commit	
r652083	gsim	2008-04-29	QPID-974 : allow the size of the queue of outgoing frames to be restricted QPID-544 : tidy up configuration (ensuring desired settings are used correctly, allowing tcp socket options to be se ...	
r652086	rhs	2008-04-29	QPID-979 : added backwards compatible uuid to qpid.datatypes	
r652114	gsim	2008-04-29	QPID-981 : allow id and exclude list to be passed through when creating a bridge with qpid-route	
r652120	astitcher	2008-04-29	Removed some unnecessary #includes	
r652170	rhs	2008-04-29	QPID-984 : override MINA's IoServiceListenerSupport class in order to fix infinite loop	Investigate whether this can be avoided by synchronizing transport connection
r652173	rhs	2008-04-29	QPID-983 : fixed deadlock between AMQConnection.close and FailoverHandler	
r652180	astitcher	2008-04-29	More RDMA Work in Progress Changes to client buffering Buffering improvement to server Removed unused state machine from RdmaIO code Move the write throttling due to limited write buff ...	

r652386	gsim	2008-04-30	QPID-988 and QPID-989 : fixes to framing for final 0-10 spec	
r652388	ritchiem	2008-04-30	QPID-889 : Removed _reapingStoreContext from CSDM replaced with local StoreContext(s) so they are not reused by different threads.	
r652389	ritchiem	2008-04-30	QPID-887 : Renamed QueueHouseKeeping threads so they can be identified in thread dump. Named Queue-housekeeping-<virtualhost name>	Changed more than just the name. Also adds a System exit - this should be documented. Action: move log statements to debug - not info.
r652399	ritchiem	2008-04-30	QPID-888 ,QPID-886 : Fixed all management uses of _lock.lock / _lock.unlock so that they correctly call unlock from a finally block in the CSDM. There are two issues that cover that. QPID-888 - Fix the ...	
r652409	rajith	2008-05-01	This commit is for QPID-992 and QPID-993 build.xml ===== I added the broker-plugins as a module to the ant build system. This was nessacery to get the plugins jar generated for QPID-993 . In gene ...	module.xml should not include specific code such as the copy which breaks building other modules.
r652411	rajith	2008-05-01	Added the following tests to 0-10 exclude lists as these tests are for the java broker. However when the java broker gets to 0-10 we should also have exclude lists per broker as well. For the time bei ...	
r652451	gsim	2008-05-01	Cleanup: Re-enable tests that now pass; delete unused templates directory.	
r652469	gsim	2008-05-01	QPID-966 : applied patch from rajith; altered to use uuid as session name; updated verify scripts for automated testing; re-enabled automated testing in c++ build	
r652506	gsim	2008-05-01	Remove preview tests (no longer required)	
r652535	nsantos	2008-05-01	applying Ted Ross's patch to handle unicode encoding and management message ordering issues	
r652548	rhs	2008-05-01	QPID-965 : added an option to turn on deprecation during compile	
r652558	gsim	2008-05-01	QPID-989 : fix decode of zero sized map	
r652567	aidan	2008-05-01	QPID-994 Dont wait for attain state as connection is closed by we get CloseOk	Wrong JIRA refrrred to... Would like some justification for change.
r652568	aidan	2008-05-01	QPID-1001 dont set the expiration time if TTL is 0	
r652591	gsim	2008-05-01	Turn auth back on by default for c++ broker (only if SASL libs are available)	
r652670	rhs	2008-05-01	QPID-987 : reduced message count in DupsOKTest	
r652672	rhs	2008-05-01	QPID-993 : added an osgi manifest to broker-plugins jar	
r652680	rhs	2008-05-01	QPID-1002 : applied patch from Senaka to make systests run using normal junit testrunner	
r652689	gsim	2008-05-01	Boost's string split function causes problems on older versions of the library. Replaced with homegrown equivalent.	
r652705	rhs	2008-05-01	QPID-1003 : added excludes for framework test classes	
r652779	gsim	2008-05-02	QPID-986 : Patch from Danushka Menikkumbura.	
r652783	gsim	2008-05-02	QPID-980 : Patch from Danushka Menikkumbura revising installation notes.	
r652799	gsim	2008-05-02	Use BOOST_CHECK_EQUAL in place of BOOST_REQUIRE_EQUAL (compatible with older boost)	
r652829	gsim	2008-05-02	Use no-ack in bridging as it is currently an exclusive, temp queue (will eventually be configurable)	
r653248	gsim	2008-05-04	Fix error handling for connection close during startup.	
r653249	gsim	2008-05-04	Extra log ouput for queue policy.	
r653251	gsim	2008-05-04	Use amq.direct for control queue in topic test.	

r653252	gsim	2008-05-04	Allow queue durability to be specified independent of message durability.	
r653253	gsim	2008-05-04	Update and cleanup scripts for automated 0-10 example testing.	
r653354	arnaudsimon	2008-05-05	QPID-1006 and QPID-1007 : -QPID-1006 :use same socket buffer size and frame size -QPID-1007 : added io write handler into MINA chain	Use Integer.getInteger and int constants rather than String constants and parsing. If System Property not an integer use default rather than none, still with warning, or throw RuntimeException? Should we have a magic value for unbounded? What are valid value for these numbers - what is the effect of a negative number... We shouldn't be catching "Exception" but specific subtype. in connect(): "true".equals should instead be Boolean.getBoolean having tcpNoDelay default to true is suspect Constant being configured is it the buffer size of the max frame size? We may default the buffer size to the max frame size... but surely the disassembler is being created relative to the max frame size?
r653399	arnaudsimon	2008-05-05	QPID-1018 : added org.apache.qpid.client.configuration and add io props	AMQConnection.java: use Long.getLong rather than System.getProperties... Can we make properties into some sort of enum/class with get accessors Property names should be made more consistent
r653400	arnaudsimon	2008-05-05	QPID-1007 : removed bad imports	
r653415	ritchiem	2008-05-05	QPID-887 : Renamed QueueHouseKeeping threads so they can be identified in thread dump. Named Queue-housekeeping-<virtualhost name>	This change does not have the same System.exit(-1) that the equivalent change on M2.x had
r653416	aidan	2008-05-05	QPID-1019 prevent messages being dequeued unnecessarily, from rgodfrey	
r653419	aidan	2008-05-05	turn on merge tracking between trunk and M2.x	
r653421	ritchiem	2008-05-05	QPID-895 : Patch provided by Senaka to prevent delay on initial Connections with SingleServer methods. Updated FailoverMethodTest to include a better description of where the times come from....	
r653426	arnaudsimon	2008-05-05	QPID-1005 : Added a property for ignoring setClientID so we are compatible with legacy applications that don't rely on the ID being set on the connection URL.	change name of property as it give wrong impression about effect.
r653427	ritchiem	2008-05-05	QPID-998 : prevented warning about missing bin/etc directories in modules that don't have bin or etc.	should have a variable to control the echo, if-verbose or something
r653434	ritchiem	2008-05-05	QPID-1021 : Update to build system to store and clean test results per module.name.	Changing of test results directory to new directory shouldn't have been done without consultation-should have been posted on the list. JIRA was filed and closed within 30 minutes. Broke the test report task
r653439	ritchiem	2008-05-05	QPID-1021 : Update to build system to store and clean test results per module.name. Sorry errant } broke everything.	
r653441	ritchiem	2008-05-05	QPID-997 : Cause of delay is the missing / in the file url for the log4j configuration file. Under windows the path would start file:///c:/ which log4j assumes is some remote file system. For windows i ...	

r653447	aidan	2008-05-05	Check if consumer is closed and dont reclose it	No JIRA number!
r653451	aidan	2008-05-05	QPID-1022 Use synchronous writes to fix race conditions	
r653452	aidan	2008-05-05	QPID-1023 increase some timeouts	
r653508	gsim	2008-05-05	QPID-1008 : allow for case where 0-10 message-properties are not included in a received message	Wrong { convention
r653518	arnaudsimon	2008-05-05	QPID-1025 : changed received so empty Payload are processed	Rafi to review with Arnaud
r653527	gsim	2008-05-05	Updated for latest 0-10 spec and added two extra queue options to set the flow to disk policy trigger size/count	Use StringBuilder for String concatenation binddingKey has too many 'd's the exchange name / binding key should be truncated so that the entire string fits inside 255 bytes. See rules laid down in exchange on qpid-dev between rgodfrey & cctrielloff. Care needs to be taken with multibyte unicode characters.
r653720	aidan	2008-05-06	Merged revisions 652388-652389,652399,652567-652568,653416 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/branches/M2.x r652388 - ritchiem - 2008-04-30 15:40:18 +0100 (...	
r653731	arnaudsimon	2008-05-06	QPID-1028 : updated report task for including all subdirs	
r653760	aidan	2008-05-06	QPID-1029 : Generate temporary queue names using GUIDs to ensure uniqueness.	
r653813	arnaudsimon	2008-05-06	QPID-1030 : This solves the issue for the 0.10 code path	
r653830	rajith	2008-05-06	This is a fix for QPID-1031 I added read/write methods for datetime that calls int64.	
r653854	aconway	2008-05-06	From QPID-879 " title="Visit page outside Confluence" rel="nofollow"linktype="raw" linktext=" https://issues.apache.org/jira/browse/QPID-879 "> QPID-879 "> https://issues.apache.org/jira/browse/QPID-879 contributed by Jonathan Robie. XML exchange allowing messages to be routed base on XQuery expressions.	
r653875	rhs	2008-05-06	QPID-1033 : made loading of the spec file not fail if the results cannot be cached, e.g. due to an unwritable directory	
r653904	rajith	2008-05-06	This patch was attached to QPID-953 . It allows to specify a comma separated list of queue names to filter with -f flag. Also I removed getopt and added optparse as it provides a more easy way of handl ...	
r653912	aconway	2008-05-06	Fix for defining HAS_XML	
r654097	aidan	2008-05-07	QPID-952 , QPID-951 , QPID-1032 Fix failover, ensure that it is properly detected, that frames are replayed appropriately and that failover does not timeout.	
r654104	aidan	2008-05-07	QPID-952 should have been part of previous commit	
r654109	aidan	2008-05-07	QPID-1036 increase timeouts to more reasonable levels, ensure that durable queues are deleted when no longer needed	
r654113	aidan	2008-05-07	Merged revisions 653420-654109 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/branches/M2.x r653447 - aidan - 2008-05-05 13:26:29 +0100 (Mon, 05 May 2008) - 1 line ...	
r654125	aidan	2008-05-07	Add integration tests to #D project	
r654143	aidan	2008-05-07	Tidy up #D project a bit	
r654144	aidan	2008-05-07	Tidy up #D project a bit	
r654158	rhs	2008-05-07	QPID-979 : added convenience accessors for headers	

r654457	arnaudsimon	2008-05-08	QPID-1004 : Disable this test as it is not finished and TTL is already tested	undo this change as the subsequent change disables TTLTest in the correct way
r654464	arnaudsimon	2008-05-08	QPID-1004 : As this test does not extend QpidTestCase we need to exclude it from the build.xml.	Log comment on commit is wrong - it is being excluded because the test is broken
r654505	arnaudsimon	2008-05-08	QPID-1037 : Added module name prop and an exclude.modules prop + switch systests off in cpp profiles.	rhs: don't think this change is necessary. Also why module.name being set when is set automatically with exception of junit-toolkit suggest most of this change should be reversed systests execution time should be fixed by fixing timeout This change should have been discussed on list (excluding an entire test suite is a *big* change)
r654513	arnaudsimon	2008-05-08	QPID-1037 : revert the junit-toolkit change	
r654529	aidan	2008-05-08	QPID-1038 add test case	
r654545	aidan	2008-05-08	QPID-1038 add a mixed send/recieve/rollback test	
r654564	nsantos	2008-05-08	QPID-1035 : managementgen only exists in qpid C++ source - applying patched supplied by Matt Farrellee	
r654566	nsantos	2008-05-08	QPID-1026 : managementgen C++ symbol validation - applied patch supplied by Matt Farrellee	
r654574	nsantos	2008-05-08	QPID-1035 : managementgen only exists in qpid C++ source - applying patched supplied by Matt Farrellee	
r654584	nsantos	2008-05-08	QPID-1035 : managementgen only exists in qpid C++ source - applying patched supplied by Matt Farrellee	
r654618	rhs	2008-05-08	QPID-979 : added access to enums through the session so that symbolic constants can be used rather than hard coded ones; also added default loading of the spec	
r654623	rhs	2008-05-08	QPID-979 : switched to a more appropriate name for locating the spec	
r654637	rhs	2008-05-08	QPID-979 : added qpid_config.py appropriate for devel checkout	
r654666	astitcher	2008-05-09	QPID-1040 : Patch from Ted Ross: Asynchronous Connector Code to allow non-blocking connection of new sockets	
r654710	gsim	2008-05-09	Make ANONYMOUS the default authentication mechanism.	
r654712	gsim	2008-05-09	Make ANONYMOUS the default mechanism	
r654737	gsim	2008-05-09	QPID-1042 : ensure delivery record is kept where accept_mode=not-required, acquire_mode=not-acquired and flow_mode=credit	
r654759	gsim	2008-05-09	Enabled PLAIN authentication and setting of username and password for 0-10 python client. Added options to all command line tools to allow a username and password to be specified.	
r654761	gsim	2008-05-09	Reverted change to use ANONYMOUS as default (I had a change of heart on that)	
r654780	ritchier	2008-05-09	Create M2.1 Bug fix branch	
r654785	ritchier	2008-05-09	Moved all references to M2.1 to M2.1.x-SNAPSHOT	

Recently Raised JIRAs

Key	Component(s)	Affects Version/s	Summary	Status	Assignee	Reporter	Review Comments
QPID-1001	Dot Net Client	M2.1, M3	(QPID-1001) .Net client sets expiration time incorrectly	Open	Aidan Skinner	Aidan Skinner	Resolved?

QPID-1002	Java Tests		(QPID-1002) Improvements to Test Framework making it possible to run systests	Closed	Martin Ritchie	Senaka Fernando	affects version should be set
QPID-1003	Ant Build System, Java Tools		(QPID-1003) Junit treats Classes ending with Test as testcases	Closed	Martin Ritchie	Senaka Fernando	affects version should be set
QPID-1004	Java Tests		(QPID-1004) org.apache.qpid.test.testcases.TTLTest Fails	Open	Unassigned	Senaka Fernando	
QPID-1005	Java Client	M3	(QPID-1005) Client ID	Open	Unassigned	Arnaud Simon	need more descriptive title. Issue needs more work - see notes on commit above
QPID-1006	Java Client	M3	(QPID-1006) (0.10 code path) Message rate is very slow when big messages are sent.	Resolved	Arnaud Simon	Arnaud Simon	Should be reopened because of comments above
QPID-1007	Java Client	M3	(QPID-1007) (0.10 code path) Need a way of controlling MINA queue size	Resolved	Unassigned	Arnaud Simon	
QPID-1008	Java Client	M3	(QPID-1008) 0-10 client throws NPE if message-properties are not present	Open	Unassigned	Gordon Sim	Should be marked as resolved
QPID-1009	Java Management Console		(QPID-1009) Update JMX Management Console FAQ on using scrips on Unix systems	Open	Unassigned	Senaka Fernando	
QPID-1010	Java Management Console		(QPID-1010) Buttons are not visible in New Connection Dialog Box	Open	Unassigned	Senaka Fernando	Someone should look at patch
QPID-1011	Java Management Console		(QPID-1011) Qpid Management Console Plugin has undefined behaviour in different Eclipse Perspectives	Open	Unassigned	Senaka Fernando	Someone should look at patch
QPID-1012	website		(QPID-1012) Attachments Links unavailable on Configuring Qpid Management Console wiki page	Open	Unassigned	Senaka Fernando	
QPID-1013	Java Management Console		(QPID-1013) Documentation on running Qpid Management Console within Eclipse is Required	Open	Unassigned	Senaka Fernando	
QPID-1014	Java Management Console		(QPID-1014) qpid.management.perspective should be replaced by a better phrase	Open	Unassigned	Senaka Fernando	Someone should look at patch
QPID-1015	Java Management Console		(QPID-1015) NPE Reported Incorrectly by Qpid Management Console	Open	Unassigned	Senaka Fernando	Someone should look at patch
QPID-1016			(QPID-1016) Can not get qpid-tool to run	Closed	Unassigned	Danushka Menikkumbura	
QPID-1017	Python Test Suite		(QPID-1017) README in Qpid Python has wrong name for run-tests	Open	Unassigned	Senaka Fernando	Someone should look at patch
QPID-1018	Java Client	M3	(QPID-1018) Client properties centralization	Open	Arnaud Simon	Arnaud Simon	See comments above
QPID-1019	Java Broker	M2.1, M3	(QPID-1019) Message cleanup can cause refcount to go below 0	Open	Rob Godfrey	Aidan Skinner	Should be marked resolved

QPID-1020	Ant Build System	M3	(QPID-1020) Test classes should be built by default ant run.	Open	Martin Ritchie	Martin Ritchie	
QPID-1021	Ant Build System	M3	(QPID-1021) Module test results should be listed by module and removed by clean	Open	Martin Ritchie	Martin Ritchie	Should be marked resolved
QPID-1022	Dot Net Client	M2.1, M3	(QPID-1022) Client doesn't wait for replies correctly	Open	Unassigned	Aidan Skinner	Should be marked resolved
QPID-1023	Dot Net Client	M2.1, M3	(QPID-1023) Various tests fail due to insufficient time outs	Open	Unassigned	Aidan Skinner	Should be marked resolved
QPID-1024	C++ Broker	M3	(QPID-1024) When a message is released by the client the c++ broker continues to serve the same message	Closed	Gordon Sim	Rajith Attapattu	
QPID-1025	Java Client	M3	(QPID-1025) (0.10 code path) Empty body messages are not propagated to the consumer	Open	Unassigned	Arnaud Simon	
QPID-1026	Code Generator		(QPID-1026) managementgen C++ symbol validation	Open	Unassigned	Matthew Farrellee	
QPID-1027	Python Test Suite		(QPID-1027) verify script is sensitive to shell in use	Open	Unassigned	Senaka Fernando	
QPID-1028	Ant Build System	M3	(QPID-1028) Test report is not generated	Open	Unassigned	Arnaud Simon	Should be marked resolved
QPID-1029	Dot Net Client	M2.1, M3	(QPID-1029) Client generates non-unique temporary queue names	Open	Aidan Skinner	Aidan Skinner	
QPID-1030	Java Client	M3	(QPID-1030) Making JMS easier for users	Open	Unassigned	Arnaud Simon	Change title to be more reflective on content. Also see comments on commit
QPID-1031	Python Client	M3	(QPID-1031) Python doesn't handle date time for 0-10 final	Resolved	Rajith Attapattu	Rajith Attapattu	
QPID-1032	Dot Net Client	M2.1, M3	(QPID-1032) Failover replays frames but is unable to handle the OKs that are generated	Open	Aidan Skinner	Aidan Skinner	Should be marked as resolved
QPID-1033	Python Client	M3	(QPID-1033) Continue on error if cannot create spec pcl	Resolved	Rafael H. Schlomming	Justin Ross	
QPID-1034			(QPID-1034) README on verify scripts	Open	Unassigned	Senaka Fernando	
QPID-1035	Code Generator		(QPID-1035) managementgen only exists in qpid C++ source	Open	Unassigned	Matthew Farrellee	can this be resolved
QPID-1036	Dot Net Client	M2.1, M3	(QPID-1036) Integration tests generate false negatives	Open	Aidan Skinner	Aidan Skinner	Should be marked as resolved
QPID-1037	Java Tests	M3	(QPID-1037) Need a way of not running some module tests	Closed	Arnaud Simon	Arnaud Simon	
QPID-1038	Dot Net Client	M2.1, M3	(QPID-1038) Blocking I/O in transport layer leads to poor performance for mixed sender/producer	Open	Aidan Skinner	Aidan Skinner	Should be marked as resolved
QPID-1039	C++ Broker		(QPID-1039) fix program options behavior for Boost 103200	Open	Unassigned	michael goulish	
QPID-1040	C++ Broker	M3	(QPID-1040) Asynchronous Protocol Connector for C++ broker	Open	Andrew Stitzer	Ted Ross	Can be resolved?
QPID-1041	Python Client	M3	(QPID-1041) Sources and sinks should all have names (a problem in at least the Python client)	Open	Rafael H. Schlomming	Jonathan Robie	

QPID-1042	C++ Broker	M3	(QPID-1042) Record of deliveries not kept for non-acquired, non-accepted subscriptions in credit flow mode	Open	Gordon Sim	Gordon Sim	Can be resolved?
QPID-1043	website	M3	(QPID-1043) Add page with a clear grid of protocol support for each qpid component	Open	Gordon Sim	Gordon Sim	

Qpid Java Meeting Minutes 2008-05-16

Agenda

Check on progress of last weeks review points
 Commit review
 JIRA Review
 Landing Java Broker Refactoring

Attendees

Aidan, Arnaud, Rafi, Carl, Martin, Rob, Marnie

Outstanding actions

Almost all prior actions need to be taken, please seem last weeks minutes for details

Committers need to get better at reviewing submitted patches

Review of Code Commits

revision	committer	date	comment	review points
r654799	aidan	2008-05-09	allow merging from M2.x	
r654803	aidan	2008-05-09	allow merging from M2.x	
r654818	aidan	2008-05-09	Merged revisions 652388-653415,653417-654109 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/branches/M2.x r652388 - ritchiem - 2008-04-30 15:40:18 +0100 (Wed, 30 Apr 20 ...	
r654866	astitcher	2008-05-09	Fix to managementgen so that VPATH builds work	
r654902	gsim	2008-05-09	QPID-648 : Patch from Matt Farrellee - support for realms - updates to packaging to create a default db and the necessary conf files for plain and anon	
r654907	rhs	2008-05-09	QPID-1045 : always notify incoming message queues of session closure and provide API for notifying listeners of closure; also preserve connection close code and report in errors	
r654913	aconway	2008-05-09	Support for 0-10 sessions, not yet integrated. Misc minor fixes.	
r654918	rhs	2008-05-09	QPID-1045 and QPID-1041 : added a destination attribute to incoming queues, and added a start() method to incoming queues as syntactic sugar for the verbose message flow idiom	
r654927	nsantos	2008-05-09	QPID-1047 : Qpidc.spec.in is missing dependencies and has misplaced files - applied patch from Matt Farrellee	
r654947	rhs	2008-05-09	QPID-947 : made python client use execution.sync instead of session.flush when not in auto_sync mode	
r655323	rgodfrey	2008-05-11	Updates on the refactoring work	
r655326	rgodfrey	2008-05-11	Copy over QPID-925	
r655330	rgodfrey	2008-05-11	Copy over QPID-926	
r655353	gsim	2008-05-11	QPID-1048 : Only wait for enqueue completion for persistent queues.	
r655455	gsim	2008-05-12	Script to test federated setup using the topic test.	
r655470	aidan	2008-05-12	QPID-839 fix test to avoid potential race condition and general incorrectness	AS: merge to trunk
r655494	gsim	2008-05-12	Couple of extra simple tests for publishing and consuming in generic fashion.	
r655495	gsim	2008-05-12	Extra ignores	

r655533	rhs	2008-05-12	QPID-947 : added handler for known_completed and generate known_completed when timely-reply is set	
r655534	rhs	2008-05-12	QPID-947 : fixed typo in prior commit	
r655536	rhs	2008-05-12	QPID-1037 : removed manual setting of the module.name property as it is set automatically by the build scripts	
r655563	gsim	2008-05-12	QPID-1050 : Patch from Ted Ross: 1) Durability for federation links (broker-to-broker connections) 2) Improved handling of federation links: a) Links can be created even if the remote broker is no ...	
r655568	gsim	2008-05-12	QPID-1044 : Part of patch from Jonathan Robie + changes to verify scripts to keep automated testing working.	
r655585	rhs	2008-05-12	QPID-1025 : updated fix for empty payload issue, this change removes state transitions that don't consume input bytes	
r655596	astitcher	2008-05-12	QPID-1039 : Patch from Mick Goulish: Fix program options behavior for Boost 103200	
r655597	astitcher	2008-05-12	Fix to allow VPATH builds to work after checkin for QPID-648	
r655619	gsim	2008-05-12	QPID-1052 : Patch from Ted Ross This patch contains the following: 1) The session-id reported by the management API now matches the session.name in the session table 2) management.py API has a new ca ...	
r655630	rgodfrey	2008-05-12	More fixing up of refactoring stuff; getting all maven tests passing and implementing management methods	
r655781	rgodfrey	2008-05-13	Fixed broken exception overriding	
r655790	gsim	2008-05-13	Fix macro used in test for backwards compatibility.	
r655798	rgodfrey	2008-05-13	Changes to MessageStore interface	
r655915	nsantos	2008-05-13	QPID-1052 : Management: session.name matches session id provided by API, handling of lost connections - applied patch supplied by Ted Ross	
r655923	arnaudsimon	2008-05-13	QPID-1006 : Don't use tcp-nodelay as default and set socket buffer size only when the corresponding property is set.	Discuss on list, config item, more investigation required
r655927	rhs	2008-05-13	QPID-1053 : updated QpidTestCase to check against broker output to ensure the broker is actually listening before the test attempts to connect; the text checked for is controlled by the broker.ready sy ...	
r655935	aconway	2008-05-13	Added enum CreditUnit MESSAGE=0, BYTE=1 ;	
r655944	gsim	2008-05-13	QPID-1054 : Fixed reporting of startup failures in daemon mode.	
r655951	rhs	2008-05-13	QPID-1055 : use int64 for encoding python both python int and longs; this ensures consistent behavior on both 64 bit and non 64 bit systems	
r655956	aconway	2008-05-13	Removed confusing broker::Message typedef intrusive_ptr<Message> shared_ptr	
r655957	gsim	2008-05-13	Fail with exception if queue is not durable and configured policy is exceeded.	
r655964	rhs	2008-05-13	QPID-1053 : add a timeout in case the broker is never ready	
r655965	aconway	2008-05-13	Fix typo in examples.	
r655966	aconway	2008-05-13	Ignore sasldb	
r655968	gsim	2008-05-13	Minor change to tests to use correlation id rather than body for identifying messages.	
r655976	rhs	2008-05-13	QPID-954 : added fallbacks and fixes for running the python client on python 2.3	
r655983	aconway	2008-05-13	Added sync() to ensure all acks are received before exiting the Dispatcher loop.	
r656004	nsantos	2008-05-13	continuation of QPID-1052	

r656005	aconway	2008-05-13	<p>From Jonathan Robie: QPID-1056 : title="Visit page outside Confluence" rel="nofollow"linktype="raw" linktext="https://issues.apache.org/jira/browse/QPID-1056 :"> QPID-1056 : title="Visit page outside Confluence" rel="nofollow"linktype="raw" linktext="https://issues.apache.org/jira/browse/QPID-1056 :"> QPID-1056 :">https://issues.apache.org/jira/browse/QPID-1056 :</p> <p>Python examples for the xml exchange. QPID-1057 " title="Visit page outside Confluence" rel="nofollow"linktype="raw" linktext="https://issues.apache.org/jira/browse/QPID-1057 "> QPID-1057 " title="Visit page outside Confluence" rel="nofollow"linktype="raw" linktext="https://issues.apache.org/jira/browse/QPID-1057 "></p> <p>QPID-1057 ">https://issues.apache.org/jira/browse/QPID-1057 Fixes to the XmlExchange.cpp that preve ...</p>	
r656023	gsim	2008-05-13	QPID-990 : Patch from Ted Ross to enable persisting of inter-broker routing entities	
r656071	rajith	2008-05-14	Modified the verify_java_python.in file to reflect the changes made to the python code in rev 655965. The change was to correct a few typos.	
r656255	aconway	2008-05-14	Fixed python/examples/xml-exchange verify script.	
r656299	aidan	2008-05-14	add ignore file	
r656301	aidan	2008-05-14	Merge branch 'python-mllib'	Should have had Jira
r656320	aconway	2008-05-14	Fix valgrind problems in VPATH builds.	
r656326	aconway	2008-05-14	Exclude XML example checks if XML support is not available.	
r656328	aconway	2008-05-14	svn:ignore properties.	
r656331	gsim	2008-05-14	Fix for large messages.	
r656335	gsim	2008-05-14	Don't fail if python tests aren't found.	
r656357	rhs	2008-05-14	QPID-965 : made the ant build report the cumulative success/failure of the test suite	
r656369	aconway	2008-05-14	Undo revision 656320, causing build problems.	
r656373	aconway	2008-05-14	Exclude XML example if XML not enabled.	
r656376	cctrielloff	2008-05-14	Added requires for XML exchange	
r656427	aconway	2008-05-14	Fix example check in rpmbuild.	
r656443	aconway	2008-05-14	Python tests running in rpmbuild.	
r656689	rhs	2008-05-15	added .class files to svn:ignore for common	
r656690	rhs	2008-05-15	fixed a typo in 010ExcludeList	
r656760	rhs	2008-05-15	QPID-1062 : phase 1 of improvements to 0-10 encode/decode; this inlines the read/write method of structs into generated code resulting in roughly a 2x improvement	
r656766	nsantos	2008-05-15	change ordering of config records in management updates; patch supplied by Ted Ross	
r656849	rgodfrey	2008-05-15	Fixed credit restoration, turned off biased write pool by default, removed unused lock from queue	
r656853	aconway	2008-05-15	<ul style="list-style-type: none"> Enable python tests and examples in make rpmbuild. - Remove hard-coded amqp.xml paths from python examples. 	
r656855	kpvdr	2008-05-15	Patch from michael goulis: QPID-1063 : "under Boost 103200, command line args with = didn't work"	
r656859	aconway	2008-05-15	<ul style="list-style-type: none"> Remove redundant comments about AMQP_SPEC 	
r656871	rhs	2008-05-15	QPID-1064 : made qpid-config close the session/connection; added incoming.stop() to cancel incoming messages and join on the listener thread; made managementBroker.removeChannel use incoming.stop(); mo ...	

r656918	nsantos	2008-05-16	QPID-1061 : Management heartbeat message from broker - applied patch supplied by Ted Ross		
r656920	nsantos	2008-05-16	QPID-1065 : Management messages may lost if client attach hits a small time window - patch supplied by Ted Ross		
r656924	cctrielloff	2008-05-16	QPID-1034 by Senaka Fernando		
r656926	cctrielloff	2008-05-16	QPID-1017 by Senaka Fernando - with a few small edits		

Recently Raised JIRAs

Key	Component(s)	Affects Version/s	Summary	Status	Assignee	Reporter	Review Comments
QPID-1038	Dot Net Client	M2.1, M3	(QPID-1038) Blocking I/O in transport layer leads to poor performance for mixed sender/producer	Open	Aidan Skinner	Aidan Skinner	
QPID-1039	C++ Broker		(QPID-1039) fix program options behavior for Boost 103200	Closed	Unassigned	michael goulish	
QPID-1040	C++ Broker	M3	(QPID-1040) Asynchronous Protocol Connector for C++ broker	Closed	Andrew Stitzer	Ted Ross	
QPID-1041	Python Client	M3	(QPID-1041) Sources and sinks should all have names (a problem in at least the Python client)	Open	Rafael H. Schlorning	Jonathan Robie	
QPID-1042	C++ Broker	M3	(QPID-1042) Record of deliveries not kept for non-acquired, non-accepted subscriptions in credit flow mode	Closed	Gordon Sim	Gordon Sim	
QPID-1043	website	M3	(QPID-1043) Add page with a clear grid of protocol support for each qpid component	Open	Gordon Sim	Gordon Sim	
QPID-1044			(QPID-1044) Python examples need updating for tutorial	Open	Unassigned	Jonathan Robie	
QPID-1045	Python Client	M3	(QPID-1045) incoming message queues are not always notified of session closure	Resolved	Rafael H. Schlorning	Rafael H. Schlorning	
QPID-1046	Python Client	M3	(QPID-1046) Provide easy way to start incoming message flow	Resolved	Rafael H. Schlorning	Justin Ross	
QPID-1047			(QPID-1047) Qpidc.spec.in is missing dependencies and has misplaced files	Open	Unassigned	Matthew Farrellee	
QPID-1048	C++ Broker	M3	(QPID-1048) Dispatch of durable message from non-durable queue held up if message is also enqueued asynchronously on durable queue	Resolved	Gordon Sim	Gordon Sim	
QPID-1049	Java Broker	M2, M2.1	(QPID-1049) Unnecessary WARN level logging in DestWildExchange	Open	Unassigned	Martin Ritchie	
QPID-1050	C++ Broker	M3	(QPID-1050) Durability of federation config, other miscellaneous fixes	Closed	Unassigned	Ted Ross	
QPID-1051	C++ Client		(QPID-1051) The C++ examples contain some typos and other mistakes that will confuse users. These are usability errors	Open	Alan Conway	William Henry	
QPID-1052	C++ Broker, Python Client	M3	(QPID-1052) Management: session.name matches session id provided by API, handling of lost connections	Closed	Unassigned	Ted Ross	
QPID-1053	Java Client	M3	(QPID-1053) QpidTestCase should use something smarter than Thread.sleep(1000) to ensure that an external broker is listening on the port	Resolved	Rafael H. Schlorning	Rafael H. Schlorning	
QPID-1054	C++ Broker	M3	(QPID-1054) Startup failures not reported when in daemon mode	Resolved	Gordon Sim	Gordon Sim	
QPID-1055	Python Client	M3	(QPID-1055) map codec failures on 64 bit systems	Resolved	Rafael H. Schlorning	Rafael H. Schlorning	
QPID-1056	Python Client		(QPID-1056) XML Exchange - Python example	Resolved	Unassigned	Jonathan Robie	

QPID-1057	C++ Broker		(QPID-1057) Fix XML Exchange for Python client (which does not supply an empty FieldTable if there are no application message headers)	Resolved	Unassigned	Jonathan Robie	
QPID-1058	Dot Net Client	M2, M2.1	(QPID-1058) Add support for CRAM-MD5-HASHED as used by Java Broker	Open	Unassigned	Martin Ritchie	
QPID-1059	Code Generator	M3	(QPID-1059) mllib passes absolute windows path as URL	Open	Aidan Skinner	Aidan Skinner	
QPID-1060	Java Broker	M2.1	(QPID-1060) (Java Broker) OutOfMemory due to continued reference to ContentHeaderBody on Persistent Messages	Open	Rob Godfrey	Rob Godfrey	
QPID-1061	C++ Broker, Python Client	M3	(QPID-1061) Management heartbeat message from broker	Closed	Unassigned	Ted Ross	
QPID-1062	Java Client	M3	(QPID-1062) improve performance of 0-10 encode/decode	Open	Rafael H. Schlorning	Rafael H. Schlorning	
QPID-1063	C++ Broker		(QPID-1063) under Boost 103200, command line args with "=" didn't work	Open	Unassigned	michael goulis	
QPID-1064	python tools	M3	(QPID-1064) intermittent background thread stack trace when using qpid-config	Open	Rafael H. Schlorning	Rafael H. Schlorning	
QPID-1065	C++ Broker	M3	(QPID-1065) Management messages may lost if client attach hits a small time window	Closed	Unassigned	Ted Ross	
QPID-1066	Java Broker	M2, M2.1	(QPID-1066) cleanMainQueue call mistakenly wrapped in isInfoEnabled()	Open	Martin Ritchie	Martin Ritchie	

Qpid Java Meeting Minutes 2008-05-23

Review of Code Commits

revision	committer	date	comment	review comments
r657064	aconway	2008-05-16	Make rpmbuild python tests work under old versions of automake.	
r657069	cctrielloff	2008-05-16	QPID-1067 by tross	
r657088	cctrielloff	2008-05-16	QPID-1067	
r657097	rgodfrey	2008-05-16	QPID-1060 : Release ref to transient meta data; cache message size	
r657101	aidan	2008-05-16	Merged revisions 653416 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/branches/M2.x r653416 - aidan - 2008-05-05 11:24:50 +0100 (Mon, 05 May 2008) - 1 line QPID-1 ...	
r657106	aidan	2008-05-16	Initialized merge tracking via "svnmerge" with revisions "651431" from https://svn.apache.org/repos/asf/incubator/qpid/trunk	
r657107	aidan	2008-05-16	Initialized merge tracking via "svnmerge" with revisions "651431" from https://svn.apache.org/repos/asf/incubator/qpid/trunk/qpid	
r657111	aidan	2008-05-16	Merged revisions 657097 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/trunk/qpid r657097 - rgodfrey - 2008-05-16 16:08:55 +0100 (Fri, 16 May 2008) - 1 line QPID-1 ...	
r657112	rhs	2008-05-16	QPID-947 : initialize docstrings for protocol methods from the spec	
r657116	aidan	2008-05-16	Merged revisions 657111 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/branches/M2.x r657111 - aidan - 2008-05-16 16:53:21 +0100 (Fri, 16 May 2008) - 9 lines ...	
r657168	aconway	2008-05-16	Added missing log/Logger.h to headers.	

r657191	rhs	2008-05-16	QPID-947 : restrict docstring initialization to recent python versions	
r657820	tross	2008-05-19	QPID-1071	
r657827	rgodfrey	2008-05-19	Refactoring perf. tweaks	
r657859	ritchiem	2008-05-19	QPID-1066 : Removed isInfo wrapping. Added test that is missing from trunk from M2.x QueueDepthSelectorTest.	
r657974	tross	2008-05-19	QPID-1073	
r658166	ritchiem	2008-05-20	Merged revisions 657859 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/trunk/qpid r657859 - ritchiem - 2008-05-19 17:54:06 +0100 (Mon, 19 May 2008) - 1 line QPID-1 ...	
r658175	ritchiem	2008-05-20	Merged revisions 658166 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/branches/M2.x r658166 - ritchiem - 2008-05-20 09:37:33 +0100 (Tue, 20 May 2008) - 9 lines ...	
r658246	aconway	2008-05-20	Support for AMQP 0-10 sessions in C++ broker.	
r658278	ritchiem	2008-05-20	Merged revisions 648740 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/branches/M2.x r648740 - ritchiem - 2008-04-16 17:29:07 +0100 (Wed, 16 Apr 2008) - 1 line QPI ...	
r658403	aconway	2008-05-20	Fix build error.	
r658614	rgodfrey	2008-05-21	QPID-1084 : Fix AMQSession race condition on no-ack flow control	
r658689	arnaudsimon	2008-05-21	QPID-1086 : changed session.flush confirmed to do the same than for session.flush completed	
r658816	aconway	2008-05-21	Replaced AtomicCount with AtomicValue template. Uses gcc atomics for gcc on i686/x86_64, falls back to mutex otherwise.	
r658849	aconway	2008-05-21	Added check to exclude old gcc compilers for atomic ops.	
r658886	tross	2008-05-21	QPID-1087	
r659083	arnaudsimon	2008-05-22	QPID-1079 : Updated ...test.client tests for using QpidTestCase	DupsOKTest: should also check that all messages have arrived! If timeout expires it should fail. Is QpidTestCase not clearing application registry between tests: Martin to raise
r659101	aidan	2008-05-22	Add more files	
r659105	aidan	2008-05-22	QPID-1085 : If an error occurs creating a durable subscriber with a selector delete the queue that was created.	0-10 : makes the client blow as the queue doesn't exist. Also how is delete sent when channel should already be closed. Also why almost duplicated code for createDurableSubscription between 0-8 and 0-10. broker appears to be processing queue delete on a closed channel?! - Raise JIRA queue delete cannot be issued after failure because channel should be closed!
r659110	tross	2008-05-22	QPID-1088	
r659127	aconway	2008-05-22	Improved logging for session state.	
r659139	aconway	2008-05-22	Improved logging for session state - show incomplete commands on receive-completed.	

r659163	arnaudsimon	2008-05-22	QPID-1079 : Updated ...test.client tests for using QpidTestCase + move QpidTestCase in main so it is visible from sys-tests	Remove amqj.AutoCreateVMBroker Investigate if tests aren't working on trunk without this setting Remove VMTestCase? Remove all classes that were no longer referenced (VMBrokerSetup) RECEIVE_TIMEOUT : get rid of and use configurable timeout when available move kills to base case
r659165	arnaudsimon	2008-05-22	QPID-1079 : Updated TTLTest to use QpidTestCase	
r659186	tross	2008-05-22	Forbid broker to route to self, default to localhost when not specified	
r659262	arnaudsimon	2008-05-22	QPID-1079 : added junit dep to client as it's not included within all environments (for example on RHEL-4)	Should move QpidTestCase into common base package; not in client.
r659271	rhs	2008-05-22	Made Range, RangeSet, and Session all use proper RFC1982 comparisons per QPID-861 . Also switched command ids from long -> int, and added a mutex to channel to prevent multi-frame commands from interleave ...	Use .intValue() not (int) (long)
r659477	arnaudsimon	2008-05-23	QPID-1089 : Changed to use coountdownlatch	

Recently Raised JIRAs

Key	Component(s)	Affects Version/s	Summary	Status	Assignee	Repo
QPID-1061	C++ Broker, Python Client	M3	(QPID-1061) Management heartbeat message from broker	Closed	Unassigned	Ted R
QPID-1062	Java Client	M3	(QPID-1062) improve performance of 0-10 encode/decode	Open	Rafael H. Schlorin	Rafael Schlorin
QPID-1063	C++ Broker		(QPID-1063) under Boost 103200, command line args with "=" didn't work	Open	Unassigned	micha goulis
QPID-1064	python tools	M3	(QPID-1064) intermittent background thread stack trace when using qpid-config	Open	Rafael H. Schlorin	Rafael Schlorin
QPID-1065	C++ Broker	M3	(QPID-1065) Management messages may lost if client attach hits a small time window	Closed	Unassigned	Ted R
QPID-1066	Java Broker	M2, M2.1	(QPID-1066) cleanMainQueue call mistakenly wrapped in isInfoEnabled()	Resolved	Martin Ritchie	Martin Ritchie
QPID-1067	C++ Broker	M3	(QPID-1067) Minor improvements/fixes for command line utilities	Closed	Unassigned	Ted R
QPID-1068	C++ Broker, C++ Client	M3	(QPID-1068) C++ configure fails if help2man not present	Open	Andrew Sticher	Steve Huston
QPID-1069	C++ Broker, C++ Client	M3	(QPID-1069) Patch to build trunk with Boost 1.35	Open	Andrew Sticher	Steve Huston
QPID-1070	C++ Broker, C++ Client	M3	(QPID-1070) Patch to allow configure options to set Boost header and lib locations	Closed	Andrew Sticher	Steve Huston
QPID-1071	C++ Broker	M3	(QPID-1071) Publish interval for management can be set to zero	Resolved	Ted Ross	Ted R
QPID-1072	Java Client, Java Common	M3	(QPID-1072) org.apache.qpid packages should be moved under org.apache.qpid	Open	Rafael H. Schlorin	Rafael Schlorin
QPID-1073	C++ Broker	M3	(QPID-1073) Broker deadlock in management code	Resolved	Ted Ross	Ted R
QPID-1074	Java Broker		(QPID-1074) Log Configuration Watcher in Java Broker halts on invalid log4j.xml	Open	Unassigned	Senaka Ferna
QPID-1075	Java Broker		(QPID-1075) README on Java Broker	Open	Unassigned	Senaka Ferna
QPID-1076	Java Broker	M3	(QPID-1076) Cannot run Java Broker built on cygwin from trunk release due to qpid-run format issue	Open	Marnie McCormack	Marnie McCormack

QPID-1077	Ant Build System	M3	(QPID-1077) Permissions incorrect on bin directory on trunk release	Open	Unassigned	Marni McCo
QPID-1078	Ant Build System	M3	(QPID-1078) Ant generated qpid-incubating.jar has the wrong paths	Open	Rafael H. Schloeming	Aidan Skinner
QPID-1079	Java Tests	M3	(QPID-1079) Tests from Systests module should extend QpidTestCase	Open	Arnaud Simon	Arnaud Simon
QPID-1080	Java Broker, Java Client	M3	(QPID-1080) (0.8 code path) org.apache.qpid.test.unit.topic.TopicSessionTest#testNoLocal intermittently failing	Open	Unassigned	Arnaud Simon
QPID-1081	Java Client	M3	(QPID-1081) org.apache.qpid.test.client.QueueBrowserAutoAckTest and org.apache.qpid.test.client.QueueBrowserNoAckTest are intermittently failing	Open	Unassigned	Arnaud Simon
QPID-1082	Java Tests	M3	(QPID-1082) Add test case to ensure that config of housekeeping.expiredMessageCheckPeriod to 0 disables housekeeping process	Open	Aidan Skinner	Aidan Skinner
QPID-1083			(QPID-1083) Qpid Messaging Tutorial	Open	Unassigned	Jonathan Robie
QPID-1084	Java Client	M2, M2.1, M3	(QPID-1084) (Java Client) Race condition suspending channel in no-ack flow control situations	Open	Rob Godfrey	Rob Godfrey
QPID-1085	Java Broker	M3	(QPID-1085) Creating an invalid subscriber with a durable subscription still creates the subscription	Open	Aidan Skinner	Aidan Skinner
QPID-1086	Java Client	M3	(QPID-1086) (01.0 code path) Client does not implement session.flush confirmed	Resolved	Unassigned	Arnaud Simon
QPID-1087	C++ Broker	M3	(QPID-1087) Improvements to inter-broker federation	Closed	Ted Ross	Ted Ross
QPID-1088	C++ Broker	M3	(QPID-1088) Locking cleanup for management objects	Open	Ted Ross	Ted Ross
QPID-1089	Java Tests	M3	(QPID-1089) Test FieldTableMessageTest and TextMessageTest may hang	Resolved	Arnaud Simon	Arnaud Simon

Qpid Java Meeting Minutes 2008-05-30

Attendees

Agenda

Use of MINA Protect-I/O mode as default

Commits review

JIRA Review

Outstanding actions

revision	committer	date	comment	review comments
r659105	aidan	2008-05-22	QPID-1085 : If an error occurs creating a durable subscriber with a selector delete the queue that was created.	0-10 : makes the client blow as the queue doesn't exist. Also how is delete sent when channel should already be closed. Also why almost duplicated code for createDurableSubscription between 0-8 and 0-10. broker appears to be processing queue delete on a closed channel?! - Raise JIRA queue delete cannot be issued after failure because channel should be closed!

r659163	arnaudsimon	2008-05-22	QPID-1079 : Updated ...test.client tests for using QpidTestCase + move QpidTestCase in main so it is visible from systests	Remove amqj.AutoCreateVMBroker Investigate if tests aren't working on trunk without this setting Remove VMTTestCase? Remove all classes that were no longer referenced (VMBrokerSetup) RECEIVE_TIMEOUT : get rid of and use configurable timeout when available move kills to base case
r659262	arnaudsimon	2008-05-22	QPID-1079 : added junit dep to client as it's not included within all environments (for example on RHEL-4)	Should move QpidTestCase into common base package; not in client.
r659271	rhs	2008-05-22	Made Range, RangeSet, and Session all use proper RFC1982 comparisons per QPID-861 . Also switched command ids from long -> int, and added a mutex to channel to prevent multi-frame commands from interleave ...	Use .intValue() not (int) (long)
r659631	rhs	2008-05-23	QPID-901 : Track and report session exceptions, modified generator validate values before trying to encode them. Also, moved createDurableSubscriber from AMQSession_0_10 -> AMQSession.	
r659647	rhs	2008-05-23	QPID-947 : Switched over to using proper RFC 1982 serial numbers.	
r659650	rhs	2008-05-23	QPID-1064 : only set the listeners to None after the thread has stopped	
r659671	rhs	2008-05-23	QPID-947 : added codec and tests for array and list types	
r659673	rhs	2008-05-23	QPID-947 : added test for nested lists	

Outstanding actions need to be carried forward in future

Use of Protect-I/O mode as default

Needs more testing before it is set as default in release, decision to be taken on list.

Review of Code Commits

revision	committer	date	comment	
r659535	tross	2008-05-23	qpid-tool fixed to cleanly handle brokers with management disabled	
r659538	aconway	2008-05-23	qpid::SessionState: Added error checking for invalid frame sequences. client: Fix client crash on error during connection shutdown.	
r659631	rhs	2008-05-23	QPID-901 : Track and report session exceptions, modified generator validate values before trying to encode them. Also, moved createDurableSubscriber from AMQSession_0_10 -> AMQSession.	
r659647	rhs	2008-05-23	QPID-947 : Switched over to using proper RFC 1982 serial numbers.	
r659650	rhs	2008-05-23	QPID-1064 : only set the listeners to None after the thread has stopped	
r659663	aconway	2008-05-23	Delete obsolete Channel class.	
r659671	rhs	2008-05-23	QPID-947 : added codec and tests for array and list types	
r659673	rhs	2008-05-23	QPID-947 : added test for nested lists	
r660173	aconway	2008-05-26	Fix compile error in examples.	
r660254	rajith	2008-05-26	I am applying the patch provided by Senaka attached to QPID-968 We need to get rid of the defaults that points to rhm.	
r660258	aconway	2008-05-26	Changes to Session API: - Session is synchronous, no futures. - AsyncSession is async, returns futures. - Conversion functions sync(s) async(s) return a sync/async view of session s. - Connection ...	
r660265	aconway	2008-05-26	Corrected examples for new session API.	

r660302	aconway	2008-05-26	Speculative "fix" for solaris compile errors discussed on qpid-dev.	
r660304	aconway	2008-05-26	Removed BOOST_REQUIRE_EQUAL, not available in older boost.test.	
r660320	aconway	2008-05-26	Fixed intermittent leak of client::Connector thread.	
r660324	aconway	2008-05-26	Make help2man and doxygen dependencies optional.	
r660490	rgodfrey	2008-05-27	Refactoring updates (job queue changes, enqueue collections..)	
r660494	rgodfrey	2008-05-27	Missed one...	
r660546	gsim	2008-05-27	Added some comments to the various connection settings.	
r660562	aconway	2008-05-27	Additional API documentation around sync vs. async sessions.	
r660568	aconway	2008-05-27	Copy valgrind support files in a VPATH builds.	
r660616	aconway	2008-05-27	Generate code in \$builddir to allow multiple VPATH builds.	
r660619	aconway	2008-05-27	VPATH fix	
r660625	aconway	2008-05-27	Use symbolic constants for message flow values.	
r660643	aconway	2008-05-27	Tighten up sync-correctness in SubscriptionManager & Dispatcher. Add a flush to SessionBase_0_10::sync() so it syncs in both directions.	
r660647	aconway	2008-05-27	Fixed error in RangeSet, caused compile failure on Solaris.	
r660715	aconway	2008-05-27	Removed obsolete src/qpid/client/SessionImpl.h .cpp	
r660886	gsim	2008-05-28	Fixes to binding of member functions as raised on qpid list by Manuel Teira.	
r660911	arnaudsimon	2008-05-28	QPID-1094 : Implement XA resource exception handling and add corresponding tests	
r660922	arnaudsimon	2008-05-28	QPID-1097 : Those changes have been suggested by Lana	Needs more detail, who is Lana?
r660924	gsim	2008-05-28	QPID-1095 : fixes to dtx error codes for latest spec changes.	
r660952	gsim	2008-05-28	QPID-1095 : another error code correction	
r660953	gsim	2008-05-28	QPID-1098 : correction to queue query when queue is not known	
r660966	aidan	2008-05-28	QPID-1099 Add tests for publishing several messages transactionally and consuming them in an OnMessage handler	
r660973	arnaudsimon	2008-05-28	QPID-1094 and QPID-1095 : Updated XaResource for handling wrong flag value, updated xa tests for using correct flag values, excluded forget test as the current 0.10 broker does not implement forget.	
r660981	gsim	2008-05-28	Improve latency test tool to allow lower rates.	
r661000	gsim	2008-05-28	Updated some 'todo' comments with clearer text.	
r661267	arnaudsimon	2008-05-29	QPID-1094 : added finally close for cleaning potential indoubt tx	
r661286	rgodfrey	2008-05-29	Avoid NPEs 	
r661287	rgodfrey	2008-05-29	Deliver async per subscription; not queue	
r661296	rgodfrey	2008-05-29	tidy up	
r661302	gsim	2008-05-29	Only record frames for replay if timeout is non-zero.	
r661309	gsim	2008-05-29	Move AckPolicy impl from header to .cpp; ensure that completion is marked even when auto-acking is turned off.	
r661323	gsim	2008-05-29	Correct declarations to be the same as definitions.	
r661324	rgodfrey	2008-05-29	Temp fix out of order issue with async(sub)	
r661325	rgodfrey	2008-05-29	Made subscription sendLock straight lock, re-enabled per subscription async delivery	
r661326	rgodfrey	2008-05-29	Fix SubscriptionTestHelper	

r661395	rgodfrey	2008-05-29	Comments and changes from review		
r661405	rgodfrey	2008-05-29	fix browser behaviour on deliverAsync(sub)		
r661445	aconway	2008-05-29	Packaing error - SessionId.h		
r661455	cctrieloff	2008-05-29	Performnace fix, improves 6-8% on high core count machines.		
r661483	tross	2008-05-29	QPID-1100 crash when config file contains commented-out commands (patch from michael goulish)		
r661561	rajith	2008-05-30	This check in is for QPID-1102 . IoHandler and IoSender uses the java.io classes for IO operations and have shown very good improvement in latency and memory usage over MINA. For certain tests with pub ...	Should use configuration and be more configurable. Narrow catch Exception clause.	
r661587	gsim	2008-05-30	Convert remaining CPPUNIT tests to boost test framework to reduce dependencies.		
r661633	arnaudsimon	2008-05-30	QPID-754 : changed prop name		

Review of Recently Raised JIRAs

Key	Component(s)	Affects Version/s	Summary	Status	Assignee	Reporter	Review Comments
QPID-1090	C++ Broker	M3	(QPID-1090) Removed pid storage for daemon mode - rely on external mechanisms	In Progress	Ted Ross	Ted Ross	
QPID-1091	Java Tests	M3	(QPID-1091) QpidTestCase does not clean up the InVM Broker after each test.	Open	Unassigned	Martin Ritchie	MR to raise Jira to clean message stores between test runs
QPID-1092	Java Client	M2, M2.1	(QPID-1092) JMSObjectMessage.toString() corrupts the ByteBuffer if the data is not a String	Open	Unassigned	Martin Ritchie	MR to assign to M3 as well
QPID-1093	Java Broker	M2.1	(QPID-1093) (Java Broker) Meta data not found when registering a consumer using a selectors	Open	Unassigned	Rob Godfrey	
QPID-1094	Java Client	M3	(QPID-1094) xaResource does not correctly handle exceptions	Open	Arnaud Simon	Arnaud Simon	
QPID-1095	C++ Broker	M3	(QPID-1095) dtx class does not fail with expected error codes	Closed	Gordon Sim	Arnaud Simon	
QPID-1096	C++ Broker	M3	(QPID-1096) Exception codes for dtx need to be updated in line with final 0-10 spec	Open	Gordon Sim	Gordon Sim	
QPID-1097	Java Client	M3	(QPID-1097) Need to improve 0.10 interface javadoc	Open	Arnaud Simon	Arnaud Simon	
QPID-1098	C++ Broker	M3	(QPID-1098) Final 0-10 specification requires empty struct to be returned if queried queue is not found	Resolved	Gordon Sim	Gordon Sim	
QPID-1099	Dot Net Client	M3	(QPID-1099) Add tests for checking transactional message consumption using OnMessage	Open	Aidan Skinner	Aidan Skinner	
QPID-1100	C++ Broker	M3	(QPID-1100) crash when config file contains commented-out commands.	Open	Ted Ross	michael goulish	
QPID-1101	Java Broker	M2, M2.1	(QPID-1101) DestWildExchange uses shallow copy of queues for routing, causing routing to fail if queueDeleted.	Open	Unassigned	Martin Ritchie	
QPID-1102	C++ Client	M3	(QPID-1102) A new java.io based blocking transport for client	Open	Rajith Attapattu	Rajith Attapattu	Java client component, should be resolved.

QPID-1103	Java Tests		(QPID-1103) Add ability to provide a configuration file for broker start up in java test suite	Open	Martin Ritchie	Martin Ritchie	
QPID-1104	Dot Net Client	M2.1	(QPID-1104) (.Net) Allow acknowledgement of a single message	Open	Unassigned	Marnie McCormack	
QPID-1105	C++ Broker, C++ Client, Code Generator	M3	(QPID-1105) Patches and additions to port to Windows	Open	Unassigned	Steve Huston	
QPID-1106		M3	(QPID-1106) Make message acknowledgment synchronous	Open	Unassigned	Arnaud Simon	Needs component.

Qpid Java Meeting Minutes 2008-06-20

Outstanding actions

Use of MINA Protect-I/O mode as default

Commits review

JIRA Review

Outstanding actions

revision	committer	date	comment	review comments
r659163	arnaudsimon	2008-05-22	QPID-1079 : Updated ...test.client tests for using QpidTestCase + move QpidTestCase in main so it is visible from sys tests	Remove amqj.AutoCreateVMBroker Investigate if tests aren't working on trunk without this setting Remove VMTestCase? Remove all classes that were no longer referenced (VMBrokerSetup) RECEIVE_TIMEOUT : get rid of and use configurable timeout when available move kills to base case
r659262	arnaudsimon	2008-05-22	QPID-1079 : added junit dep to client as it's not included within all environments (for example on RHEL-4)	Should move QpidTestCase into common base package; not in client.
r659271	rhs	2008-05-22	Made Range, RangeSet, and Session all use proper RFC1982 comparisons per QPID-861 . Also switched command ids from long -> int, and added a mutex to channel to prevent multi-frame commands from interleave ...	Use .intValue() not (int) (long)
r659631	rhs	2008-05-23	QPID-901 : Track and report session exceptions, modified generator validate values before trying to encode them. Also, moved createDurableSubscriber from AMQSession_0_10 -> AMQSession.	
r659647	rhs	2008-05-23	QPID-947 : Switched over to using proper RFC 1982 serial numbers.	
r659650	rhs	2008-05-23	QPID-1064 : only set the listeners to None after the thread has stopped	
r659671	rhs	2008-05-23	QPID-947 : added codec and tests for array and list types	
r659673	rhs	2008-05-23	QPID-947 : added test for nested lists	
r661561	rajith	2008-05-30	This check in is for QPID-1102 . IoHandler and IoSender uses the java.io classes for IO operations and have shown very good improvement in latency and memory usage over MINA. For certain tests with pub ...	Should use configuration and be more configurable. Narrow catch Exception clause.
r669431	rgodfrey	2008-06-19	QPID-950 : Broker refactoring, copied / merged from branch	
r669480	rgodfrey	2008-06-19	QPID-950 : Fixed Derby Message Store	
r669841	rgodfrey	2008-06-20	QPID-1144 : Reference count drops to zero too early for immediate messages in a txn	

revision	committer	date	comment	Review Comments
----------	-----------	------	---------	-----------------

r661698	gsim	2008-05-30	Add short sleep before killing python server to ensure it has had a chance to send the message acknowledgement.	
r661730	gsim	2008-05-30	Removed redundant flush request.	
r661739	ritchiem	2008-05-30	QPID-1103 :Changed VMTestCase to allow the creation of InVM brokers based on a configuration file. Updated ApplicationRegistry as it was not correctly utilising the set configuration and always using ...	
r661746	ritchiem	2008-05-30	QPID-1101 : Update to DestNameExchange to perform deep copy.	Needs test
r662310	aconway	2008-06-02	Use InlineVector for AMQFrame, reduces heap allocations by 13%.	
r662373	gsim	2008-06-02	Minor updates to tests: * sync on commit in transactional topic test * disable loading of modules from automated test to preserve isolation * update federated topic test script in line with command li ...	
r662390	gsim	2008-06-02	disable use of module-dir when running examples	
r662397	arnaudsimon	2008-06-02	QPID-1110 : use pre-acquire mode when message selector is the empty string	
r662461	aconway	2008-06-02	Fix compiler warning with gcc 4.3	
r662467	ritchiem	2008-06-02	Merged revisions 661739-661746 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/branches/M2.x r661739 - ritchiem - 2008-05-30 15:42:38 +0100 (Fri, 30 May 2008) - 1 line ...	
r662470	tross	2008-06-02	QPID-1113 Management cleanup and performance enhancements	
r662472	aconway	2008-06-02	Backed out previous fix compiler for warning, it fails with boost 1.33. Will seek a fix that works for all versions.	
r662490	tross	2008-06-02	Fixed dereference of null pointer	
r662497	aconway	2008-06-02	Fix that works on 1.33/gcc4.1 up to boost 1.34.13/gcc 4.3	
r662503	aidan	2008-06-02	Remove old doc	
r662505	aidan	2008-06-02	Move to documentation directory	
r662506	aidan	2008-06-02	Remove stuff thats been moved	
r662507	aidan	2008-06-02	Add generated forrest structure	
r662508	aidan	2008-06-02	Move to proper location	
r662558	aconway	2008-06-02	Added --syslog-name, --syslog-facility options.	
r662561	gsim	2008-06-02	Improve performance of synchronous publication by not requesting known-completed response for every completed sent.	
r662570	tross	2008-06-02	QPID-1114 Daemon mode improvements	
r662581	aconway	2008-06-02	Separate option parsing from qpid::client::ClientSettings.	
r662588	tross	2008-06-02	Added byteDepth back into Queue managment class	
r662592	tross	2008-06-02	Queue stats: byteDepth now computed periodically	
r662613	cctrielloff	2008-06-03	QPID-1108 patch from Manuel Teira	
r662665	arnaudsimon	2008-06-03	QPID-1112 : Added sessionCompleted support and changed onMessage for invoking sessionCompleted when all expected messages have been received.	
r662675	gsim	2008-06-03	Move ConnectionOptions into qpid::client.	
r662681	gsim	2008-06-03	Reverted move of ConnectionOptions (without the parse functionality they aren't off much use). Corrected include in ConnectionOptions.h	
r662700	gsim	2008-06-03	Add ConnectionOptions.h to sources for each test program.	
r662701	gsim	2008-06-03	Fixed typo in options.	

r662755	arnaudsimon	2008-06-03	QPID-1115 : Only generate client ID when necessary	RG to comment on Jira
r662770	ritchiem	2008-06-03	QPID-1092 : Changed toString to be String.valueOf(getObject()) Added MessageToStringTest, tests performing toString on Message before calling getObject().	Weird catch in close()
r662773	ritchiem	2008-06-03	Merged revisions 662770 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/branches/M2.x r662770 - ritchiem - 2008-06-03 13:32:47 +0100 (Tue, 03 Jun 2008) - 3 lines QP ...	
r662774	gsim	2008-06-03	Better exception handling for commit.	
r662818	ritchiem	2008-06-03	QPID-1117 : Added tests for all other message types. Refactored the common parts out of the objectTest.	
r662820	ritchiem	2008-06-03	Merged revisions 662818 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/branches/M2.x r662818 - ritchiem - 2008-06-03 16:07:07 +0100 (Tue, 03 Jun 2008) - 1 line QPI ...	
r662821	tross	2008-06-03	QPID-1114 moved --pid-dir from config file to startup script	
r662827	arnaudsimon	2008-06-03	QPID-1112 : Update previous commit by re-using messageAcknowledge (added a flag specifying whether to send an messageAccept)	inRecover check in BMC_0_10.postDeliver might be a problem with async delivery
r662830	tross	2008-06-03	Management fixes: set session.detachedLifetime to 0, set journal->queue link in all cases	
r662849	rhs	2008-06-03	QPID-1062 : modified generated code to keep packing flags in wire form and override commonly used size methods for improved performance	Remove commented out code
r662854	tross	2008-06-03	QPID-1114 Change defaults for data-dir and pid-dir to /home/ross/.qpid	
r662859	rhs	2008-06-03	QPID-901 : honor the timely-reply flag and handle known-completed	
r662869	aconway	2008-06-03	Use help2man if available, pre-generated qpidd.1 if available, fall back to dummy man page.	
r662900	tross	2008-06-03	Create pid-dir if it does not exist	
r663080	tross	2008-06-04	Removed assignment of a string literal that causes problems with some newer compilers	
r663124	arnaudsimon	2008-06-04	QPID-1120 : Changed addDeliveredMessage and commit so session.completed is sent before credits dry up	should send session complete when tx size is a multiple of PrefetchSize
r663125	ritchiem	2008-06-04	QPID-1119 : M2x commit : Addition of a System property to AMQProtocolHandler.java to allow the syncWait default to be changed. To perform this a new SlowMessageStore has been added to the systest pack ...	
r663142	ritchiem	2008-06-04	Merged revisions 663125 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/branches/M2.x r663125 - ritchiem - 2008-06-04 15:32:49 +0100 (Wed, 04 Jun 2008) - 6 lines QP ...	
r663158	aconway	2008-06-04	Avoid use of valgrind --log-file-exactly flag, removed in valgrind 3.3	
r663243	gsim	2008-06-04	Change to lazy-loading to avoid relying on the content-size to be set by client.	
r663271	aconway	2008-06-04	Increased default flush interval to 1MB, send spontaneous known-completed at the flush interval.	
r663304	tross	2008-06-04	QPID-1121	
r663318	aconway	2008-06-04	Request a timely reply to session.completed based on configured flush interval.	
r663325	rhs	2008-06-04	QPID-1062 : use BBDecoder for non fragmented segments, modified BBDecoder/Encoder to use byte buffer primitives, made various classes final (including generated classes)	

r663338	tross	2008-06-04	Management clean-up. Made the management broker more defensive with regard to received messages. Default and management exchanges now have 'durable' object IDs.	
r663340	aconway	2008-06-04	Fix valgrind error.	
r663351	aconway	2008-06-04	Remove unused classes IList and ISList.	
r663364	arnaudsimon	2008-06-04	QPID-1120 : don't reset batch size as part of the messages are not accepted and then still available.	
r663386	tross	2008-06-04	Management cleanup - Synchronized with the spec on the Wiki	
r663413	tross	2008-06-04	Management cleanup - renamed config/inst elements to properties and statistics	
r663507	arnaudsimon	2008-06-05	QPID-1123 : Added a timeout (threading issue is still to be fixed)	
r663519	gsim	2008-06-05	Re-introduced previously clobbered realm option.	
r663601	gsim	2008-06-05	Fix to makefile and tests (one test temporarily disabled until a fix is found).	
r663614	tross	2008-06-05	Load modules from /usr/lib64/qpidd on x86_64 architecture	
r663619	nsantos	2008-06-05	install libs in arch-appropriate directory	
r663621	nsantos	2008-06-05	install libs in arch-appropriate directory	
r663637	nsantos	2008-06-05	install libs in arch-appropriate directory	
r663653	nsantos	2008-06-05	install libs in arch-appropriate directory	
r663675	gsim	2008-06-05	cleanup old irrelevant tests (from 0-10 preview functions) fix dtx.recover test	
r663676	nsantos	2008-06-05	install libs in arch-appropriate directory	
r663677	rhs	2008-06-05	QPID-1116 : fixed a race condition in connection/session close, session close now waits for the session to be detached before returning, this guarantees we won't have any active sessions when the conne ...	Client.java has random cruft added.
r663730	kpvdr	2008-06-05	Minor additions to Range and RangedSet	
r663731	aconway	2008-06-05	Fixed bug in InlineAllocator	
r663742	gsim	2008-06-05	Uncomment test now that inline allocator is fixed.	
r663755	tross	2008-06-05	Dequeue persistent messages from store in queue purge	
r663761	aconway	2008-06-05	Modified to work with boost-1.32	
r663813	rhs	2008-06-06	QPID-1062 : merge writes of separate frames within an assembly, use sync flag instead of sync command on message transfer	
r663874	arnaudsimon	2008-06-06	QPID-1062 : use sync flag instead of sync command on tx commit	
r663999	ritchiem	2008-06-06	QPID-1058 : Added new CramMD5HexSaslClient.cs and registered it in the Sasl Factory and the client CallbackHandler	
r664001	ritchiem	2008-06-06	QPID-1058 : Addition of a CRAM-MD5-HEX as discussed on the JIRA. An additional test is provided to ensure that the handle method correctly wraps a given Database password in hex.	
r664020	ritchiem	2008-06-06	QPID-1058 : Removal of Console WriteLine as highlighted in code review by Robert Godfrey.	
r664028	ritchiem	2008-06-06	Merged revisions 663999-664020 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/branches/M2.x r663999 - ritchiem - 2008-06-06 17:03:42 +0100 (Fri, 06 Jun 2008) - 1 line ...	
r664112	tross	2008-06-06	Added mutexes back in to protect management counts from corruption	

r664114	aconway	2008-06-06	Added exceptions to sys::Waitable. Fixed client side deadlock involving client::Bounds. Fixed incorrect exception messages during connection shutdown.	
r664129	rgodfrey	2008-06-06	QPID-1124 : Use thread-safe map for messageListeners	
r664139	nsantos	2008-06-06	add missing header	
r664140	rhs	2008-06-06	QPID-1125 : log exceptions destined to be swallowed by MINA	
r664153	rgodfrey	2008-06-06	QPID-1124 : Use thread-safe map for messageListeners	
r664339	rhs	2008-06-07	QPID-1126 : reuse channel numbers for sessions that have closed, and honor the negotiated channel-max; also removed unnecessary catches that were swallowing stack traces from several tests	
r664695	aconway	2008-06-09	Missing lock in SessionManager::forget()	
r664698	arnaudsimon	2008-06-09	QPID-1127 : disable direct buffers as default.	
r665733	rhs	2008-06-09	QPID-901 : added logging of sync bit and command-id	
r665798	rhs	2008-06-09	QPID-901 : made logging of ids less expensive, also limit how much data we dump into the log	
r665841	rhs	2008-06-09	QPID-901 : always reset the auto-sync mode even if the call fails	RHS: should raise Jira for autosync flag
r665890	gsim	2008-06-09	Moved from AccumulatedAck to SequenceSet in managing transactional accepts Added transactional option to perftest Removed clientid from ConnectionSettings as it appears not to be used	
r665891	aconway	2008-06-09	Updated doxygen comments in qpid/client/*.h Changed request-response example to use SubscriptionManager like the others.	
r666051	gsim	2008-06-10	Improved exception handling for commit.	
r666138	gsim	2008-06-10	Removed import of deleted test modules.	
r666146	rhs	2008-06-10	updated the hello-world script	
r666244	rhs	2008-06-10	updated hello-world	
r666259	rhs	2008-06-10	QPID-1129 : unless otherwise specified, limit the receive buffer size to 64K	Mina makes OOM'ing hard to figure out
r666296	arnaudsimon	2008-06-10	Qpid-1130: don't store unack message tags when the session is transacted	
r666610	gsim	2008-06-11	<ul style="list-style-type: none"> make tcp-nodelay option available for all tests * option for outputting csv from latency test (from lgoncalv@redhat.com) * option for cumulative output from latency test (from lgoncalv@redhat.com) * ... 	
r666743	rhs	2008-06-11	load the old version of the spec file for old codec tests, removed unused test exclude list	
r666850	rhs	2008-06-11	replaced example usages of message_flow with the start() method	
r667015	arnaudsimon	2008-06-12	QPID-1134 : updated username:password --> guest:guest	
r667095	aidan	2008-06-12	Added ignore file	
r667096	aidan	2008-06-12	add generated bumpf	
r667097	aidan	2008-06-12	QPID-1135 : Fix multi-frame message handling. This fix is suboptimal since it creates an extra copy, as a result it's slower and less memory efficient. But it is correct. Qpid.Buffer/SlicedByteBuffer.c ...	
r667176	aconway	2008-06-12	Improvements to comment clarity.	
r667205	aconway	2008-06-12	Propagate error messages across the Demux between network & user threads.	

r667215	aconway	2008-06-12	Fix test error.	
r667217	nsantos	2008-06-12	add missing DESIGN file to Makefile.am	
r667253	aconway	2008-06-12	Default --log-output to syslog in --daemon mode.	
r667324	rajith	2008-06-13	Removed --store-async option as it is no longer relevant	
r667501	rhs	2008-06-13	QPID-901 : flush after every 64K commands issued	
r667503	aconway	2008-06-13	Fix bug in SessionState - avoid all replay calculations for timeout==0.	
r667540	rhs	2008-06-13	QPID-901 : don't send known-completed for ranges we ignore	
r667549	rajith	2008-06-13	Changed the store path back to what it was. I think we need to provide a better solution for this as making an assumption for the store path is inconvenient. We could use an env var like STORE_PATH tha ...	
r667554	aconway	2008-06-13	Revert SessionState changes in r667503.	
r667561	ritchiem	2008-06-13	QPID-1136 : Provided a fix for the leak in UnacknowledgedMessage when acking. Added a new InternalBrokerBaseCase for performing testing on the broker without using the client libraries. This allows fo ...	InternalMinaProtocolSession has a bug in awaitDelivery where it can hang because deliveryCount is already set to !0
r667574	ritchiem	2008-06-13	Merged revisions 667561 via svnmerge from https://svn.apache.org/repos/asf/incubator/qpid/branches/M2.x r667561 - ritchiem - 2008-06-13 15:56:45 +0100 (Fri, 13 Jun 2008) - 3 lines QP ...	
r667603	aconway	2008-06-13	Fix for broker wraparound problem.	
r667615	rhs	2008-06-13	QPID-901 : request known-completed every 64K incoming commands, fixed handling of incoming known-completed to clear out processed set	
r668151	tross	2008-06-16	Bugfix: usage line did not show with --help option	
r668164	aidan	2008-06-16	QPID-1104 : Add an IMessage.Acknowledge(bool) so that only specific messages can be acknowledged, not all messages received on the Channel up to that point. Qpid.Client/Client/Message/AbstractQmsMessa ...	
r668191	rhs	2008-06-16	QPID-1078 : fix the broken paths in qpid-incubating.jar and use the proper delimiter for manifest class paths	Still doesn't work on Windoze, AS and RHS to hug
r668308	rhs	2008-06-16	QPID-901 : set the frame track correctly	
r668309	rhs	2008-06-16	QPID-901 : add tests for RangeSet; fixed a bug found by the new tests	
r668311	rhs	2008-06-16	QPID-1139 : use RFC1982 comparisons for rollback mark and update rollback mark to track dispatched messages	
r668325	gsim	2008-06-16	QPID-1138 : codec support for timestamps	
r668333	rajith	2008-06-16	This is a fix for QPID-1140 and QPlid-1141. I also removed commented code as well as code that wasn't used. Cleaned up unused imports as well.	
r668343	rhs	2008-06-16	add enough to the server delegate to permit java clients to connect	
r668344	rhs	2008-06-16	QPID-1142 : made session.sync() always set the sync flag on execution_sync	
r668345	rhs	2008-06-16	QPID-1143 : added buffering, we now only issue one write per assembly	
r668378	rhs	2008-06-17	QPID-1143 : removed race condition in test	
r668582	gsim	2008-06-17	Added option to start topictest script in transactional mode (&durable)	
r668692	marnie	2008-06-17	QPID-168 Applying an amended version of Suran's patch for this JIRA, expanding the .bat file's functionality to be closer to the bash scripts	
r669215	gsim	2008-06-18	Fix bug that commits after every message. Oops!	

r669236	aconway	2008-06-18	Bring cluster code up to date.	
r669237	aconway	2008-06-18	Bring cluster code up to date.	
r669272	aconway	2008-06-18	Fix packaging error.	
r669430	rgodfrey	2008-06-19	Branch created from trunk prior to java refactor broker merge	
r669431	rgodfrey	2008-06-19	QPID-950 : Broker refactoring, copied / merged from branch	
r669480	rgodfrey	2008-06-19	QPID-950 : Fixed Derby Message Store	
r669841	rgodfrey	2008-06-20	QPID-1144 : Reference count drops to zero too early for immediate messages in a txn	

Qpid Java Meeting Minutes 2008-06-27

Outstanding actions

Commits review
JIRA Review

Outstanding actions

revision	committer	date	comment	review comments
r659163	arnaudsimon	2008-05-22	QPID-1079 : Updated ...test.client tests for using QpidTestCase + move QpidTestCase in main so it is visible from systests	Remove amqj.AutoCreateVMBroker Investigate if tests aren't working on trunk without this setting Remove VMTestCase? Remove all classes that were no longer referenced (VMBrokerSetup) RECEIVE_TIMEOUT : get rid of and use configurable timeout when available move kills to base case
r659262	arnaudsimon	2008-05-22	QPID-1079 : added junit dep to client as it's not included within all environments (for example on RHEL-4)	Should move QpidTestCase into common base package; not in client.
r659271	rhs	2008-05-22	Made Range, RangeSet, and Session all use proper RFC1982 comparisons per QPID-861 . Also switched command ids from long -> int, and added a mutex to channel to prevent multi-frame commands from interleave ...	Use .intValue() not (int) (long)
r659631	rhs	2008-05-23	QPID-901 : Track and report session exceptions, modified generator validate values before trying to encode them. Also, moved createDurableSubscriber from AMQSession_0_10 -> AMQSession.	
r659647	rhs	2008-05-23	QPID-947 : Switched over to using proper RFC 1982 serial numbers.	
r659650	rhs	2008-05-23	QPID-1064 : only set the listeners to None after the thread has stopped	
r659671	rhs	2008-05-23	QPID-947 : added codec and tests for array and list types	
r659673	rhs	2008-05-23	QPID-947 : added test for nested lists	
r661561	rajith	2008-05-30	This check in is for QPID-1102 . IoHandler and IoSender uses the java.io classes for IO operations and have shown very good improvement in latency and memory usage over MINA. For certain tests with pub ...	Should use configuration and be more configurable. Narrow catch Exception clause.
r669431	rgodfrey	2008-06-19	QPID-950 : Broker refactoring, copied / merged from branch	
r669480	rgodfrey	2008-06-19	QPID-950 : Fixed Derby Message Store	
r669841	rgodfrey	2008-06-20	QPID-1144 : Reference count drops to zero too early for immediate messages in a txn	
r661746	ritchiem	2008-05-30	QPID-1101 : Update to DestNameExchange to perform deep copy.	Needs test
r662755	arnaudsimon	2008-06-03	QPID-1115 : Only generate client ID when necessary	RG to comment on Jira

r662770	ritchiem	2008-06-03	QPID-1092 : Changed toString to be String.valueOf(getObject()) Added MessageToStringTest, tests performing toString on Message before calling getObject().	Weird catch in close()
r662827	arnaudsimon	2008-06-03	QPID-1112 : Update previous commit by re-using messageAcknowledge (added a flag specifying whether to send an messageAccept)	inRecover check in BMC_0_10.postDeliver might be a problem with async delivery
r662849	rhs	2008-06-03	QPID-1062 : modified generated code to keep packing flags in wire form and override commonly used size methods for improved performance	Remove commented out code
r663124	arnaudsimon	2008-06-04	QPID-1120 : Changed addDeliveredMessage and commit so session.completed is sent before credits dry up	should send session complete when tx size is a multiple of PrefetchSize
r663677	rhs	2008-06-05	QPID-1116 : fixed a race condition in connection/session close, session close now waits for the session to be detached before returning, this guarantees we won't have any active sessions when the conne ...	Client.java has random cruft added.
r665841	rhs	2008-06-09	QPID-901 : always reset the auto-sync mode even if the call fails	RHS: should raise Jira for autosync flag
r666259	rhs	2008-06-10	QPID-1129 : unless otherwise specified, limit the receive buffer size to 64K	Mina makes OOM'ing hard to figure out
r667561	ritchiem	2008-06-13	QPID-1136 : Provided a fix for the leak in UnacknowledgedMessage when acking. Added a new InternalBrokerBaseCase for performing testing on the broker without using the client libraries. This allows fo ...	InternalMinaProtocolSession has a bug in awaitDelivery where it can hang because deliveryCount is already set to !0
r668191	rhs	2008-06-16	QPID-1078 : fix the broken paths in qpid-incubating.jar and use the proper delimiter for manifest class paths	Still doesn't work on Windoze, AS and RHS to hug

New this week

revision	committer	date	comment
r669841	rgodfrey	2008-06-20	QPID-1144 : Reference count drops to zero too early for immediate messages in a txn
r669885	rgodfrey	2008-06-20	QPID-1101 : Updated Direct Exchange so it does not modify lists of queues
r669917	arnaudsimon	2008-06-20	QPID-1112 : send completed every maxPrefetch / 2 instead of after every messages once maxPrefetch / 2 has been reached
r670066	aconway	2008-06-20	Patch from Manuel Teira: https://issues.apache.org/jira/secure/CommentAssignIssue!default.jspa?action=5&id=12398038 - Use standard automake makefiles to build cpp/examples. - Rationalize examples di ...
r670089	astitcher	2008-06-20	QPID-1069 : Patch from Steve Huston: Build with Boost 1.35
r670568	aconway	2008-06-23	Fix build problems in examples on older automakes
r670571	aconway	2008-06-23	Fix path problems in examples make check
r670718	aconway	2008-06-23	Const-correctness fixes in MessageStore.h
r671491	gsim	2008-06-25	<ul style="list-style-type: none"> remove generated Makefile from svn * add back check for exclusions where xml exchange support is not available
r671519	aidan	2008-06-25	QPID-551 Don't take arbitrary stack lengths. I really hate that we sublist at all, it's gross.
r671553	rhs	2008-06-25	QPID-1078 : use file.separator so that globmapper generates the correct manifest class path on cygwin
r671604	aconway	2008-06-25	<ul style="list-style-type: none"> use flock to lock data dir rather than a lock file. - removed troublesome global constructor in Mutex initialization.
r671655	aconway	2008-06-25	Additions to the client API: - SubscriptionManager::get(queue) to get a single message from a queue. - Set FlowControl per-subscription.
r671824	gsim	2008-06-26	QPID-1147 : Avoid usage of 'source' builtin in pure sh scripts Patch from Manuel Teira that replaces "source" builtin with a dot inclusion on run_test script

r671825	arnaudsimon	2008-06-26	QPID-1112 : Changed addDeliveredMessage so to avoid division by 0 error when messages are not pre-fetched
r671845	aidan	2008-06-26	QPID-854 QPID-999 : Merge Changes to the client to make the dispatcher responsible for closing the queue browser when all the messages have been processed.
r671849	ritchiem	2008-06-26	QPID-909 : Commented out the TimerTask so that it can be wrapped with a ScaledTestDecorator. Minimal change to get our existing tests to run. If closer duration control is required then further time c ...
r671850	ritchiem	2008-06-26	Updated .gitignore with Intelij project files
r671877	gsim	2008-06-26	QPID-1137 : don't treat connection as opened if the open never succeeds
r671887	arnaudsimon	2008-06-26	QPID-1112 : Changed addDeliveredMessage so to avoid division by 0 error when max pre-fetch=1
r671902	aconway	2008-06-26	From Matt Farrellee - QPID-1151 "> https://issues.apache.org/jira/browse/QPID-1151 Remove un-necessary link dependencies from client and common libraries.
r671916	aconway	2008-06-26	Use run_test to run valgrind for start_broker consistently with other tests.
r671931	aidan	2008-06-26	QPID-1152 : Change visibility to public so that it isn't narrowed
r671949	ritchiem	2008-06-26	QPID-909 : Commented out the TimerTask so that it can be wrapped with a ScaledTestDecorator. Minimal change to get our existing tests to run. If closer duration control is required then further time c ...
r671969	aconway	2008-06-26	Consolidated cluster tests in cluster_test.cpp Improvements to BrokerFixture for testing.
r672032	aconway	2008-06-26	Plugin framework change: single PluginFactory creates per-target Plugin instances.

JIRAs

Key	Component(s)	Affects Version/s	Summary	Status
[]	https://issues.apache.org/jira/browse/		(QPID-1157) Add CruiseControl support	
[]	https://issues.apache.org/jira/browse/		(QPID-1156) Compile warning re casts in cpp/src/qpid/framing/Blob.h	
[]	https://issues.apache.org/jira/browse/		(QPID-1155) Compile warning re casts in cpp/src/tests/InlineVector.c	
[]	https://issues.apache.org/jira/browse/		(QPID-1154) Compile warning re casts in qpid/broker/SaslAuthentica	
[]	https://issues.apache.org/jira/browse/		(QPID-1153) managementgen/schema.py initializes unsigned value \	
[]	https://issues.apache.org/jira/browse/		(QPID-1152) JUnit toolkit does not compile	
[]	https://issues.apache.org/jira/browse/		(QPID-1151) All qpid C++ libraries link against everything from AC_C	
[]	https://issues.apache.org/jira/browse/		(QPID-1150) IoSession interface is not implemeted in any place	
[]	https://issues.apache.org/jira/browse/		(QPID-1149) Unused JMX instrumentation in Qpid broker	
[]	https://issues.apache.org/jira/browse/		(QPID-1148) c++ broker: need abstraction layer for flock and lockf po	

[https://issues.apache.org/jira/browse/]			(QPID-1147) Avoid usage of 'source' builtin in pure sh scripts
[https://issues.apache.org/jira/browse/]			(QPID-1146) Excel RTD Server
[https://issues.apache.org/jira/browse/]			(QPID-1145) Client has merge related imperfectitudes
[https://issues.apache.org/jira/browse/]			(QPID-1144) Test org.apache.qpid.test.testcases.ImmediateMessageTest.test_QPID_5 intermittently failing
[https://issues.apache.org/jira/browse/]			(QPID-1143) python client doesn't buffer
[https://issues.apache.org/jira/browse/]			(QPID-1142) session.sync() hangs if session.auto_sync is False
[https://issues.apache.org/jira/browse/]			(QPID-1141) Exceptions caught during connection created are not se
[https://issues.apache.org/jira/browse/]			(QPID-1140) connection id count should be static in the IO transport
[https://issues.apache.org/jira/browse/]			(QPID-1139) java client (0-10) will start rejecting messages after 2**3
[https://issues.apache.org/jira/browse/]			(QPID-1138) Ruby client doesn't support timestamp in codec
[https://issues.apache.org/jira/browse/]			(QPID-1137) (C++) It is required to call Connection::close even if it fa
[https://issues.apache.org/jira/browse/]			(QPID-1136) Broker does not correctly remove persistent message d
[https://issues.apache.org/jira/browse/]			(QPID-1135) Large messages are mangled
[https://issues.apache.org/jira/browse/]			(QPID-1134) Default username/password should be guest/guest
[https://issues.apache.org/jira/browse/]			(QPID-1133) Poller implementation based on the Solaris Event Comp
[https://issues.apache.org/jira/browse/]			(QPID-1132) Add support for Sun Studio compiler suite detection
[https://issues.apache.org/jira/browse/]			(QPID-1131) Refactor cpp/examples directory to be build under autot
[https://issues.apache.org/jira/browse/]			(QPID-1130) (0.10 code path) Tx sessions are not releasing unackno
[https://issues.apache.org/jira/browse/]			(QPID-1129) java client runs out of memory when the socket receive
[https://issues.apache.org/jira/browse/]			(QPID-1128) NPE displayed when adding queue to navigation

[https://issues.apache.org/jira/browse/]			(QPID-1127) (0.10 code path) The use of direct buffers causes memory corruption
[https://issues.apache.org/jira/browse/]			(QPID-1126) Client dies after 2^16 sessions are opened on a single connection
[https://issues.apache.org/jira/browse/]			(QPID-1125) MINA swallows exceptions in the I/O thread
[https://issues.apache.org/jira/browse/]			(QPID-1124) (Java Client) Use of thread-unsafe HashMap for destination mapping
[https://issues.apache.org/jira/browse/]			(QPID-1123) (0.10 code path) Connection close intermittently hangs
[https://issues.apache.org/jira/browse/]			(QPID-1122) Setting sync_persistence on the connection URL doesn't work
[https://issues.apache.org/jira/browse/]			(QPID-1121) Broker Federation - Link to unresolvable destination can cause broker to hang
[https://issues.apache.org/jira/browse/]			(QPID-1120) (01.0 code path) Large txs may exhaust credits
[https://issues.apache.org/jira/browse/]			(QPID-1119) (Java) Timeout on consumer creation with large queues
[https://issues.apache.org/jira/browse/]			(QPID-1118) (Java Client) JMS Destination Type no longer being set
[https://issues.apache.org/jira/browse/]			(QPID-1117) AbstractBytesMessage.getText corrupts the ByteBuffer
[https://issues.apache.org/jira/browse/]			(QPID-1116) (0.10 code path) Connection establishment process may hang
[https://issues.apache.org/jira/browse/]			(QPID-1115) Client ID is set even if it's disabled
[https://issues.apache.org/jira/browse/]			(QPID-1114) Improvements to daemon mode operations
[https://issues.apache.org/jira/browse/]			(QPID-1113) Management Cleanup
[https://issues.apache.org/jira/browse/]			(QPID-1112) (01.0 code path) only receives up to max prefetch messages
[https://issues.apache.org/jira/browse/]			(QPID-1111) (0.10 code path) Message.transfer sync flag should be ignored
[https://issues.apache.org/jira/browse/]			(QPID-1110) (0.10 code path) messages are not acquired when using transfer
[https://issues.apache.org/jira/browse/]			(QPID-1109) Decide of Protect io default settings
[https://issues.apache.org/jira/browse/]			(QPID-1108) QPID broker asserts in qpid::sys::RWlock::RWlock()

Qpid Java Meeting Minutes 2008-07-11

Agenda

Commits review
JIRA Review
AOCB

Outstanding actions

revision	committer	date	comment	review comments
r659163	arnaudsimon	2008-05-22	QPID-1079 : Updated ...test.client tests for using QpidTestCase + move QpidTestCase in main so it is visible from systests	Remove amqj.AutoCreateVMBroker Investigate if tests aren't working on trunk without this setting Remove VMTestCase? Remove all classes that were no longer referenced (VMBrokerSetup) RECEIVE_TIMEOUT : get rid of and use configurable timeout when available move kills to base case
r659262	arnaudsimon	2008-05-22	QPID-1079 : added junit dep to client as it's not included within all environments (for example on RHEL-4)	Should move QpidTestCase into common base package; not in client.
r659271	rhs	2008-05-22	Made Range, RangeSet, and Session all use proper RFC1982 comparisons per QPID-861 . Also switched command ids from long -> int, and added a mutex to channel to prevent multi-frame commands from interleave ...	Use .intValue() not (int) (long)
r661561	rajith	2008-05-30	This check in is for QPID-1102 . IoHandler and IoSender uses the java.io classes for IO operations and have shown very good improvement in latency and memory usage over MINA. For certain tests with pub ...	Should use configuration and be more configurable. Narrow catch Exception clause.
r669431	rgodfrey	2008-06-19	QPID-950 : Broker refactoring, copied / merged from branch	
r669480	rgodfrey	2008-06-19	QPID-950 : Fixed Derby Message Store	
r669841	rgodfrey	2008-06-20	QPID-1144 : Reference count drops to zero too early for immediate messages in a txn	
r661746	ritchier	2008-05-30	QPID-1101 : Update to DestNameExchange to perform deep copy.	Needs test
r662755	arnaudsimon	2008-06-03	QPID-1115 : Only generate client ID when necessary	RG to comment on Jira
r662770	ritchier	2008-06-03	QPID-1092 : Changed toString to be String.valueOf(getObject()) Added MessageToStringTest, tests performing toString on Message before calling getObject().	Weird catch in close()
r662827	arnaudsimon	2008-06-03	QPID-1112 : Update previous commit by re-using messageAcknowledge (added a flag specifying whether to send an messageAccept)	inRecover check in BMC_0_10.postDeliver might be a problem with async delivery
r662849	rhs	2008-06-03	QPID-1062 : modified generated code to keep packing flags in wire form and override commonly used size methods for improved performance	Remove commented out code
r663677	rhs	2008-06-05	QPID-1116 : fixed a race condition in connection/session close, session close now waits for the session to be detached before returning, this guarantees we won't have any active sessions when the conn ...	Client.java has random cruft added.
r665841	rhs	2008-06-09	QPID-901 : always reset the auto-sync mode even if the call fails	RHS: should raise Jira for autosync flag
r666259	rhs	2008-06-10	QPID-1129 : unless otherwise specified, limit the receive buffer size to 64K	Mina makes OOM'ing hard to figure out
r667561	ritchier	2008-06-13	QPID-1136 : Provided a fix for the leak in UnacknowledgedMessage when acking. Added a new InternalBrokerBaseCase for performing testing on the broker without using the client libraries. This allows fo ...	InternalMinaProtocolSession has a bug in awaitDelivery where it can hang because deliveryCount is already set to !0

r669841	rgodfrey	2008-06-20	QPID-1144 : Reference count drops to zero too early for immediate messages in a txn	Review LTC with Rob
r669885	rgodfrey	2008-06-20	QPID-1101 : Updated Direct Exchange so it does not modify lists of queues	No Test
r671845	aidan	2008-06-26	QPID-854 QPID-999 : Merge Changes to the client to make the dispatcher responsible for closing the queue browser when all the messages have been processed.	JIRA clean up of anon CloseMessage DeliveryBody class.
r671949	ritchier	2008-06-26	QPID-909 : Commented out the TimerTask so that it can be wrapped with a ScaledTestDecorator. Minimal change to get our existing tests to run. If closer duration control is required then further time c ...	Change Commit list

Qpid Java Meeting Minutes 2008-07-25

Agenda

Commits review
JIRA Review
AOCB Rich Text

Outstanding actions

revision	committer	date	comment	review comments
r659163	arnaudsimon	2008-05-22	QPID-1079 : Updated ...test.client tests for using QpidTestCase + move QpidTestCase in main so it is visible from systests	RECEIVE_TIMEOUT : get rid of and use configurable timeout when available
r669431	rgodfrey	2008-06-19	QPID-950 : Broker refactoring, copied / merged from branch	
r669480	rgodfrey	2008-06-19	QPID-950 : Fixed Derby Message Store	
r661746	ritchier	2008-05-30	QPID-1101 : Update to DestNameExchange to perform deep copy.	Needs test
r662755	arnaudsimon	2008-06-03	QPID-1115 : Only generate client ID when necessary	RG to comment on Jira
r662770	ritchier	2008-06-03	QPID-1092 : Changed toString to be String.valueOf(getObject()) Added MessageToStringTest, tests performing toString on Message before calling getObject().	Weird catch in close()
r662827	arnaudsimon	2008-06-03	QPID-1112 : Update previous commit by re-using messageAcknowledge (added a flag specifying whether to send an messageAccept)	inRecover check in BMC_0_10.postDeliver might be a problem with async delivery
r665841	rhs	2008-06-09	QPID-901 : always reset the auto-sync mode even if the call fails	RHS: make sure flag is used where appropriate
r667561	ritchier	2008-06-13	QPID-1136 : Provided a fix for the leak in UnacknowledgedMessage when acking. Added a new InternalBrokerBaseCase for performing testing on the broker without using the client libraries. This allows fo ...	InternalMinaProtocolSession has a bug in awaitDelivery where it can hang because deliveryCount is already set to !0
r669841	rgodfrey	2008-06-20	QPID-1144 : Reference count drops to zero too early for immediate messages in a txn	RG: document LocalTransactionalContext
r669885	rgodfrey	2008-06-20	QPID-1101 : Updated Direct Exchange so it does not modify lists of queues	No Test
r671845	aidan	2008-06-26	QPID-854 QPID-999 : Merge Changes to the client to make the dispatcher responsible for closing the queue browser when all the messages have been processed.	JIRA clean up of anon CloseMessage DeliveryBody class.
r671949	ritchier	2008-06-26	QPID-909 : Commented out the TimerTask so that it can be wrapped with a ScaledTestDecorator. Minimal change to get our existing tests to run. If closer duration control is required then further time c ...	Change Commit list
r672810	rajith	2008-06-30	This commit is related to QPID-1161 . Please refer to the JIRA for complete details. In Summary this contains a simple test kit comprising of perf and soak tests. The focus is on producing a packaged ...	

r674085	ritchiem	2008-07-04	QPID-871 - Added a ConnectionRegistry per Virtualhost to track the open connections. Altered the ApplicationRegistry so that when the shutdown hook is fired it: Unbinds from the listening sockets Then ...	
---------	----------	------------	--	--

Commits

revision	committer	date	comment	
r672032	aconway	2008-06-26	Plugin framework change: single PluginFactory creates per-target Plugin instances.	
r672269	aconway	2008-06-27	Fix exit status when VALIGRIND=	
r672280	arnaudsimon	2008-06-27	QPID-1157 : Added CC scripts and config files	
r672296	ritchiem	2008-06-27	Added qpid/java/release to ignore list	
r672300	ritchiem	2008-06-27	Updated the performance tests to ensure we use all the available test, added additional comments in pom.xml about each test section.	Why update maven?
r672303	ritchiem	2008-06-27	Added comment to gitignore to explain previous additions	
r672763	aidan	2008-06-30	QPID-1159 : remove @Override tags	
r672766	arnaudsimon	2008-06-30	QPID-1157 : Added cc example automation scripts	
r672810	rajith	2008-06-30	This commit is related to QPID-1161 . Please refer to the JIRA for complete details. In Summary this contains a simple test kit comprising of perf and soak tests. The focus is on producing a packaged ...	
r672854	tross	2008-06-30	Switch to async mode for management communication	
r672855	tross	2008-06-30	Cosmetic change: rename ID to be 'tag'	
r672864	tross	2008-06-30	QPID-1160 - Per-thread counters in management API to avoid locking	
r673031	gsim	2008-07-01	Added extra option (fixed time limit in rate mode) to latency test. Patch from acme@redhat.com.	
r673058	rgodfrey	2008-07-01	QPID-1084 : Applying patch previously applied to M2.x	
r673071	arnaudsimon	2008-07-01	QPID-1157 : Updated scripts	
r673074	arnaudsimon	2008-07-01	QPID-1163 : Moved message ack in pre-deliver method	Needs test. Commenting out of preDeliver Group review of BasicMessageConsumer* Check inRecovery change
r673082	aidan	2008-07-01	QPID-887 : name housekeeping thread properly. Apply patch from suran at wso2 dot com	
r673158	aconway	2008-07-01	Added timeout to SubscriptionManager::get(), LocalQueue::get() and BlockingQueue::get()	
r673343	aidan	2008-07-02	QPID-962 Exception handling was... unpleasing... Fix up of patch from rhs AMQConnection: Refactor listener and remove list, we're only interested in the most recent one anyway. Add get/set for lastEx ...	
r673347	aidan	2008-07-02	QPID-960 make protocol negotiation work from 0-10 down to 0-9 and then 8-0 still needs love to do with failover, see QPID-959 AMQConnection.java: use 8_0 delegate for in-vm tests AMQConnectionDelegat ...	
r673350	aidan	2008-07-02	QPID-960 copy delegate properly	
r673351	aidan	2008-07-02	QPID-960 remember to rename class	
r673359	gsim	2008-07-02	Improved text and raised severity of log entry when client sessions are deleted without first being closed.	
r673401	aidan	2008-07-02	Revert "QPID-962 Exception handling was... unpleasing... Fix up of patch from rhs" This reverts commit 673343.	

r673688	aidan	2008-07-03	QPID-962 Exception handling was... unpleasing... Fix up of patch from rhs AMQConnection.java: Refactor listener and stack exceptions in a list. Add get lastException, which can now be any Exception. ...	
r673718	tross	2008-07-03	QPID-1160 - Use array-style delete for allocated array	
r673725	aconway	2008-07-03	rubigen: Change default for client API accept-mode parameters to 1.	
r673947	ritchiem	2008-07-04	Removed SimpleACLTest from the build whilst we resolve the client exception handling problems causing the failure	
r674003	gsim	2008-07-04	Only override default value for accept-mode field in message.transfer (not message.subscribe)	
r674015	arnaudsimon	2008-07-04	QPID-1157 : Added perftests project	
r674040	gsim	2008-07-04	Allow default values for packed structs to be overridden (currently used for message.transfer.accept-mode)	
r674055	arnaudsimon	2008-07-04	QPID-1079 : Remove all classes that were no longer referenced + updated FlowControlTest for using QpidTestCase	
r674058	ritchiem	2008-07-04	Qpid-940 - ConnectionTest#testPasswordFailureConnection fails occasionally so while these race conditions are addressed I've converted the ConnectionTest to QpidTestCase and use it to skip the PasswordFailure test ...	
r674085	ritchiem	2008-07-04	QPID-871 - Added a ConnectionRegistry per Virtualhost to track the open connections. Altered the ApplicationRegistry so that when the shutdown hook is fired it: Unbinds from the listening sockets Then ...	
r674097	ritchiem	2008-07-04	QPID-940 : Forgot to exclude the test from the test run	
r674102	ritchiem	2008-07-04	Addition of tools directory for various Qpid Java tools The first tool JNDICheck allows the contents of a JNDI properties file to be parsed and presented as JNDI will process it. Handly for validating ...	
r674107	aconway	2008-07-04	Cluster prototype: handles client-initiated commands (not dequeues) Details - Cluster.cpp: serializes all frames thru cluster (see below) - broker/ConnectionManager: Added handler chain in front of ...	
r674113	aconway	2008-07-04	Remove debugging cout accidentally left in.	
r674124	aconway	2008-07-04	Disabled cluster_test temporarily, it leaks processes.	
r674389	rajith	2008-07-07	The last checkin for this class was using a Java 1.6 specific method called isEmpty in the String class. This fails the build in Java 1.5. I modified it to use str.length == 0 which has the same effect ...	
r674391	rajith	2008-07-07	This is related to QPID-1161 . Made minor modifications to the scripts and added a log4j file for the tests. The scripts are now modified to use the JAVA_HOME.	
r674392	rajith	2008-07-07	This is related to QPID-1161 . Added the absolute path to setenv.sh, so that the following scripts can be called from any location.	
r674482	gsim	2008-07-07	Temporarily reverting changes to signal handling; as checked in by r674107 it prevents the broker being shutdown.	
r674493	aconway	2008-07-07	configure.ac: check for cpg_local_get to exclude older CPG versions.	
r674504	aconway	2008-07-07	Restore use of SignalHandler in qpidd.cpp, fixed errors in previous commit.	
r674510	aidan	2008-07-07	QPID-474 Make sure that our SASL servers actually, y'know, validate the password AmqPlainSaslServer.java: Actually check password PlainSaslServer.java: Actually check password SaslServerTestCase.java ...	Need to define security notification mechanism
r674513	aidan	2008-07-07	QPID-474 forgot ASL header, oops	
r674541	aidan	2008-07-07	Disable certain ConnectionTest tests since the 010 broker doesn't currently implement that behaviour	
r674569	rajith	2008-07-07	This is related to QPID-1161. Modified the soak tests to print latency samples and throughput rates for every iteration. Added run_soak_client.sh soak_report.sh as an example of how to use soak test a ...	
r674587	aconway	2008-07-07	ForkedBroker: child process exits on completion.	

r674622	rajith	2008-07-07	This is related to QPID-1162 Added a README file to describe what the tests are and how they can be run. Modified to consumers to print the iteration number instead of the message id.	
r674747	aidan	2008-07-08	QPID-293 allow messages which have been received by the consumer before a message listener has been set to be delivered. BasicMessageConsumer.java: If there are messages on the synchronous queue when ...	
r674825	aconway	2008-07-08	Fix leak in XmlClientSessionTests - was leaking a Session.	
r674826	aconway	2008-07-08	svn:ignore properties.	
r674848	gsim	2008-07-08	<ul style="list-style-type: none"> • release message lock when notifying queue listeners * take copy of listeners * remove unused functionality 	
r674855	aconway	2008-07-08	Removed static Cpg::handlers, fixed ForkedBroker shutdown.	
r674865	aconway	2008-07-08	QPID-1148 - from Manuel Tiera Lock file abstraction in sys/ with implementation portable to Linux and Solaris. Changes by myself: - Makefile.am - must be updated for any new/renamed/removed source ...	
r674915	aconway	2008-07-08	Revert un-necessary Plugin complications. Better solution for plugin extension points coming up...	
r674939	rhs	2008-07-08	Branch at a stable point for 0-10 support (prior to M3). This includes the C++ broker, C++ client, Java client, and Python client all speaking the 0-10 protocol.	
r674945	aconway	2008-07-08	Fix packaging error.	
r674955	aconway	2008-07-08	Remove unused Serializer code.	
r674976	rajith	2008-07-08	This is related to QPID-1161 . Added the ability to pass in JVM ARGs.	
r674994	tross	2008-07-08	QPID-1170 - Remove boost dependency from management agent interface	
r675017	aconway	2008-07-08	HandlerChain: plug-in handler chain extension points. Replaces Handler<T>::Chain. Updated Sessoin & Connection handler chains and Cluster.	
r675144	aconway	2008-07-09	Fix for older boost versions	
r675146	aconway	2008-07-09	Fix signed/unsigned compare error	
r675155	aconway	2008-07-09	Removed dead code.	
r675165	rhs	2008-07-09	Primarily profiling driven changes: - added batched writes of commands/controls issued on a session - copy fragmented frames and segments rather than trying to decode them piecemeal, removed Fr ...	
r675252	gsim	2008-07-09	Allow for pluggable exchange types.	
r675338	astitcher	2008-07-09	Some small changes which clean up header file inclusions and generally start to tidy up the network layer so that it's a bit easier to implement new network transports	
r675397	rhs	2008-07-10	QPID-1062 : moved channel id into the ProtocolEvent interface and removed ConnectionEvent, this removes the overhead of creating ConnectionEvents	
r675433	rhs	2008-07-10	QPID-1171 : batch acks when prefetch is used	
r675477	gsim	2008-07-10	Honour timeout in BlockingQueue::pop(); added test for SubscriptionManager::get() where no message exists.	
r675486	gsim	2008-07-10	Assume accept-mode=1 (i.e. none required) where not explicitly specified on a message.transfer	
r675598	gsim	2008-07-10	Add a get() method to subscription manager that retrieves one message from the specified queue if available, returns false otherwise.	
r675674	tross	2008-07-10	Move shutdown of management broker to end of shutdown sequence	

r676067	tross	2008-07-11	QPID-1174 Remote Management Agent for management of external components	
r676581	gsim	2008-07-14	Allow for pluggable exchange types.	
r676613	rajith	2008-07-14	This is related to QPID-1163 . This is already in trunk and I am porting it to the qpid.0-10 branch.	
r676831	aidan	2008-07-15	Multi-version interop test script	
r676878	ritchiem	2008-07-15	QPID-909 Added missing license header and fixed execute bit on MessageSize.sh Added RunAll.sh for good measure	
r676879	ritchiem	2008-07-15	Updated gitignore with cpp example output and other generated files	
r676883	ritchiem	2008-07-15	QPID-1175 : VirtualHost now validates that name is non-null and non-empty. Full protocol validation of the virtualhost name has not been performed.	
r676884	ritchiem	2008-07-15	QPID-1176 : Updated Tasks and gentools build to use the java.source and java.target values. Added echo statements to show the targeted build Updated other info echo statements to be an info level so t ...	
r676885	ritchiem	2008-07-15	Updated log4j format as per discussion on mailing list.	
r676886	ritchiem	2008-07-15	Removed the non ASCII characters that are causing the build to minorly complain.	
r676887	ritchiem	2008-07-15	QPID-1172 : Moved unregistration out of the sendLock. Potential refactor possible between processQueue and flushSubscription	
r676932	aconway	2008-07-15	Switched from shared_ptr to intrusive_ptr and RefCounted for Broker.	
r676938	rajith	2008-07-15	This is related to QPID-1102 . I have fixed the error handling and revised the while loop in IoSender based on the comments received during the code review	
r676951	aidan	2008-07-15	fix cpp client, path changes	
r676963	aconway	2008-07-15	Fix "ignoring return value" warning from LockFile.h.	
r676969	ritchiem	2008-07-15	QPID-1079 : Based on Code Review : Remvoed AutoCreateVMBroker code from QpidTestCase. Removed VMTTestCase and all references to it, it was only used in JUnit4 testSuite wrappers. Rather than move QpidT ...	
r676971	ritchiem	2008-07-15	QPID-1079 : Based on Code Review : Remvoed AutoCreateVMBroker code from QpidTestCase. Removed VMTTestCase and all references to it, it was only used in JUnit4 testSuite wrappers. Rather than move QpidT ...	
r676972	ritchiem	2008-07-15	QPID-1176 : Update to gentools to remove commented out properties that I left in via git 😊	
r676973	ritchiem	2008-07-15	QPID-984 : Applied fix from M2.1.x that adds required synchronization around setup and tear down of Connections.	
r676978	ritchiem	2008-07-15	QPID-940 ,QPID-594 ,QPID-805 ,QPID-826 : Updated the client exception handling so that exceptions are not lost. In performing the changes I noted that the AMQStateManager is only used for connection crea ...	
r676982	ritchiem	2008-07-15	QPID-1177 : Added Protocol Level Debug logging. Uses a final static so should JIT out if disabled. To enable set -Damq.protocol.logging.level=info	
r677256	ritchiem	2008-07-16	QPID-1178 : Prevent Rejecting messages destined for known QueueBrowsers	
r677257	ritchiem	2008-07-16	Changed erroneous error level logging to info level	
r677258	ritchiem	2008-07-16	Added a warning log statement if the TransportConnection autocreates an InVM Broker	
r677259	ritchiem	2008-07-16	Converted client.failover.FailoverTest so it can utilise the standard mechanism for failover testing, as the local CruiseControl had testP2PFailoveWithMessagesLeft fail with extra messages being left ...	

r677260	ritchiem	2008-07-16	QPID-1179 : Adjusted the test size from 100 to 10, this should reduce the likely hood of a slow machine failing the test.	
r677262	ritchiem	2008-07-16	Renamed shutdownServer to restartBroker as that is what is does	
r677263	ritchiem	2008-07-16	QPID-1181 : Added additional logging to help diagnose a NullPointerException	
r677319	ritchiem	2008-07-16	Update to the logging to ensure QpidTestCase is always logged and standardized the protocol output format between 0-8/0-9 and 0-10	
r677327	ritchiem	2008-07-16	QPID-871 : The shutdown change had a spurious getInstance() call which would case a new instance of ID 1 to be created if there wasn't one, it would then proceed to shutdown that MBeanServer not the M ...	
r677408	tross	2008-07-16	QPID-1170 - Provide a better factory for creation and deletion of the management agent	
r677412	tross	2008-07-16	QPID-1170 - Removed spurious include from example	
r677471	aconway	2008-07-17	Cluster: shadow connections, fix lifecycle & valgrind issues. - tests/ForkedBroker: improved broker forking, exec full qpidd. - Plugin::addFinalizer - more flexible way to shutdown plugins. - Rewo ...	
r677486	aconway	2008-07-17	Enable dequeue for prototype cluster - qpid/broker/SemanticState.cpp: moved doOutput into write idle callback. - qpid/broker/Connection.cpp: make doOutput an intercept point. - qpid/cluster/*: inte ...	
r677525	ritchiem	2008-07-17	QPID-1177 : Fixed the format of the messages, realised that the transport.Connection uses a logging wrapper but in my haste to make the format the same in AMQProtocolHandler hadn't checked the output ...	
r677629	ritchiem	2008-07-17	Moved the Reflection Wrapping code used by the system tests to the system test. If they are left in common then we must include the common directory when using the systest-testing frameworks no matter ...	
r677633	ritchiem	2008-07-17	QPID-1182 : Added additional logging to identify the exception that caused Authentication to fail.	
r678211	rhs	2008-07-19	QPID-1184 : redirect stdout and stderr from QpidTestCase	
r678260	rhs	2008-07-20	QPID-1185 : replaced occurrences of with , also made default.testprofile always load so that all other testprofiles only need to override values that are different	
r678759	gsim	2008-07-22	Fix to transaction batching. (Backport of r669215).	
r678848	rhs	2008-07-22	Updated the io transport to use a separate write thread with a circular buffer that does opportunistic write batching. Fixed error handling and shutdown for the io transport. Switched default from min ...	
r679038	gsim	2008-07-23	Further fixes to transactional perftest: * correction to transaction boundaries * ensure any outstanding acks are sent on completion of subscriber	
r679045	gsim	2008-07-23	Fixes for transactional perftest (merge of r679038 from qpid.0-10)	
r679048	gsim	2008-07-23	QPID-1183 : Use the right sizes to insert data inside the message payload where sizeof(size_t) != sizeof(uint32_t). Patch from Manuel Teira.	
r679059	ritchiem	2008-07-23	QPID-1187 : The broker did not correctly handle subscriptions that would suspend due to exhaustion of bytes credit. The processQueue loop would spin, this fix marks the subscription inactive for that ...	
r679105	arnaudsimon	2008-07-23	qpid-1157: added jms tck scripts + README file + config file for setting email related properties	
r679232	rhs	2008-07-23	excluded a known-failing test for durable subscriptions, the fix is on trunk and doesn't as yet need to be backported	
r679268	astitcher	2008-07-24	Refactor to change client connector state machine to be held in ConnectionHandler	
r679276	astitcher	2008-07-24	Refactored so that Dispatcher is now independent from DispatchHandle	

r679462	gsim	2008-07-24	Set a configurable default size limit on queues	
r679469	gsim	2008-07-24	Allow configurable default size limit to be set for queues (merged from r679462).	
r679481	arnaudsimon	2008-07-24	qpid-1157: updated java trunk so a report is generated when there is a fault	
r679689	gsim	2008-07-25	QPID-447 : Optional mechanism to avoid race when automating topic tests. Patch from David Sommerseth.	
r679699	gsim	2008-07-25	QPID-447 : Patch from David Sommerseth merged from r679689.	
r679717	gsim	2008-07-25	QPID-1154 , QPID-1155 & QPID-1156 : Patches from Steve Huston to fix various minor compiler errors.	
r679739	gsim	2008-07-25	Fixed bug in SubscriptionManager::get() where flush was issued before waiting and if message showed up after flush completed but before wait was finished there was no credit (due to flush) to deliver ...	
r679748	gsim	2008-07-25	Merged fix to SubscriptionManager (was r679739)	
r679756	gsim	2008-07-25	Exclude core verify script from verifications run when python examples cannot be found.	
r679762	aidan	2008-07-25	Add xslt magic for creating code review agenda, and add wrapper script	

Jiras

Key	Component(s)	Affects Version/s	Summary	Status	Assignee	Reporter	Review Comments
QPID-1182	Java Broker		(QPID-1182) SimpleACLTest authentication failures	Open	Martin Ritchie	Martin Ritchie	
QPID-1183	C++ Broker	M3	(QPID-1183) perftest doesn't work correctly when sizeof(size_t) != sizeof(uint32_t)	Resolved	Unassigned	Manuel Teira	
QPID-1184	Java Tests	M3	(QPID-1184) all output during tests is buffered until the test case finishes	Open	Rafael H. Schloeming	Rafael H. Schloeming	
QPID-1185	Ant Build System	M3	(QPID-1185) -Dlog=foo is ignored	Open	Rafael H. Schloeming	Rafael H. Schloeming	
QPID-1186	Java Client	M2.1	(QPID-1186) (Client Race Condition) After Failover client can ack last message from previous session.	Open	Martin Ritchie	Martin Ritchie	
QPID-1187	Java Broker	M3	(QPID-1187) Java Broker appears to be stuck in a loop	In Progress	Martin Ritchie	Martin Ritchie	
QPID-1188	Java Client	M3	(QPID-1188) java 0-10 client deadlocks when running with -Dprotectio=true	Open	Rafael H. Schloeming	Rafael H. Schloeming	
QPID-1189	Java Broker	M3	(QPID-1189) Ant target need to be execute twice to build Qpid	Open	Unassigned	Asanka Abeysinghe	
QPID-1190	Java Broker	M3	(QPID-1190) Broker logs 0-10 negotiation failure	Open	Marnie McCormack	Aidan Skinner	
QPID-1191	Java Broker, JMS Compliance		(QPID-1191) Enable Exchange level filters.	Open	Martin Ritchie	Martin Ritchie	
QPID-1192	Java Client, JMS Compliance		(QPID-1192) Client needs to send selector string as part of Binding request when using topics	Open	Martin Ritchie	Martin Ritchie	
QPID-1193	Java Broker MessageStore - DerbyStore		(QPID-1193) Bind arguments must be stored with binding in DerbyStore	Open	Martin Ritchie	Martin Ritchie	
QPID-1194	Java Broker, Java Client		(QPID-1194) Enable Selector use on JMS Topics in the Java Broker	Open	Martin Ritchie	Martin Ritchie	

QPID-1195	Java Broker		(QPID-1195) Recovery with Argument Maps	Open	Martin Ritchie	Martin Ritchie	
QPID-1196	Java Broker, Java Broker MessageStore - DerbyStore		(QPID-1196) Queue Entries should be in terms of id's not queue -names	Open	Martin Ritchie	Martin Ritchie	
QPID-1197	Java Broker		(QPID-1197) Improve persistent recovery	Open	Unassigned	Martin Ritchie	
QPID-1198	C++ Broker	M3	(QPID-1198) Changes for the solaris port	Open	Andrew Stitzer	Manuel Teira	

Qpid Java Meeting Minutes 2008-08-01

Agenda

Commits review
JIRA Review
AOCB

Outstanding actions

revision	committer	date	comment	review comments
r659163	arnaudsimon	2008-05-22	QPID-1079 : Updated ...test.client tests for using QpidTestCase + move QpidTestCase in main so it is visible from sys-tests	RECEIVE_TIMEOUT : get rid of and use configurable timeout when available
r669431	rgodfrey	2008-06-19	QPID-950 : Broker refactoring, copied / merged from branch	
r669480	rgodfrey	2008-06-19	QPID-950 : Fixed Derby Message Store	
r661746	ritchiem	2008-05-30	QPID-1101 : Update to DestNameExchange to perform deep copy.	Needs test
r662755	arnaudsimon	2008-06-03	QPID-1115 : Only generate client ID when necessary	RG to comment on Jira
r662770	ritchiem	2008-06-03	QPID-1092 : Changed toString to be String.valueOf(getObject()) Added MessageToStringTest, tests performing toString on Message before calling getObject().	Weird catch in close()
r662827	arnaudsimon	2008-06-03	QPID-1112 : Update previous commit by re-using messageAcknowledge (added a flag specifying whether to send an messageAccept)	inRecover check in BMC_0_10.postDeliver might be a problem with async delivery
r665841	rhs	2008-06-09	QPID-901 : always reset the auto-sync mode even if the call fails	RHS: make sure flag is used where appropriate
r667561	ritchiem	2008-06-13	QPID-1136 : Provided a fix for the leak in UnacknowledgedMessage when acking. Added a new InternalBrokerBaseCase for performing testing on the broker without using the client libraries. This allows for ...	InternalMinaProtocolSession has a bug in awaitDelivery where it can hang because deliveryCount is already set to !0
r669841	rgodfrey	2008-06-20	QPID-1144 : Reference count drops to zero too early for immediate messages in a txn	RG: document LocalTransactionalContext
r669885	rgodfrey	2008-06-20	QPID-1101 : Updated Direct Exchange so it does not modify lists of queues	No Test
r671845	aidan	2008-06-26	QPID-854 QPID-999 : Merge Changes to the client to make the dispatcher responsible for closing the queue browser when all the messages have been processed.	JIRA clean up of anon CloseMessage DeliveryBody class.
r671949	ritchiem	2008-06-26	QPID-909 : Commented out the TimerTask so that it can be wrapped with a ScaledTestDecorator. Minimal change to get our existing tests to run. If closer duration control is required then further time c ...	Change Commit list
r672810	rajith	2008-06-30	This commit is related to QPID-1161 . Please refer to the JIRA for complete details. In Summary this contains a simple test kit comprising of perf and soak tests. The focus is on producing a packaged ...	

r674085	ritchiem	2008-07-04	QPID-871 - Added a ConnectionRegistry per Virtualhost to track the open connections. Altered the ApplicationRegistry so that when the shutdown hook is fired it: Unbinds from the listening sockets Then ...	
---------	----------	------------	--	--

Commits

revision	committer	date	comment	
r674976	rajith	2008-07-08	This is related to QPID-1161 . Added the ability to pass in JVM ARGs.	
r674994	tross	2008-07-08	QPID-1170 - Remove boost dependency from management agent interface	
r675017	aconway	2008-07-08	HandlerChain: plug-in handler chain extension points. Replaces Handler<T>::Chain. Updated Sessoin & Connection handler chains and Cluster.	
r675144	aconway	2008-07-09	Fix for older boost versions	
r675146	aconway	2008-07-09	Fix signed/unsigned compare error	
r675155	aconway	2008-07-09	Removed dead code.	
r675165	rhs	2008-07-09	Primarily profiling driven changes: - added batched writes of commands/controls issued on a session - copy fragmented frames and segments rather than trying to decode them piecemeal, removed Fr ...	
r675252	gsim	2008-07-09	Allow for pluggable exchange types.	
r675338	astitcher	2008-07-09	Some small changes which clean up header file inclusions and generally start to tidy up the network layer so that it's a bit easier to implement new network transports	
r675397	rhs	2008-07-10	QPID-1062 : moved channel id into the ProtocolEvent interface and removed ConnectionEvent, this removes the overhead of creating ConnectionEvents	
r675433	rhs	2008-07-10	QPID-1171 : batch acks when prefetch is used	
r675477	gsim	2008-07-10	Honour timeout in BlockingQueue::pop(); added test for SubscriptionManager::get() where no message exists.	
r675486	gsim	2008-07-10	Assume accept-mode=1 (i.e. none required) where not explicitly specified on a message.transfer	
r675598	gsim	2008-07-10	Add a get() method to subscription manager that retrieves one message from the specified queue if available, returns false otherwise.	
r675674	tross	2008-07-10	Move shutdown of management broker to end of shutdown sequence	
r676067	tross	2008-07-11	QPID-1174 Remote Management Agent for management of external components	
r676581	gsim	2008-07-14	Allow for pluggable exchange types.	
r676613	rajith	2008-07-14	This is related to QPID-1163 . This is already in trunk and I am porting it to the qpid.0-10 branch.	
r676831	aidan	2008-07-15	Multi-version interop test script	
r676878	ritchiem	2008-07-15	QPID-909 Added missing license header and fixed execute bit on MessageSize.sh Added RunAll.sh for good measure	
r676879	ritchiem	2008-07-15	Updated gitignore with cpp example output and other generated files	
r676883	ritchiem	2008-07-15	QPID-1175 : VirtualHost now validates that name is non-null and non-empty. Full protocol validation of the virtualhost name has not been performed.	
r676884	ritchiem	2008-07-15	QPID-1176 : Updated Tasks and gentools build to use the java.source and java.target values. Added echo statements to show the targeted build Updated other info echo statements to be an info level so t ...	
r676885	ritchiem	2008-07-15	Updated log4j format as per discussion on mailing list.	
r676886	ritchiem	2008-07-15	Removed the non ASCII characters that are causing the build to minorly complain.	
r676887	ritchiem	2008-07-15	QPID-1172 : Moved unregistration out of the sendLock. Potential refactor possible between processQueue and flushSubscription	

r676932	aconway	2008-07-15	Switched from shared_ptr to intrusive_ptr and RefCounted for Broker.	
r676938	rajith	2008-07-15	This is related to QPID-1102 . I have fixed the error handling and revised the while loop in IoSender based on the comments received during the code review	
r676951	aidan	2008-07-15	fix cpp client, path changes	
r676963	aconway	2008-07-15	Fix "ignoring return value" warning from LockFile.h.	
r676969	ritchiem	2008-07-15	QPID-1079 : Based on Code Review : Remvoed AutoCreateVMBroker code from QpidTestCase. Removed VMTestCase and all references to it, it was only used in JUnit4 testSuite wrappers. Rather than move QpidT ...	
r676971	ritchiem	2008-07-15	QPID-1079 : Based on Code Review : Remvoed AutoCreateVMBroker code from QpidTestCase. Removed VMTestCase and all references to it, it was only used in JUnit4 testSuite wrappers. Rather than move QpidT ...	
r676972	ritchiem	2008-07-15	QPID-1176 : Update to gentools to remove commented out properties that I left in via git 😞	
r676973	ritchiem	2008-07-15	QPID-984 : Applied fix from M2.1.x that adds required synchronization around setup and tear down of Connections.	
r676978	ritchiem	2008-07-15	QPID-940 , QPID-594 , QPID-805 , QPID-826 : Updated the client exception handling so that exceptions are not lost. In performing the changes I noted that the AMQStateManager is only used for connection crea ...	
r676982	ritchiem	2008-07-15	QPID-1177 : Added Protocol Level Debug logging. Uses a final static so should JIT out if disabled. To enable set -Damqj.protocol.logging.level=info	
r677256	ritchiem	2008-07-16	QPID-1178 : Prevent Rejecting messages destined for known QueueBrowsers	
r677257	ritchiem	2008-07-16	Changed erroneous error level logging to info level	
r677258	ritchiem	2008-07-16	Added a warning log statement if the TransportConnection autocreates an InVM Broker	
r677259	ritchiem	2008-07-16	Converted client.failover.FailoverTest so it can utilise the standard mechanism for failover testing, as the local CruiseControl had testP2PFailoveWithMessagesLeft fail with extra messages being left ...	
r677260	ritchiem	2008-07-16	QPID-1179 : Adjusted the test size from 100 to 10, this should reduce the likely hood of a slow machine failing the test.	
r677262	ritchiem	2008-07-16	Renamed shutdownServer to restartBroker as that is what is does	
r677263	ritchiem	2008-07-16	QPID-1181 : Added additional logging to help diagnose a NullPointerException	
r677319	ritchiem	2008-07-16	Update to the logging to ensure QpidTestCase is always logged and standardized the protocol output format between 0-8/0-9 and 0-10	
r677327	ritchiem	2008-07-16	QPID-871 : The shutdown change had a spurious getInstance() call which would case a new instance of ID 1 to be created if there wasn't one, it would then procede to shutdown that MBeanServer not the M ...	
r677408	tross	2008-07-16	QPID-1170 - Provide a better factory for creation and deletion of the management agent	
r677412	tross	2008-07-16	QPID-1170 - Removed spurious include from example	
r677471	aconway	2008-07-17	Cluster: shadow connections, fix lifecycle & valgrind issues. - tests/ForkedBroker: improved broker forking, exec full qpid. - Plugin::addFinalizer - more flexible way to shutdown plugins. - Rewo ...	
r677486	aconway	2008-07-17	Enable dequeue for prototype cluster - qpid/broker/SemanticState.cpp: moved doOutput into write idle callback. - qpid/broker/Connection.cpp: make doOutput an intercept point. - qpid/cluster/*: inte ...	
r677525	ritchiem	2008-07-17	QPID-1177 : Fixed the format of the messages, realised that the transport.Connection uses a logging wrapper but in my haste to make the format the same in AMQProtocolHandler hadn't checked the output ...	
r677629	ritchiem	2008-07-17	Moved the Reflection Wrapping code used by the system tests to the system test. If they are left in common then we must include the common directory when using the systest-testing frameworks no matter ...	
r677633	ritchiem	2008-07-17	QPID-1182 : Added additional logging to identify the exception that caused Authentication to fail.	

r678211	rhs	2008-07-19	QPID-1184 : redirect stdout and stderr from QpidTestCase	
r678260	rhs	2008-07-20	QPID-1185 : replaced occurrences of with , also made default.testprofile always load so that all other testprofiles only need to override values that are different	
r678759	gsim	2008-07-22	Fix to transaction batching. (Backport of r669215).	
r678848	rhs	2008-07-22	Updated the io transport to use a separate write thread with a circular buffer that does opportunistic write batching. Fixed error handling and shutdown for the io transport. Switched default from min ...	
r679038	gsim	2008-07-23	Further fixes to transactional perftest: * correction to transaction boundaries * ensure any outstanding acks are sent on completion of subscriber	
r679045	gsim	2008-07-23	Fixes for transactional perftest (merge of r679038 from qpid.0-10)	
r679048	gsim	2008-07-23	QPID-1183 : Use the right sizes to insert data inside the message payload where sizeof(size_t) != sizeof(uint32_t). Patch from Manuel Teira.	
r679059	ritchiem	2008-07-23	QPID-1187 : The broker did not correctly handle subscriptions that would suspend due to exhaustion of bytes credit. The processQueue loop would spin, this fix marks the subscription inactive for that ...	
r679105	arnaudsimon	2008-07-23	qpid-1157: added jms tck scripts + README file + config file for setting email related properties	
r679232	rhs	2008-07-23	excluded a known-failing test for durable subscriptions, the fix is on trunk and doesn't as yet need to be backported	
r679268	astitcher	2008-07-24	Refactor to change client connector state machine to be held in ConnectionHandler	
r679276	astitcher	2008-07-24	Refactored so that Dispatcher is now independent from DispatchHandle	
r679462	gsim	2008-07-24	Set a configurable default size limit on queues	
r679469	gsim	2008-07-24	Allow configurable default size limit to be set for queues (merged from r679462).	
r679481	arnaudsimon	2008-07-24	qpid-1157: updated java trunk so a report is generated when there is a fault	
r679689	gsim	2008-07-25	QPID-447 : Optional mechanism to avoid race when automating topic tests. Patch from David Sommerseth.	
r679699	gsim	2008-07-25	QPID-447 : Patch from David Sommerseth merged from r679689.	
r679717	gsim	2008-07-25	QPID-1154 , QPID-1155 & QPID-1156 : Patches from Steve Huston to fix various minor compiler errors.	
r679739	gsim	2008-07-25	Fixed bug in SubscriptionManager::get() where flush was issued before waiting and if message showed up after flush completed but before wait was finished there was no credit (due to flush) to deliver ...	
r679748	gsim	2008-07-25	Merged fix to SubscriptionManager (was r679739)	
r679756	gsim	2008-07-25	Exclude core verify script from verifications run when python examples cannot be found.	
r679762	aidan	2008-07-25	Add xslt magic for creating code review agenda, and add wrapper script	
r679801	gsim	2008-07-25	Only reduce count and size maintained for queue plicy when messages are actually dequeued (i.e. acked).	
r679805	gsim	2008-07-25	Only reduce count and size maintained for queue plicy when messages are actually dequeued (i.e. acked).	
r679822	gsim	2008-07-25	Reduce the size of messages in fanout perftest to keep the queues from getting too large.	
r679827	gsim	2008-07-25	Reduce the size of messages in fanout perftest to keep the queues from getting too large.	
r680309	aidan	2008-07-28	QPID-1200 : Only set out and err if we're actually redirecting them.	
r680313	aidan	2008-07-28	Add some escaping action to the sed in svncmd so that it works right	
r680318	gsim	2008-07-28	Remove unused Module.h header file.	
r680349	aidan	2008-07-28	Add java test profile	
r680362	gsim	2008-07-28	Ensure that the management thread is stopped before shutdown() returns (to allow sensible behaviour for deletion of statics).	

r680395	astitcher	2008-07-28	Refactor of EpollPoller to make PollerHandler lifecycle easier	
r680601	rhs	2008-07-29	removed defaulted entries from the java testprofile	
r680602	rhs	2008-07-29	QPID-1201 : fixed up version of aidan's patch, there are still failures when running against an external java broker, however we seem to get past basic connection negotiation now	Need to check that this works with 0-8 only broker
r680673	rhs	2008-07-29	QPID-1201 : fixed some brainos in IoSender	
r680691	astitcher	2008-07-29	Fix for client busy looping whilst waiting for a message	
r680695	gsim	2008-07-29	Merged r680691	
r680750	aidan	2008-07-29	QPID-1203 Don't treat protocol negotiation failure as failover reducing error.	
r680751	aidan	2008-07-29	QPID-1203 : use slf4j instead of log4j directly	
r680752	aidan	2008-07-29	QPID-1203 : Add 08ExcludeList for external Java broker and make the profile use that. Make AMQConnectionFactory take an optional clientid and use that if specified.	
r680798	tross	2008-07-29	QPID-1153 - Patch from Steve Huston	
r680803	rhs	2008-07-29	QPID-1072 : renamed org.apache.qpidity -> org.apache.qpid	
r680826	astitcher	2008-07-29	QPID-1198 : (Partial) Fix test shell scripts to work with /bin/sh Patches from Manuel Teira. These scripts have #!/bin/sh but they were previously really dependent on bash.	
r680827	astitcher	2008-07-29	QPID-1198 (Partial): Added explicit namespaces that the Sun C++ requires (that gcc doesn't) Patches from Manuel Teira. It's not clear at this point whether there is a compiler problem with gcc that it ...	
r680828	astitcher	2008-07-29	QPID-1198 (Partial): replace all uses of u_intX_t with uintX_t Patches from Manuel Teira. The u_int* versions are not available in the Sun header files. In any case using only uint* is more consistent ...	
r680829	astitcher	2008-07-29	QPID-1198 (Partial): Add #include <alloca.h> for all uses of ::alloca() Patches from Manuel Teira.	
r680830	astitcher	2008-07-29	QPID-1198 (Partial): Missing header files that are really needed Patches from Manuel Teira. Compilation works on Linux due to implicit header inclusions but fails on Solaris Some tightening up of std ...	
r680831	astitcher	2008-07-29	Small comment tidy	
r680833	astitcher	2008-07-29	Removed unused functions Removed unused Thread and Socket functions - These functions also cause problems with Solaris compilations Remove unused client connector functionality	
r680918	astitcher	2008-07-30	Removed errno from a default parameter as I'm not convinced that this is always portable as errno could be a macro	
r680919	astitcher	2008-07-30	QPID-1198 (adapted): Change use of uuid lib not to assume const parameters The Solaris version of uuid.h takes uint8_t* not const uint8_t*	
r680920	astitcher	2008-07-30	Related to QPID-1198 : Moved posix platform specific "strerror" code to platform specific directory	
r680921	astitcher	2008-07-30	QPID-1198 : Solaris ECF (port) based Poller Patch from Manuel Teira	
r680937	gsim	2008-07-30	Added error handling for case where socket cannot be accepted e.g. due to constraints on file handles.	
r680939	gsim	2008-07-30	Merged r680937. Added error handling for case where socket cannot be accepted e.g. due to constraints on file handles.	
r680941	ritchiem	2008-07-30	QPID-1000 : Made both changes as per JIRA notes	
r680942	ritchiem	2008-07-30	Update QpidTestCase to add /bin to the path for the external broker	
r680987	rhs	2008-07-30	added defaulting of QPID_HOME	
r681117	aidan	2008-07-30	QPID-1192 : Make consumer send Selector as part of binding. QPID-1191 : Add test to exhibit leak Change DurableSubscriptionTest to validate exception type received Make BasicMessageConsumer validate th ...	
r681164	gsim	2008-07-30	QPID-1162 : added patches and additions required to build against boost 1.32. These are not deemed desirable for direct application to the trunk, but can be used to simply update an svn checkout for co ...	

r681193	astitcher	2008-07-30	The previous attempt to only get an xpg strerror_r with the GNU failed instead use the definition of _GNU_SOURCE as a proxy for the gnu version	
r681318	aidan	2008-07-31	Turn off TopicSessionTest#testNonMatchingMessagesDoNotFillQueue since c++ broker doesn't do server side selectors	
r681320	aidan	2008-07-31	Fix line break	
r681333	arnaudsimon	2008-07-31	qpid-1205: deleted cpp.sync profile, added cpp.noprefetch profile	
r681336	arnaudsimon	2008-07-31	qpid-1205: deleted exclude list from cc	
r681362	tross	2008-07-31	QPID-1174 - Management updates for remote agents	
r681367	arnaudsimon	2008-07-31	qpid-1163: Added test for qpid-1163 (Note: I have checked that this test did not pass before r673074)	
r681407	gsim	2008-07-31	Fixed for 64bit systems	
r681408	rajith	2008-07-31	This is related to QPID-1208 . I have enabled the code which will print xxxx when the log level != debug.	Should not ever print the password.
r681411	rajith	2008-07-31	This is related to QPID-1208 I have enabled the code which will print xxxx when the log level != debug.	
r681474	rhs	2008-07-31	QPID-1207 : fixed io transport close to ensure threads shutdown properly	
r681476	rhs	2008-07-31	QPID-1210 : made qpid-run output level configurable	Fix setting of level in qpid-server to include \ or indent properly
r681477	rhs	2008-07-31	added tools module to the main build	
r681479	cctrielloff	2008-07-31	<ul style="list-style-type: none"> Implementation of ACL plugin - Apply ACL to Exchange, Queue, Binding, Subscribe - Follow Java ACL types, few added To complete the implementation of ACL the following items are remaining. - ACL on ... 	
r681483	cctrielloff	2008-07-31	missing file	
r681491	cctrielloff	2008-07-31	small cleanup	
r681494	cctrielloff	2008-07-31	another missing file	
r681505	cctrielloff	2008-07-31	header file fix fr build	
r681509	cctrielloff	2008-07-31	attempt to fix spec file	
r681512	tross	2008-07-31	Added signed integer datatypes for use in management schemas	

Qpid Java Meeting Minutes 2008-08-08

Agenda

Commits review
JIRA Review
AOCB

Outstanding actions

revision	committer	date	comment	review comments
r659163	arnaudsimon	2008-05-22	QPID-1079 : Updated ...test.client tests for using QpidTestCase + move QpidTestCase in main so it is visible form systests	RECEIVE_TIMEOUT : get rid of and use configurable timeout when available
r669431	rgodfrey	2008-06-19	QPID-950 : Broker refactoring, copied / merged from branch	
r669480	rgodfrey	2008-06-19	QPID-950 : Fixed Derby Message Store	
r661746	ritchiem	2008-05-30	QPID-1101 : Update to DestNameExchange to perform deep copy.	Needs test
r662755	arnaudsimon	2008-06-03	QPID-1115 : Only generate client ID when necessary	RG to comment on Jira

r662770	ritchiem	2008-06-03	QPID-1092 : Changed toString to be String.valueOf(getObject()) Added MessageToStringTest, tests performing toString on Message before calling getObject().	Weird catch in close()
r662827	arnaudsimon	2008-06-03	QPID-1112 : Update previous commit by re-using messageAcknowledge (added a flag specifying whether to send an messageAccept)	inRecover check in BMC_0_10.postDeliver might be a problem with async delivery
r665841	rhs	2008-06-09	QPID-901 : always reset the auto-sync mode even if the call fails	RHS: make sure flag is used where appropriate
r667561	ritchiem	2008-06-13	QPID-1136 : Provided a fix for the leak in UnacknowledgedMessage when acking. Added a new InternalBrokerBaseCase for performing testing on the broker without using the client libraries. This allows fo ...	InternalMinaProtocolSession has a bug in awaitDelivery where it can hang because deliveryCount is already set to !0
r669841	rgodfrey	2008-06-20	QPID-1144 : Reference count drops to zero too early for immediate messages in a txn	RG: document LocalTransactionalContext
r669885	rgodfrey	2008-06-20	QPID-1101 : Updated Direct Exchange so it does not modify lists of queues	No Test
r671845	aidan	2008-06-26	QPID-854 QPID-999 : Merge Changes to the client to make the dispatcher responsible for closing the queue browser when all the messages have been processed.	JIRA clean up of anon CloseMessage DeliveryBody class.
r671949	ritchiem	2008-06-26	QPID-909 : Commented out the TimerTask so that it can be wrapped with a ScaledTestDecorator. Minimal change to get our existing tests to run. If closer duration control is required then further time c ...	Change Commit list
r672810	rajith	2008-06-30	This commit is related to QPID-1161 . Please refer to the JIRA for complete details. In Summary this contains a simple test kit comprising of perf and soak tests. The focus is on producing a packaged ...	
r674085	ritchiem	2008-07-04	QPID-871 - Added a ConnectionRegistry per Virtualhost to track the open connections. Altered the ApplicationRegistry so that when the shutdown hook is fired it: Unbinds from the listening sockets Then ...	
r680602	rhs	2008-07-29	QPID-1201 : fixed up version of aidan's patch, there are still failures when running against an external java broker, however we seem to get past basic connection negotiation now	Need to check that this works with 0-8 only broker
r681408	rajith	2008-07-31	This is related to QPID-1208 . I have enabled the code which will print xxxx when the log level != debug.	Should not ever print the password.
r681476	rhs	2008-07-31	QPID-1210 : made qpid-run output level configurable	Fix setting of level in qpid-server to include \ or indent properly

Commits

revision	committer	date	comment	Review Notes
r681666	rhs	2008-08-01	added benchmark tool for java native + jms APIs	
r681674	rhs	2008-08-01	improved usage	
r681690	cctrielloff	2008-08-01	<ul style="list-style-type: none"> • Add support for ACL on message transfer - Performance optimizations for ACL on message transfer 	
r681709	tross	2008-08-01	Make md5-hash of table recursively include data from referenced groups	
r681727	kpvdr	2008-08-01	Removed typedefs which were generating ignored warnings on gcc 4.3 compiler.	
r681773	tross	2008-08-01	QPID-1174 - Clean up agent objects when the remote agent disconnects	
r681821	tross	2008-08-01	Don't pad out the last column of a table	
r681824	kpvdr	2008-08-01	Initial framework for ACL reader	
r682309	ritchiem	2008-08-04	QPID-1215 : Replaced use of FileReader with FileInputStream	
r682410	cctrielloff	2008-08-04	updated ais instructions	
r682418	cctrielloff	2008-08-04	correc version number	

r682644	aidan	2008-08-05	Add slf4j deps to perftests, we should move all this to ant	
r682672	aidan	2008-08-05	QPID-1206 : Fix failover and failover tests AMQConnection: remove dead and confusingly misnamed method AMQSession: rename failedOver to failedOverDirty to convey actual usage, only set it if we fail ...	
r682685	astitcher	2008-08-05	Modified error checking on TCP socket read so that it's no longer fatal	
r682688	gsim	2008-08-05	Merged r682685: Modified error checking on TCP socket read so that it's no longer fatal	
r682710	tross	2008-08-05	QPID-1214 - Committed William's patch	
r682711	tross	2008-08-05	Removed spurious print	
r682764	tross	2008-08-05	Restructured qpid-tool commands to allow active-only lists	
r682774	aconway	2008-08-05	Fix sporadic shutdown hang in clustered broker. Add start-stop_cluster test scripts	
r682785	gsim	2008-08-05	<ul style="list-style-type: none"> revised approach for setting tcp-nodelay on client to avoid breaking platform abstractions * added ability to set tcp-nodelay on server side of the socket also 	
r682791	gsim	2008-08-05	<ul style="list-style-type: none"> revised approach for setting tcp-nodelay on client to avoid breaking platform abstractions * added ability to set tcp-nodelay on server side of the socket also Merged from r682785. 	
r682885	aconway	2008-08-05	Fix Cluster::send encode race.	
r682887	rhs	2008-08-05	Profiling driven changes: - made AMQShortString cache the toString() value - added static initializer to IoTransport to disable use of pooled byte buffers - modified IoSender to permit buffer ...	
r682915	rhs	2008-08-05	QPID-1219 : cleanup of prior commit (r682887)	
r682979	cctrieloff	2008-08-05	Added actions for ACL on mgmt actions	
r683087	cctrieloff	2008-08-06	correct action on purge & remove ROUTINGKEY type	
r683301	gsim	2008-08-06	Merged r681164.	
r683337	rhs	2008-08-06	QPID-1221 : added customizable UUID generation and switched the default strategy to use nameUUIDFromBytes rather than randomUUID	
r683402	astitcher	2008-08-06	Refactor Thread platform code so that the implementation is completely decoupled from its interface	
r683416	aconway	2008-08-06	<ul style="list-style-type: none"> Added OutputTask::hasOutput() test. - Cluster only sends doOutput events when hasOutput() 	
r683437	rhs	2008-08-06	QPID-1222 : round up the buffer size to the nearest power of two	
r683560	gsim	2008-08-07	Remove reference to deleted sys/posix/Thread.h	
r683583	aidan	2008-08-07	QPID-1218 : Boost broker performance by lots. AMQMessage: Allow references to be incremented in a pile IncomingMessage: Increment message references in one go, flatten delivery loop a little. Make _d ...	TopicExchange: init array size
r683588	gsim	2008-08-07	Updated suppressions for changes to Thread	
r683595	gsim	2008-08-07	Removed recursive patch to patch	
r683597	aidan	2008-08-07	QPID-1218 : fix stupid used-only-by-tests method breakage that I have exposed	
r683603	gsim	2008-08-07	Updated suppressions	
r683617	aconway	2008-08-07	Patch from Gordon Sim to fix issues with hasOutput implementation.	
r683619	tross	2008-08-07	On broker shutdown, re-join the timer thread outside of a locked scope to prevent deadlock.	
r683632	ritchier	2008-08-07	QPID-1195 , QPID-1193 Initial changes to allow bind and queue arguments to be stored and recovered from the MessageStore. Created a test to validate that the stored values can be recovered. DerbyStore ...	Passing in null for stuff sucks.

r683635	ritchiem	2008-08-07	QPID-1182 : Some of the NullPointerExceptions from the SimpleACLTest are due to the close and the notification overlapping due to the lack of locking. The problem is that the AtomicBoolean _closed is ...	
r683683	rhs	2008-08-07	QPID-1213 : Patch from rgodfrey to refactor AbstractJMSMessage and descendants to move AMQP version specific code into delegates and remove unnecessary conversion between 0-8 and 0-10 objects	
r683711	aconway	2008-08-07	Check CPG flow control.	
r683744	rhs	2008-08-07	QPID-1213 : removed empty .java files leftover from applying a patch	
r683932	aidan	2008-08-08	QPID-1224 : add methods to get the list of message ids from a queue, with optional offset. Test class for this.	
r683941	rhs	2008-08-08	QPID-1213 : fixed a performance regressing from converting uuid -> string and back again	
r683947	ritchiem	2008-08-08	QPID-1225 cause test to fail if it times out.	
r683948	ritchiem	2008-08-08	Reverted the addition of *.rej and *.orig so we can see them in a status list.	
r683949	ritchiem	2008-08-08	QPID-1136 : Provided a fix for the leak in UnacknowledgedMessage when acking. Added a new InternalBrokerBaseCase for performing testing on the broker without using the client libraries. This allows f ...	
r683950	ritchiem	2008-08-08	QPID-1223 : added ApplicationRegistry.remove. Need to convert to QTC.	
r683955	ritchiem	2008-08-08	QPID-1226 : DupsOk test never creates the client so create one for the messages. Also improved the testing to ensure we check for failure scenarios.	

Jiras

Key	Component(s)	Affects Version/s	Summary	Status	Assignee	Reporter	Review Comments
QPID-1206	Java Client	M3	(QPID-1206) Failover Tests fail	Resolved	Aidan Skinner	Aidan Skinner	
QPID-1207	Java Client	M3	(QPID-1207) 0-10 client throws runtime exceptions when talking to a 0-9 broker	Open	Rafael H. Schlomig	Aidan Skinner	Needs resolved
QPID-1208	Java Client	M3	(QPID-1208) Java client logs password in plain text when info level is enabled	Open	Rajith Attapattu	Rajith Attapattu	Needs resolved
QPID-1209	C++ Client	M3	(QPID-1209) Port to Windows	Open	Andrew Stitzer	Steve Huston	
QPID-1210	Java Common	M3	(QPID-1210) qpid-run output is confusing for command line scripts	In Progress	Rafael H. Schlomig	Rafael H. Schlomig	Needs Resolved
QPID-1211	Interop Testing	M3	(QPID-1211) C++ interop test runner does not build.	Open	Unassigned	Martin Ritchie	
QPID-1212	C++ Client		(QPID-1212) C++ client trunk cannot be used in interop testing, as it is 0-10 only and cannot interop with the java broker.	Open	Unassigned	Martin Ritchie	
QPID-1213	Java Client	M3	(QPID-1213) UnprocessedMessage needs to die	Open	Rafael H. Schlomig	Aidan Skinner	
QPID-1214	C++ Broker, Python Test Suite		(QPID-1214) Requirement to be able to purge the top item or top n items from a queue.	Resolved	Ted Ross	William Henry	
QPID-1215	Java Tools		(QPID-1215) JNDICheck uses 1.6 specific methods Propertites.load(Reader)	Resolved	Martin Ritchie	Martin Ritchie	
QPID-1216	Java Client	M3	(QPID-1216) ConnectionCloseTest failed with 0-10 client	Open	Rafael H. Schlomig	Aidan Skinner	
QPID-1217	Java Client	M2, M2.1, M3	(QPID-1217) AMQSession 0-8 createTemporaryQueue does not perform the creation on the broker.	Open	Unassigned	Martin Ritchie	
QPID-1218	Java Broker	M3	(QPID-1218) Qpid Broker has slowed dramatically	Open	Aidan Skinner	Aidan Skinner	
QPID-1219	Java Client	M3	(QPID-1219) profiling related changes for the java 0-10 client	Resolved	Rafael H. Schlomig	Rafael H. Schlomig	

QPID-1220	Java Client	M3	(QPID-1220) ConnectionTest testClosedNotificationAndWriteToClosed fails intermittently	Open	Rafael H. Schloeming	Aidan Skinner	
QPID-1221	Java Client	M3	(QPID-1221) UUID.randomUUID() is really slow	Resolved	Rafael H. Schloeming	Rafael H. Schloeming	
QPID-1222		M3	(QPID-1222) IoSender breaks when the send buffer is not a power of 2	Resolved	Rafael H. Schloeming	Rafael H. Schloeming	
QPID-1223	Java Tests		(QPID-1223) All System tests not extending QpidTestCase do not correctly clean up VM State.	Open	Unassigned	Martin Ritchie	
QPID-1224	Java Broker	M3	(QPID-1224) No way to get list of message ids off of queue	Resolved	Aidan Skinner	Aidan Skinner	
QPID-1225	Java Tests	M3	(QPID-1225) DupsOk doesn't fail if it times out waiting for messages to be received.	Resolved	Martin Ritchie	Martin Ritchie	
QPID-1226	Java Tests		(QPID-1226) DupsOk doesn't create the consumer so will always fail to receive messages.	Open	Unassigned	Martin Ritchie	

Qpid Java Meeting Minutes 2008-08-15

Agenda

Commits review
JIRA Review
AOCB

Outstanding actions

revision	committer	date	comment	review comments
r659163	arnaudsimon	2008-05-22	QPID-1079 : Updated ...test.client tests for using QpidTestCase + move QpidTestCase in main so it is visible from systests	RECEIVE_TIMEOUT : get rid of and use configurable timeout when available
r669431	rgodfrey	2008-06-19	QPID-950 : Broker refactoring, copied / merged from branch	
r669480	rgodfrey	2008-06-19	QPID-950 : Fixed Derby Message Store	
r661746	ritchier	2008-05-30	QPID-1101 : Update to DestNameExchange to perform deep copy.	Needs test
r662755	arnaudsimon	2008-06-03	QPID-1115 : Only generate client ID when necessary	RG to comment on Jira
r662770	ritchier	2008-06-03	QPID-1092 : Changed toString to be String.valueOf(getObject()) Added MessageToStringTest, tests performing toString on Message before calling getObject().	Weird catch in close()
r662827	arnaudsimon	2008-06-03	QPID-1112 : Update previous commit by re-using messageAcknowledge (added a flag specifying whether to send an messageAccept)	inRecover check in BMC_0_10.postDeliver might be a problem with async delivery
r665841	rhs	2008-06-09	QPID-901 : always reset the auto-sync mode even if the call fails	RHS: make sure flag is used where appropriate
r669841	rgodfrey	2008-06-20	QPID-1144 : Reference count drops to zero too early for immediate messages in a tx	RG: document LocalTransactionalContext
r669885	rgodfrey	2008-06-20	QPID-1101 : Updated Direct Exchange so it does not modify lists of queues	No Test
r671949	ritchier	2008-06-26	QPID-909 : Commented out the TimerTask so that it can be wrapped with a ScaledTestDecorator. Minimal change to get our existing tests to run. If closer duration control is required then further time c ...	Change Commit message to reflect content : Update to .gitignore - DONE
r680602	rhs	2008-07-29	QPID-1201 : fixed up version of aidan's patch, there are still failures when running against an external java broker, however we seem to get past basic connection negotiation now	Need to check that this works with 0-8 only broker

r681476	rhs	2008-07-31	QPID-1210 : made qpid-run output level configurable	Fix setting of level in qpid-server to include \ or indent properly
r683583	aidan	2008-08-07	QPID-1218 : Boost broker performance by lots. AMQMessage: Allow references to be incremented in a pile IncomingMessage: Increment message references in one go, flatten delivery loop a little. Make _d ...	TopicExchange: init array size
r683632	ritchiem	2008-08-07	QPID-1195 , QPID-1193 Initial changes to allow bind and queue arguments to be stored and recovered from the MessageStore. Created a test to validate that the stored values can be recovered. DerbyStore ...	Restore createQueue(queue) method remove calls with null. DONE r684714

Additional Actions

Rob : email list about Qpid testing strategy now that we have multiple performance testing frameworks.

Martin : Document InternalBrokerBaseCase

Aidan : Email list about logging of plain text passwords in debug logging.

revision	committer	date	comment	review notes
r683955	ritchiem	2008-08-08	QPID-1226 : DupsOk test never creates the client so create one for the messages. Also improved the testing to ensure we check for failure scenarios.	
r683989	rhs	2008-08-08	QPID-1210 : added missing \	
r684016	aidan	2008-08-08	QPID-1218 Optionally use IoTransport, it's hot, but doesn't pass all the tests yet.	
r684017	tross	2008-08-08	Ported from trunk: Usage of lockf for locking the data directory. This ensures that locks aren't left by crashed processes	
r684036	rhs	2008-08-08	QPID-1213 : simplified unprocessed message and moved version specific code into the _0_8 and _0_10 variants	FlowControllingBlockingQueue - revert - Add new JIRA for profiling change ByteBufferMessage - revert - add to profiling jira .transport.* - revert - add to profiling jira
r684182	rhs	2008-08-09	QPID-1218 : cleaned up the interface to IoTransport a bit; added IoAcceptor; fixed Session tracking of sync point; default JAVA inside qpid-run	
r684707	ritchiem	2008-08-11	QPID-1220 Updated assert to show exception message when there is no cause set	
r684708	ritchiem	2008-08-11	QPID-1223 : Updated tests to correctly close the ApplicationRegistry that were created during the test run by non QpidTestCase classes	
r684710	ritchiem	2008-08-11	QPID-1193 : re-added createQueue(AMQQueue queue) method, after code review call.	
r684711	ritchiem	2008-08-11	Update to review generator to have title on all columns and h2. for Jira section	
r684713	ritchiem	2008-08-11	QPID-1223 : Updated AckTest to correctly create and close the ApplicationRegistry	
r684714	ritchiem	2008-08-11	QPID-1193 : Actually removed the calls that pass in the ugly null	
r684785	gsim	2008-08-11	Added some extra info to example doc for xml exchange.	
r684787	astitcher	2008-08-11	Decouple the DispatchHandle from its clients by using a DispatchHandleRef class which can be deleted at any time, but will only cause the DispatchHandle to be deleted later when it's definitely no lon ...	
r684865	aconway	2008-08-11	Integrate CPG file descriptor into broker polling.	
r684880	aconway	2008-08-11	Added doxygen comments on using the Poller.	
r685104	ritchiem	2008-08-12	QPID-1136 : Fixed Flow Control problem due to this change and added test to validate that Flow Control is operating correctly	

r685115	aidan	2008-08-12	QPID-1092 : Merge 662770 to trunk from ritchiem: Changed toString to be String.valueOf(getObject()) Added MessageToStringTest, tests performing toString on Message before calling getObject().	
r685142	aidan	2008-08-12	QPID-1117 merge ritchiem's 662818: QPID-1117 : Added tests for all other message types. Refactored the common parts out of the objectTest.	
r685151	aidan	2008-08-12	QPID-615 : Merge rupertlssmiths 581293 QPID-615 , Added patched version of MINAs VM Pipe cleanup thread. Will replace once bug fix is in newer version of MINA.	
r685189	gsim	2008-08-12	use decimal literal (python 2.3 converts the old hex literal into a negative value)	
r685198	gsim	2008-08-12	Merged r685189 (specify literal as decimal for the sake of python 2.3)	
r685207	rhs	2008-08-12	QPID-1233 : made getStringProperty(nonexistent) return null instead of NPE	
r685218	rhs	2008-08-12	QPID-1235 : fixed setXXXProperty to check for empty strings	
r685237	aconway	2008-08-12	Move frame processing out of CPG dispatch queue for cluster. PollableQueue is a pollable in-memory queue, will probably move it to sys.	
r685273	gsim	2008-08-12	Add extra boost headers to dist tarball	
r685278	aconway	2008-08-12	Replace eventfd with more portable pipe implementation in PollableCondition.	
r685289	gsim	2008-08-12	Add extra boost headers to dist tarball	
r685317	aconway	2008-08-12	Queue cluster send frames, do cpg_mcast in separate thread, batching if possible. 5x thruput improvement 	
r685506	rhs	2008-08-13	removed dead code from message Echo utility, and added a message Sink utility	
r685536	rhs	2008-08-13	QPID-1236 : made setObjectProperty validate the passed in value	
r685871	ritchiem	2008-08-14	QPID-1077 : Update release target to correctly specify execute permissions on release targets	
r685874	arnaudsimon	2008-08-14	QPID-1239 : changed the way QPID_ARGS is set	
r685952	ritchiem	2008-08-14	QPID-1077 : Noticed that the defaults of 644, and 755 for files and directories were not being applied so forcibly set these values.~	
r685967	gsim	2008-08-14	distclean should delete qpid/framing/MaxMethodBodySize.h	
r686021	aconway	2008-08-14	echo with_ais_group errors to stderr.	
r686035	aconway	2008-08-14	Add option to run singleton cluster on default port.	
r686043	aconway	2008-08-14	Stop previous qpidd before running singleton cluster.	
r686068	rhs	2008-08-14	QPID-1244 : fix for NPE on broker initiated connection close, also preserve the connection close text for better error reporting	
r686071	rhs	2008-08-14	increased timeout for DupsOKTest to prevent intermittent failure on build machine	
r686136	rhs	2008-08-15	updated qpid.0-10/java to match trunk/qpid/java@686097	
r686172	rhs	2008-08-15	QPID-1245 : use notifyMessage rather than onMessage in setMessageListener so that messages from the synchronous queue actually get acked	
r686175	rhs	2008-08-15	updated qpid.0-10/java to match trunk/qpid/java@686172	

Jiras

Key	Component(s)	Affects Version/s	Summary	Status	Assignee	Reporter	Review Comments
-----	--------------	-------------------	---------	--------	----------	----------	-----------------

QPID-1223	Java Tests		(QPID-1223) All System tests not extending QpidTestCase do not correctly clean up VM State.	Open	Unassigned	Martin Ritchie	
QPID-1224	Java Broker	M3	(QPID-1224) No way to get list of message ids off of queue	Resolved	Aidan Skinner	Aidan Skinner	
QPID-1225	Java Tests	M3	(QPID-1225) DupsOk doesn't fail if it times out waiting for messages to be received.	Resolved	Martin Ritchie	Martin Ritchie	
QPID-1226	Java Tests		(QPID-1226) DupsOk doesn't create the consumer so will always fail to receive messages.	Resolved	Unassigned	Martin Ritchie	
QPID-1227	Java Tests		(QPID-1227) OutOfMemoryError running tests with 1.6 or 1.7 JDK	Closed	Unassigned	Nicholas Dronen	
QPID-1228	Java Broker	M3	(QPID-1228) DefaultManagedObject utilises ApplicationRegistry to get ObjectRegistry	Open	Unassigned	Martin Ritchie	
QPID-1229	Java Tests	M3	(QPID-1229) Move broker only tests from systests to broker package	Open	Martin Ritchie	Martin Ritchie	
QPID-1230	Java Tests	M3	(QPID-1230) Broker contains duplicated mock object functionality and tests.	Open	Martin Ritchie	Martin Ritchie	
QPID-1231	Java Broker		(QPID-1231) Adding auto creation ability of qpid Java broker through connection URL	Open	Unassigned	Rajika Kumarasiri	
QPID-1232	Dot Net Client	M3	(QPID-1232) Random acknowledge modes can be created	Open	Aidan Skinner	Aidan Skinner	
QPID-1233	Java Client	M3	(QPID-1233) AMQMessageDelegate_0_10 NPEs on getStringProperty("nonexistent")	Resolved	Rafael H. Schloeming	Rafael H. Schloeming	
QPID-1234		M3	(QPID-1234) VirtualhostConfiguration defaults the lack of exchange for queue binding to null not a ConfigurationError.	Open	Unassigned	Martin Ritchie	
QPID-1235	Java Client	M3	(QPID-1235) setBooleanProperty("", x) is supposed to throw an IllegalArgumentException	Resolved	Rafael H. Schloeming	Rafael H. Schloeming	
QPID-1236	Java Client	M3	(QPID-1236) setObjectProperty("foo", new Object()) should throw a MessageFormatException	Resolved	Rafael H. Schloeming	Rafael H. Schloeming	
QPID-1237	Java Client, JMS Compliance		(QPID-1237) Queue creation via the session does not create and bind the queue on the broker.	Open	Unassigned	Martin Ritchie	
QPID-1238	Java Broker	M3	(QPID-1238) Java Documentation issue for Windows users	Open	Unassigned	Arnaud Simon	
QPID-1239	Java Broker	M3	(QPID-1239) qpid-server.bat does not handle command line arguments	Resolved	Arnaud Simon	Arnaud Simon	
QPID-1240	Java Management Console		(QPID-1240) NullPointerException when setting access level in UserManagement Panel	Open	Unassigned	Martin Ritchie	
QPID-1241	Java Management Console		(QPID-1241) qpidmc.bat script does not cope with spaces (" ") in the path.	Open	Unassigned	Martin Ritchie	
QPID-1242	Java Management Console		(QPID-1242) Ability to view Binary Messages was lost in the merge	Open	Unassigned	Martin Ritchie	
QPID-1243	Ant Build System, Java Management Console		(QPID-1243) Provide mechanism to build management console	Open	Unassigned	Martin Ritchie	
QPID-1244	Java Client		(QPID-1244) The 0-10 java client sometimes NPEs when the broker initiates connection close	Open	Rafael H. Schloeming	Rafael H. Schloeming	

QPID-1245	Java Client	M3	(QPID-1245) message prefetched into the synchronous queue don't get acked when a message listener is set	Resolved	Rafael H. Scholomig	Rafael H. Scholomig	
-----------	-------------	----	--	----------	---------------------	---------------------	--

Qpid Java Meeting Minutes 2008-08-22

Agenda

Commits review
JIRA Review
AOCB

Outstanding actions

revision	committer	date	comment	review comments
r659163	arnaudsimon	2008-05-22	QPID-1079 : Updated ...test.client tests for using QpidTestCase + move QpidTestCase in main so it is visible from systests	RECEIVE_TIMEOUT : get rid of and use configurable timeout when available
r669431	rgodfrey	2008-06-19	QPID-950 : Broker refactoring, copied / merged from branch	
r669480	rgodfrey	2008-06-19	QPID-950 : Fixed Derby Message Store	
r661746	ritchier	2008-05-30	QPID-1101 : Update to DestNameExchange to perform deep copy.	Needs test
r662755	arnaudsimon	2008-06-03	QPID-1115 : Only generate client ID when necessary	RG to comment on Jira
r662770	ritchier	2008-06-03	QPID-1092 : Changed toString to be String.valueOf(getObject()) Added MessageToStringTest, tests performing toString on Message before calling getObject().	Weird catch in close()
r662827	arnaudsimon	2008-06-03	QPID-1112 : Update previous commit by re-using messageAcknowledge (added a flag specifying whether to send an messageAccept)	inRecover check in BMC_0_10.postDeliver might be a problem with async delivery
r665841	rhs	2008-06-09	QPID-901 : always reset the auto-sync mode even if the call fails	RHS: make sure flag is used where appropriate
r669841	rgodfrey	2008-06-20	QPID-1144 : Reference count drops to zero too early for immediate messages in a txn	RG: document LocalTransactionalContext
r669885	rgodfrey	2008-06-20	QPID-1101 : Updated Direct Exchange so it does not modify lists of queues	No Test
r680602	rhs	2008-07-29	QPID-1201 : fixed up version of aidan's patch, there are still failures when running against an external java broker, however we seem to get past basic connection negotiation now	Need to check that this works with 0-8 only broker
r681476	rhs	2008-07-31	QPID-1210 : made qpid-run output level configurable	Fix setting of level in qpid-server to include \ or indent properly
r683583	aidan	2008-08-07	QPID-1218 : Boost broker performance by lots. AMQMessage: Allow references to be incremented in a pile IncomingMessage: Increment message references in one go, flatten delivery loop a little. Make _d ...	TopicExchange: init array size
r684036	rhs	2008-08-08	QPID-1213 : simplified unprocessed message and moved version specific code into the _0_8 and _0_10 variants	FlowControllingBlockingQueue - Revert change - Add new JIRA for profiling change ByteBufferMessage - revert change - add to profiling jira .transport.* - revert - add to profiling jira

Additional Actions

Rob : email list about Qpid testing strategy now that we have multiple performance testing frameworks.

Martin : Document InternalBrokerBaseCase

Aidan : Email list about logging of plain text passwords in debug logging.

RG : JIRA + Commit Build.xml

RHS : JIRA : Add QTC

RHS : Remove forked execution mode from tests

Rob : New VM Transport, two of them one that encodes one that sends objects

Commits

revision	committer	date	comment	review notes
r686722	ritchiem	2008-08-18	QPID-1226 : Last few changes to correctly shutdown all ApplicationRegistries on each test run	
r686811	rhs	2008-08-18	QPID-1252 : modified tests to unsubscribe the durable subscriptions they create	
r686818	gsim	2008-08-18	QPID-1250 : Ensure broker receives session.detached before channel can be reused.	
r686843	rhs	2008-08-18	updated qpid.0-10/java to match trunk/qpid/java@686835	
r686846	aconway	2008-08-18	Configure --without-cpg by default for M3.	
r686851	gsim	2008-08-18	QPID-1250 : Ensure broker receives session.detached before channel can be reused. Merge of 686818 from trunk.	
r686852	kpvdr	2008-08-18	Added --dtx option to txtest for DTX transaction testing	
r686854	kpvdr	2008-08-18	Added --dtx option to txtest for DTX transaction testing	
r687010	aidan	2008-08-19	QPID-1202 : Rebind durable subscriptions if the arguments have changed TopicExchange: take field arguments into account when determining if topic binding already exists when binding, but not for regul ...	
r687108	tross	2008-08-19	Created a development branch for C++ broker management	
r687138	kpvdr	2008-08-19	Missing DTX recover code for --dtx mode in txtest	
r687139	rhs	2008-08-19	increased the timeout in the new DurableSubscriptionTest from 250 milliseconds to 1 second	
r687140	kpvdr	2008-08-19	Missing DTX recover code for --dtx mode in txtest	
r687141	kpvdr	2008-08-19	Forgot to remove unneeded comment	
r687142	kpvdr	2008-08-19	Forgot to remove unneeded comment	
r687156	gsim	2008-08-19	Build tweaks for distcheck: * ensure examples dirs are writable so executables can be compiled * cleanup files on distclean	
r687310	arnaudsimon	2008-08-20	qpid-1251: changed close method for closing the underlying socket on windows only.	
r687313	rhs	2008-08-20	QPID-1252 : remove the durable subscription when done with it, and increased another timeout	
r687361	gsim	2008-08-20	Remove 'clever' locking as it actually degrades performance.	
r687377	rhs	2008-08-20	branch for experimental work	
r687382	aidan	2008-08-20	QPID-1202 : TopicExchange.removeFilteredQueue: if there are no instances of the filter, it's ok to remove it.	
r687383	aidan	2008-08-20	QPID-1217 : make temporary queue creation actually create the temporary queue. AMQSession*: consolidate createTemporaryQueue into AMQSession. ConnectionTest: declare custom exchanges before testing t ...	
r687540	rhs	2008-08-21	added codegen for 1-0-draft xml	
r687664	ritchiem	2008-08-21	QPID-1225 : Temporary commit to allow CI systems to help diagnose cause of race condition. My guess is that the session is open but closes right after the isClosed call is done. So the client goes ...	
r687665	ritchiem	2008-08-21	Add Simple Request/Response Example from M2.x	
r687667	ritchiem	2008-08-21	Remove old crufty helper that is not used.	
r687688	ritchiem	2008-08-21	QPID-1225 : Remove SimpleACLTest from the test runs as the issues has been identified.	
r687741	ritchiem	2008-08-21	Stopped the broker closing the ProtocolSessions as this was causing the client to lock in Mina seemingly missing the notify for the CloseFuture and hangs indefinitely	
r687742	ritchiem	2008-08-21	QPID-1256 : Initial commit of build creator tool. Documentation to appear on Wiki. (http://cwiki.apache.org/confluence/display/qpid/Build+Creator)	

r687743	ritchiem	2008-08-21	QPID-1225 : Changed SimpleACLTest to use QpidTestCase so the failing test can be excluded. This change DOES NOT mean the test will run agains the external brokers. The test explicitly shutdowns the QT ...		
r687749	ritchiem	2008-08-21	Sorry went crazy with git and didn't meant to commit this change		
r687764	aidan	2008-08-21	QPID-1167 : reset queue notification lists when creating queues. Pull out defaults centrally.		
r687807	aidan	2008-08-21	Update version, NOTICE files.		
r687808	aidan	2008-08-21	Tag M3		
r687813	aconway	2008-08-21	Pre-buffering output strategy for cluster. Additional hooks in broker code, should not affect standalone broker.		
r687850	aconway	2008-08-21	Fix typo.		
r687872	aconway	2008-08-21	Use numeric version number 0.3 in AC_INIT.		
r688045	gsim	2008-08-22	Update & correct some of the notes included with the release.		

Jiras

Key	Component(s)	Affects Version/s	Summary	Status	Assignee	Reporter	Review Comments
QPID-1240	Java Management Console		(QPID-1240) NullPointerException when setting access level in UserManagement Panel	Open	Unassigned	Martin Ritchie	
QPID-1241	Java Management Console		(QPID-1241) qpidmc.bat script does not cope with spaces (" ") in the path.	Open	Unassigned	Martin Ritchie	
QPID-1242	Java Management Console		(QPID-1242) Ability to view Binary Messages was lost in the merge	Open	Unassigned	Martin Ritchie	
QPID-1243	Ant Build System, Java Management Console		(QPID-1243) Provide mechanism to build management console	Open	Unassigned	Martin Ritchie	
QPID-1244	Java Client		(QPID-1244) The 0-10 java client sometimes NPEs when the broker initiates connection close	Resolved	Rafael H. Schloeming	Rafael H. Schloeming	
QPID-1245	Java Client	M3	(QPID-1245) message prefetched into the synchronous queue don't get acked when a message listener is set	Resolved	Rafael H. Schloeming	Rafael H. Schloeming	
QPID-1246	Java Broker		(QPID-1246) Broker/bin scripts do not have execute permission set in svn	Resolved	Martin Ritchie	Martin Ritchie	
QPID-1247	Java Client		(QPID-1247) the qpid tests have a lot of boilerplate code that could be moved to a helper class or a base test case	Open	Unassigned	Rafael H. Schloeming	
QPID-1248	C++ Broker	M3	(QPID-1248) Add "last image caching" so a new subscriber immediately gets the most recently sent msg - PLEASE PLEASE!!	Open	Unassigned	andrew M	
QPID-1249	C++ Broker	M3	(QPID-1249) Distribution created on system where openais is not installed fails make check on system where it is	Open	Alan Conway	Gordon Sim	
QPID-1250	Python Client	M3	(QPID-1250) Race in detaching for 0-10 python client	Resolved	Gordon Sim	Gordon Sim	
QPID-1251	Java Client	M3	(QPID-1251) IO transport does not cleanly close on Windows	Resolved	Arnaud Simon	Arnaud Simon	
QPID-1252		M3	(QPID-1252) various tests create durable subscriptions and then leave them there	Resolved	Rafael H. Schloeming	Rafael H. Schloeming	
QPID-1253		M4	(QPID-1253) Add an option to sync after 'n' bytes.	Open	Unassigned	Rajith Attapattu	

QPID-1254			(QPID-1254) Testing Qpid workflow mods	Resolved	Unassigned	Gavin	
QPID-1255		M3	(QPID-1255) SimpleACLTest testClientPublishInvalidQueueSuccess intermittent failure in CI	In Progress	Unassigned	Martin Ritchie	
QPID-1256			(QPID-1256) Provide build tool for combining source and patches from a variety of sources.	In Progress	Martin Ritchie	Martin Ritchie	
QPID-1257			(QPID-1257) Allow individual broker and client binary packages to be built	Open	Unassigned	Martin Ritchie	
QPID-1258	Java Broker, Java Client, Java Common, Java Management Console, Java Tests, Java Tools	M3	(QPID-1258) Releases missing license files etc.	Open	Aidan Skinner	Martin Ritchie	

Documentation

Documentation

Getting Started

- Download
- Getting Started

AMQP (Advanced Message Queueing Protocol)

- Toward a Commodity Enterprise Middleware
- AMQP (Advanced Message Queueing Protocol)

Qpid AMQP Brokers

- AMQP Messaging Broker (implemented in C++)
- AMQP Messaging Broker (implemented in Java)

Qpid AMQP Clients

- AMQP Java JMS Messaging Client
- AMQP C++ Messaging Client
- AMQP .NET Messaging Client
- AMQP Python Messaging Client
- AMQP Ruby Messaging Client

Interoperability

- AMQP Compatibility
- SASL Interoperability

FAQ

- Frequently Asked Questions

Build Creator

Overview

- Overview
- Build Creator
 - Command Line arguments
 - Targets
 - retrieve
 - prepare
 - release
- Configuration Details
 - Config file
 - Source Section
 - Source Types

- Patch Section
- Builds section
- Build file
 - Build Section
 - Writing release scripts
- Example Files

Build Creator

The BuildCreator tool was devised to enable the easy generation of binary packages that include more than one source. The purpose of the build creator is to simplify the building of projects that require the combining of multiple sources. To aid discussion of the tool we shall look at the configuration file combining the Apache Qpid release with that of the BerkeleyDB Store plugin from JBoss. However, the tool is a general binary build tool written in Python.

Command Line arguments

There are various command line arguments, mainly to change the amount of logging the tool generates.

```
subprocess is required for this tool and is not present in versions prior to 2.4.0
usage: buildCreator.py [options]

options:
-h, --help           show this help message and exit
-c, --config=CONFIG  set configuration file : default = build.config
-v, --verbose         enable verbose output
-d, --debug           enable debug output
-q, --quiet           Enable quiet output
-i, --ignore-errors   Ignore errors
```

There are a number of available targets:

Available Targets:	distclean [source] - Remove all or specified retrieved source clean [source] - Remove all or specified source build directory retrieve [source] - Retrieve all or specified source prepare [source] - Prepare all or specified source : i.e. extract archives patch [source] - Patch all or specified source showbuilds - List all builds build [build] - Perform the build scripts for all or specified build release [build] - Perform the release scripts for all or specified source full - Perform clean, retrieve, prepare, patch, build, release for all builds (DEFAULT)
--------------------	---

Targets

The default target is full which as listed above will perform all the required steps to create all the releases for the builds listed in the configuration. Most of the targets do exactly what they say above however a few have additional points of interest:

retrieve

The *retrieve* target pulls together all the source and patches and places this pristine copy in the *builder/src* directory. This is done so that the build can be repeatable performed with the untouched source.

prepare

The *prepare* target creates the *builder/build* tree and copies the source and patches for the later stages. If the source was not retrieved *svn* then the files are inspected and if an archive format is found this is expanded into the build directory.

release

The release scripts that are specified in the build files can contain a version keyword substitution *\$writeVersions(file)*. This will append details about the sources and patches used to generate this release artefact. If *svn* targets are used then the revision information will be added to the file.

Configuration Details

Two files are currently required by the tool the main configuration file and the build file.

Config file

The main configuration file (build.config by default) specifies the various sources, patches and builds that should be utilised.

The configuration file is used to define any common environment variables that may be used for substitution in the build scripts. The *<sources>* section contains a list of all source locations that the build will utilise. The *<patch>* section allows a number of patches to be

specified to apply to the pristinely downloaded source to be modified.

NOTE: Limitations

The build section can only contain `<include>` values not `<build>` elements directly

```
<builder>
    <environment>
        <[variable]>[value]</[variable]>
    </environment>

    <sources>
        <source>
            <name>[source-name]</name>
            <type>[source-type:svn|file|http|ftp]</type>
            <url>[value]</url>
            <path>[root offset, useful if to point the root of the source in to an archive
output]</path>
        </source>
    </sources>
    <patches>
        <patch>
            <name>[patch-name]</name>
            <type>[source-type:svn|file|http|ftp]</type>
            <url>[value]</url>
            <source>[source name this patches]</source>
            <prefix>[patch prefix -p value]</prefix>
            <path>[root offset, useful if the base of the patch is not the root of the
source]</path>
        </patch>
    </patches>

    <builds>
        <include>[string which is sent to ls to retrieve build include file so 'builds/*.build'
works]</include>
    </builds>
</builder>
```

Source Section

The `<source>` section contains a number of required values.

Entry	Description
<code><name></code>	The name you wish to give to this build. This value should not contain spaces as it can be used in the build and release scripts as a variable to refer to the build location.
<code><type></code>	There are four types of source svn , file , http , ftp . These are explained further below
<code><url></code>	This is the URL or file path to this source
<code><path></code>	Optional path if you wish to move the root of the source. i.e. in to an archive file

Source Types

The `<type>` of the determines how the creator will post process the file.

Type	Description
svn	Uses svn to retrieve the given <code><url></code> , An optional <code><revision></code> element can be added to source to be used by the svn checkout
file	Uses cp to take a copy of the specified file.
http	Uses wget to retrieve the specified file or directory, it currently does not recurse.
ftp	Uses wget to retrieve the specified file or directory, it currently does not recurse.

Patch Section

The patch section contains all of the same entries as the Source Section. It also has two additional entries that are used in applying the patches.

Entry	Description

<source>	Source name that this patch entry should be applied to.
<prefix>	This is the patch prefix (-p) value used on the command line.

The <url> value here can either be a file or a directory. In the case that it is a directory all files contained in that directory will be treated as patch files and applied to the specified <source> with the given options.

Builds section

The final section in the config file is the <builds> section. Currently this can only contain a single <include> value. This specifies the build scripts definitions that should be included in this configuration. Any value that is a valid argument to /s can be used here. This allows multiple files to be included. Future work includes the direct embedding of the build configuration.

Build file

The build file contains scripts to perform the build and release of a desired release. There is no restriction on what can be performed during either section however only one <build> may be contained in each file.

```

<builds>
  <build>
    <name>[build-name]</name>

    <dependency>
      <source>[source-name]</source>
    </dependency>

    <targets>
      <build>
        <script><![CDATA[
<shell script>
]]>
        </script>
      </build>

      <release>
        <script><![CDATA[
<shell script>
]]>
        </script>
      </release>
    </targets>

  </build>
</builds>

```

Build Section

The build section contains three elements all must be present to be valid.

Element	Description
<name>	The name of this build. This is used to name directories on disk and to refer to this build directly on the command line.
<dependency>	This is a list of <source> entries that this build requires.
<targets>	This contains the two scripts one for <build> and one for <release>. The text in these scripts are executed in a shell rooted in the current directory.

Writing release scripts

To ease the writing of release scripts there are a number of variables that have been predefined for your use. The source <name> values can be used as variables to refer to the root of that source's build location, in the examples below that would mean that \$qpid would be converted to builder/build/qpid. As mentioned above you can also define variables in the <environment> section of the configuration. Release scripts can contain a version keyword substitution \$writeVersions(file). This will append details about the sources and patches used to generate this release artefact. If svn targets are used then the revision information will be added to the file.

Example Files

```
<builder>
  <environment>
    <version>M3.0-beta</version>
  </environment>

  <sources>
    <source>
      <name>qpid</name>
      <type>file</type>
      <url>http://people.apache.org/~aidan/qpid/M3-beta/qpid-incubating-M3-beta.tar.gz</url>
      <path>qpid-incubating-M3</path>
    </source>
    <source>
      <name>bdb</name>
      <type>svn</type>
    </source>
    <url>https://svn.jboss.org/repos/rhessaging/store/branches/java/broker-queue-refactor/java/bdbstore</url>
  </sources>

  <patches>
    <patch>
      <name>BDB-Classpath</name>
      <type>file</type>
      <url>/local/patches/bdb-qpid-run-classpath.diff</url>
      <source>qpid</source>
      <prefix>2</prefix>
      <path>qpid-incubating-M3/qpid/java/<path>
    </patch>
  </patches>

  <builds>
    <include>builds/*.config</include>
  </builds>
</builder>
```

```

<builds>
    <build>
        <name>qpid-broker</name>

        <dependency>
            <source>[source-name]</source>
            <source>bdb</source>
        </dependency>

        <targets>
            <build>
                <script><![CDATA[
pushd $qpid/java
ant -Dproject.version=$version build
popd

cp $qpid/java/build/lib/qpid-broker-$version.jar $bdb/lib
cp $qpid/java/build/lib/qpid-broker-test-$version.jar $bdb/lib
cp $qpid/java/build/lib/qpid-common-$version.jar $bdb/lib
cp $qpid/java/build/lib/qpid-systests-$version.jar $bdb/lib
cp $qpid/java/build/lib/qpid-perftests-$version.jar $bdb/lib
cp $qpid/java/build/lib/qpid-junit-toolkit-$version.jar $bdb/lib

cd $bdb
ant build
]]>
                </script>
            </build>

            <release>
                <script><![CDATA[
# Create build package
mkdir -p $release/$build-$version
cp -r $qpid/java/build/* $release/$build-$version
cp $bdb/build/qpid-bdbstore.jar $bdb/lib/je-3.3.62.jar $release/$build-$version/lib

# Build release artifact
cd $release/$build-$version

# Create release revisions
echo "Qpid Broker Release : $version" > REVISIONS.txt
echo -n "Built:" >> REVISIONS.txt
date +%Y-%m-%d-%H%M >> REVISIONS.txt
$writeVersions(REVISIONS.txt)

cd ..
tar cvzf $build-$version.tgz $build-$version
]]>
                </script>
            </release>
        </targets>

        </build>
    </builds>

```

Cheat Sheet for configuring Exchange Options

Configuring Exchange Options

The C++ Broker M4 or later supports the following additional Exchange options in addition to the standard AMQP define options

- Exchange Level Message sequencing
- Initial Value Exchange

Note that these features can be used on any exchange type, that has been declared with the options set.

It also supports an additional option to the bind operation on a direct exchange

- Exclusive binding for key

Exchange Level Message sequencing

This feature can be used to place a sequence number into each message's headers, based on the order they pass through an exchange. The sequencing starts at 0 and then wraps in an AMQP int64 type.

The field name used is "qpidd.msg_sequence"

To use this feature an exchange needs to be declared specifying this option in the declare

```
...
    FieldTable args;
    args.setInt("qpidd.msg_sequence",1);

...
    // now declare the exchange
    session.exchangeDeclare(arg::exchange="direct", arg::arguments=args);
```

Then each message passing through that exchange will be numbers in the application headers.

```
unit64_t seqNo;
//after message transfer
seqNo = message.getHeaders().getAsInt64("qpidd.msg_sequence");
```

Initial Value Exchange

This feature caches a last message sent to an exchange. When a new binding is created onto the exchange it will then attempt to route this cached message to the queue, based on the binding. This allows for topics or the creation of configurations where a new consumer can receive the last message sent to the broker, with matching routing.

To use this feature an exchange needs to be declared specifying this option in the declare

```
...
    FieldTable args;
    args.setInt("qpidd.ive",1);

...
    // now declare the exchange
    session.exchangeDeclare(arg::exchange="direct", arg::arguments=args);
```

now use the exchange in the same way you would use any other exchange.

Exclusive binding for key

Direct exchanges in qpid support a qpid.exclusive-binding option on the bind operation that causes the binding specified to be the only one for the given key. I.e. if there is already a binding at this exchange with this key it will be atomically updated to bind the new queue. This means that the binding can be changed concurrently with an incoming stream of messages and each message will be routed to exactly one queue.

```
...
    FieldTable args;
    args.setInt("qpidd.exclusive-binding",1);

    //the following will cause the only binding from amq.direct with 'my-key'
    //to be the one to 'my-queue'; if there were any previous bindings for that
    //key they will be removed. This is atomic w.r.t message routing through the
    //exchange.
    session.exchangeBind(arg::exchange="amq.direct", arg::queue="my-queue",
                         arg::bindingKey="my-key", arg::arguments=args);

...
```

Cheat Sheet for configuring Queue Options

Configuring Queue Options

The C++ Broker M4 or later supports the following additional Queue constraints.

- Configuring Queue Options
 - Applying Queue Sizing Constraints
 - Changing the Queue ordering Behaviors (FIFO/LVQ)
 - Setting additional behaviors
 - Persist Last Node
 - Queue event generation
 - Other Clients

Applying Queue Sizing Constraints

This allows to specify how to size a queue and what to do when the sizing constraints have been reached. The queue size can be limited by the number messages (message depth) or byte depth on the queue.

Once the Queue meets/ exceeds these constraints the follow policies can be applied

- REJECT - Reject the published message
- FLOW_TO_DISK - Flow the messages to disk, to preserve memory
- RING - start overwriting messages in a ring based on sizing. If head meets tail, advance head
- RING_STRICT - start overwriting messages in a ring based on sizing. If head meets tail, AND the consumer has the tail message acquired it will reject

Examples:

Create a queue an auto delete queue that will support 100 000 bytes, and then REJECT

```
#include "qpid/client/QueueOptions.h"

QueueOptions qo;
qo.setSizePolicy(REJECT,100000,0);

session.queueDeclare(arg::queue=queue, arg::autoDelete=true, arg::arguments=qo);
```

Create a queue that will support 1000 messages into a RING buffer

```
#include "qpid/client/QueueOptions.h"

QueueOptions qo;
qo.setSizePolicy(RING,0,1000);

session.queueDeclare(arg::queue=queue, arg::arguments=qo);
```

Changing the Queue ordering Behaviors (FIFO/LVQ)

The default ordering in a queue in Qpid is FIFO. However additional ordering semantics can be used namely LVQ (Last Value Queue). Last Value Queue is define as follows.

If I publish symbols RHT, IBM, JAVA, MSFT, and then publish RHT before the consumer is able to consume RHT, that message will be overwritten in the queue and the consumer will receive the last published value for RHT.

Example:

```
#include "qpid/client/QueueOptions.h"

QueueOptions qo;
qo.setOrdering(LVQ);

session.queueDeclare(arg::queue=queue, arg::arguments=qo);

.....
string key;
qo.getLVQKey(key);

.....
for each message, set the into application headers before transfer
message.getHeaders().setString(key, "RHT");
```

Notes:

- Messages that are dequeued and the re-queued will have the following exceptions. a.) if a new message has been queued with the same key, the re-queue from the consumer, will combine these two messages. b.) If an update happens for a message of the same key, after the re-queue, it will not update the re-queued message. This is done to protect a client from being able to adversely manipulate the queue.
- Acquire: When a message is acquired from the queue, no matter its position, it will behave the same as a dequeue
- LVQ does not support durable messages. If the queue or messages are declared durable on an LVQ, the durability will be ignored.

A fully worked [LVQ Example](#) can be found here

Setting additional behaviors

Persist Last Node

This option is used in conjunction with clustering. It allows for a queue configured with this option to persist transient messages if the cluster fails down to the last node. If additional nodes in the cluster are restored it will stop persisting transient messages.

Note

- if a cluster is started with only one active node, this mode will not be triggered. It is only triggered the first time the cluster fails down to 1 node.
- The queue MUST be configured durable

Example:

```
#include "qpid/client/QueueOptions.h"

QueueOptions qo;
qo.clearPersistLastNode();

session.queueDeclare(arg::queue=queue, arg::durable=true, arg::arguments=qo);
```

Queue event generation

This option is used to determine whether enqueue/dequeue events representing changes made to queue state are generated. These events can then be processed by plugins such as that used for [queue state replication](#).

Example:

```
#include "qpid/client/QueueOptions.h"

QueueOptions options;
options.enableQueueEvents(1);
session.queueDeclare(arg::queue="my-queue", arg::arguments=options);
```

The boolean option indicates whether only enqueue events should be generated. The key set by this is 'qpid.queue_event_generation' and the value is an integer value of 1 (to replicate only enqueue events) or 2 (to replicate both enqueue and dequeue events).

Other Clients

Note that these options can be set from any client. QueueOptions just correctly formats the arguments passed to the QueueDeclare() method.

LVQ Example

Worked LVQ example

When running the following example, that happen when a message with the same key is published before the previous one if old message gets replaced. This example sends for messages with data & values set to key1, key2, key3, key1.

If the messages are enqueued before the listener consumes then you get the following output:

```

Sending Data:key1
Sending Data:key2
Sending Data:key3
Sending Data:key1
Sending Data:last
Receiving Data:key1
Receiving Data:key2
Receiving Data:key3
Receiving Data:last

```

Source for example

```

#include <qpid/client/Connection.h>
#include <qpid/client/SubscriptionManager.h>
#include <qpid/client/Session.h>
#include <qpid/client/Message.h>
#include <qpid/client/MessageListener.h>
#include <qpid/client/QueueOptions.h>

#include <iostream>

using namespace qpid::client;
using namespace qpid::framing;
using namespace qpid::sys;
using namespace std;

struct Args : public qpid::Options,
              public qpid::client::ConnectionSettings
{
    bool help;

    Args() : qpid::Options("Simple latency test optins"), help(false)
    {
        using namespace qpid;
        addOptions()
            ("help", optValue(help), "Print this usage statement")
            ("broker,b", optValue(host, "HOST"), "Broker host to connect to")
            ("port,p", optValue(port, "PORT"), "Broker port to connect to")
            ("username", optValue(username, "USER"), "user name for broker log in.")
            ("password", optValue(password, "PASSWORD"), "password for broker log in.")
            ("mechanism", optValue(mechanism, "MECH"), "SASL mechanism to use when
authenticating.")
            ("tcp-nodelay", optValue(tcpNoDelay), "Turn on tcp-nodelay");
    }
};

class Listener : public MessageListener
{
private:
    Session session;
    SubscriptionManager subscriptions;
    std::string queue;
    Message request;
    QueueOptions args;
public:
    Listener(Session& session);
    void setup();
    void send(std::string kv);
    void received(Message& message);
    void start();
};

Listener::Listener(Session& s) :
    session(s), subscriptions(s),
    queue(session.getId().getName())
{ }

void Listener::setup()

```

```

{
    // set queue mode
    args.setOrdering(LVQ);

    session.queueDeclare(arg::queue=queue, arg::exclusive=true, arg::autoDelete=true,
    arg::arguments=args);
    request.getDeliveryProperties().setRoutingKey(queue);

}

void Listener::start()
{
    subscriptions.subscribe(*this, queue, SubscriptionSettings(FlowControl::unlimited(),
ACCEPT_MODE_NONE));
    subscriptions.run();
}

void Listener::send(std::string kv)
{
    std::string key;
    args.getLVQKey(key);
    request.getHeaders().setString(key, kv);

    request.setData( kv);

    cout << "Sending Data:" << kv << std::endl;
    async(session).messageTransfer(arg::content=request);
}

void Listener::received(Message& response)
{
    cout << "Receiving Data:" << response.getData() << std::endl;
    if (response.getData() == "last"){
        subscriptions.cancel(queue);
    }
}

int main(int argc, char** argv)
{
    Args opts;
    opts.parse(argc, argv);

    if (opts.help) {
        std::cout << opts << std::endl;
        return 0;
    }

    Connection connection;
    try {
        connection.open(opts);
        Session session = connection.newSession();
        Listener listener(session);
        listener.setup();
        listener.send("key1");
        listener.send("key2");
        listener.send("key3");
        listener.send("key1");
        listener.send("last");
        listener.start();

        connection.close();
        return 0;
    } catch(const std::exception& error) {
        std::cout << error.what() << std::endl;
    }
    return 1;
}

```

```
}
```

queue state replication

Asynchronous Replication of Queue State

Overview

There is support in qpidd for selective asynchronous replication of queue state. This is achieved by:

- (a) enabling event generation for the queues in question
- (b) loading a plugin on the 'source' broker to encode those events as messages on a replication queue (this plugin is called replicating_listener.so)
- (c) loading a custom exchange plugin on the 'backup' broker (this plugin is called replication_exchange.so)
- (d) creating an instance of the replication exchange type on the backup broker
- (e) establishing a federation bridge between the replication queue on the source broker and the replication exchange on the backup broker

The bridge established between the source and backup brokers for replication (step (e) above) should have acknowledgements turned on (this may be done through the --ack N option to qpid-route). This ensures that replication events are not lost if the bridge fails.

The replication protocol will also eliminate duplicates to ensure reliably replicated state. Note though that only one bridge per replication exchange is supported. If clients try to publish to the replication exchange or if more than a single required bridge from the replication queue on the source broker is created, replication will be corrupted. (Access control may be used to restrict access and help prevent this).

The replicating event listener plugin (step (b) above) has the following options:

Queue Replication Options: --replication-queue QUEUE --replication-listener-name NAME (replicator) --create-replication-queue	Queue on which events for other queues are recorded name by which to register the replicating event listener if set, the replication will be created if it does not exist
--	--

The name of the queue is required. It can either point to a durable queue whose definition has been previously recorded, or the --create-replication-queue option can be specified in which case the queue will be created a simple non-durable queue if it does not already exist.

Use with Clustering

The source and/or backup brokers may also be clustered brokers. In this case the federated bridge will be re-established between replicas should either of the originally connected nodes fail. There are however the following limitations at present:

- The backup site does not process membership updates after it establishes the first connection. In order for newly added members on a source cluster to be eligible as failover targets, the bridge must be recreated after those members have been added to the source cluster.
- New members added to a backup cluster will not receive information about currently established bridges. Therefore in order to allow the bridge to be re-established from these members in the event of failure of older nodes, the bridge must be recreated after the new members have joined.
- Only a single URL can be passed to create the initial link from backup site to the primary site. This means that at the time of creating the initial connection the initial node in the primary site to which the connection is made needs to be running. Once connected the backup site will receive a membership update of all the nodes in the primary site, and if the initial connection node in the primary fails, the link will be re-established on the next node that was started (time) on the primary site.

Due to the acknowledged transfer of events over the bridge (see note above) manual recreation of the bridge and automatic re-establishment of the bridge after connection failure (including failover where either or both ends are clustered brokers) will not result in event loss.

Operations on Backup Queues

When replicating the state of a queue to a backup broker it is important to recognise that any other operations performed directly on the backup queue may break the replication.

If the backup queue is to be an active (i.e. accessed by clients while replication is on) only enqueues should be selected for replication. In this mode, any message enqueued on the source brokers copy of the queue will also be enqueued on the backup brokers copy. However no attempt will be made to remove messages from the backup queue in response to removal of messages from the source queue.

Selecting Queues for Replication

Queues are selected for replication by specifying the types of events they should generate (it is from these events that the replicating plugin constructs messages which are then pulled and processed by the backup site). This is done through options passed to the initial queue-declare command that creates the queue and may be done either through qpid-config or similar tools, or by the application.

With qpid-config, the --generate-queue-events option is used:

```
--generate-queue-events N
    If set to 1, every enqueue will generate an event that can be processed
by
    registered listeners (e.g. for replication). If set to 2, events will be
    generated for enqueues and dequeues
```

From an application, the arguments field of the queue-declare AMQP command is used to convey this information. An entry should be added to the map with key 'qpid.queue_event_generation' and an integer value of 1 (to replicate only enqueue events) or 2 (to replicate both enqueue and dequeue events).

Applications written using the c++ client API may find the qpid::client::QueueOptions class convenient. This has a enableQueueEvents() method on it that can be used to set the option (the instance of QueueOptions is then passed as the value of the arguments field in the queue-declare command. The boolean option to that method should be set to true if only enqueue events should be replicated; by default it is false meaning that both enqueues and dequeues will be replicated. E.g.

```
QueueOptions options;
options.enableQueueEvents(false);
session.queueDeclare(arg::queue="my-queue", arg::arguments=options);
```

Example

Lets assume we will run the primary broker on host1 and the backup on host2, have installed qpidd on both and have the replicating_listener and replication_exchange plugins in qpidd's module directory(*1).

On host1 we start the source broker and specify that a queue called 'replication' should be used for storing the events until consumed by the backup. We also request that this queue be created (as transient) if not already specified:

```
qpidd --replication-queue replication-queue --create-replication-queue true --log-enable info+
```

On host2 we start up the backup broker ensuring that the replication exchange module is loaded:

```
qpidd
```

We can then create the instance of that replication exchange that we will use to process the events:

```
qpid-config -a host2 add exchange replication replication-exchange
```

If this fails with the message "Exchange type not implemented: replication", it means the replication exchange module was not loaded. Check that the module is installed on your system and if necessary provide the full path to the library.

We then connect the replication queue on the source broker with the replication exchange on the backup broker using the qpid-route command:

```
qpidd-route --ack 50 queue add host2 host1 replication-exchange replication-queue
```

The example above configures the bridge to acknowledge messages in batches of 50.

Now create two queues (on both source and backup brokers), one replicating both enqueues and dequeues (queue-a) and the other replicating only dequeues (queue-b):

```

qpidd-config -a host1 add queue queue-a --generate-queue-events 2
qpidd-config -a host1 add queue queue-b --generate-queue-events 1

qpidd-config -a host2 add queue queue-a
qpidd-config -a host2 add queue queue-b

```

We are now ready to use the queues and see the replication.

Any message enqueued on queue-a will be replicated to the backup broker. When the message is acknowledged by a client connected to host1 (and thus dequeued), that message will be removed from the copy of the queue on host2. The state of queue-a on host2 will thus mirror that of the equivalent queue on host1, albeit with a small lag. (Note however that we must not have clients connected to host2 publish to-or consume from- queue-a or the state will fail to replicate correctly due to conflicts).

Any message enqueued on queue-b on host1 will also be enqueued on the equivalent queue on host2. However the acknowledgement and consequent dequeuing of messages from queue-b on host1 will have no effect on the state of queue-b on host2.

(*1) If not the paths in the above may need to be modified. E.g. if using modules built from a qpid svn checkout, the following would be added to the command line used to start qpidd on host1:

```
--load-module <path-to-qpid-dir>/src/.libs/replicating_listener.so
```

and the following for the equivalent command line on host2:

```
--load-module <path-to-qpid-dir>/src/.libs/replication_exchange.so
```

Documentation2

- General
- Qpid Brokers
 - Java Broker
 - General User Guides
 - How Tos
 - Management Tools
 - C++ Broker
 - General User Guides
 - How Tos
 - Management
- Qpid Clients
 - JMS Client
 - General User Guides.
 - C++ Client
 - General User Guides
 - How Tos
 - Python Client
 - General User Guides
 - How Tos
 - Ruby Client
 - General User Guides
 - How Tos
 - .NET Client
 - General User Guides
 - How Tos
- Management Tools
 - C++ Broker Management Tools
 - Java Broker Management Tools
- Developer Guides
- Development Tools

General

- FAQ
- Quick start guide
- Example guide
- Qpid Interoperability Documentation

Qpid Brokers

Java Broker

General User Guides

- Feature Guide
- FAQ
- Getting Started Guide
- Broker Environment Variables
- Troubleshooting Guide

How Tos

- Add New Users
- Configure ACLs
- Configure Java Qpid to use a SSL connection.
- Configure Log4j CompositeRolling Appender
- Configure the Broker via config.xml
- Configure the Virtual Hosts via virtualhosts.xml
- Debug using log4j
- How to Tune M3 Java Broker Performance
- Qpid Java Build How To
- Use Priority Queues

Management Tools

- Qpid JMX Management Console
- MessageStore Tool
- Qpid Java Broker Management CLI
- Management Design notes

C++ Broker

General User Guides

- [Feature Guide]
- Running an AMQP 0-10 C++ broker
- Getting Started
- Queue State Replication
- Understanding ACLs
- Using Broker Federation
- [Clustering Guide]

How Tos

- How to use SSL
- [RDMA How To]
- [Kerberos Support]
- [SASL Support]

Management

- Management Tools Overview
- QMan - Qpid Management bridge
- Qpid Management Framework
- Manage anything with Qpid - QMF Python Console Tutorial
- Qpid Management Framework (QMF) Protocol

Qpid Clients

JMS Client

The Java Client supported by Qpid implements the JMS 1.1 specification.

General User Guides.

- [Feature Guide]
- FAQ
- JMS 1.1 Specification
- System Properties
- Connection URL Format - The format used to describe a connection.
- BindingURLFormat - The format used for creating bindings within and to a broker.
- How to Use JNDI
- [Using JMS client with RT Java]
- [JMS Client Tuning Guide]

C++ Client

General User Guides

- [Feature Guide]
- C++ API Guide
- Configuring Queue Options
- Configuring Exchange Options
- Understanding Last Value Queues (LVQ)

How Tos

- How to use SSL
- [Message TTL, auto expire]
- [RDMA How To]
- [Kerberos Support]
- [SASL Support]

Python Client

General User Guides

- Python Client API Guide
- Python Test Framework

How Tos

Ruby Client

General User Guides

- Ruby Client API Guide (todo)

How Tos

.NET Client

General User Guides

- .NET client user guide
- .NET client Excel plug-in
- The WCF interface for the .NET client

How Tos

Management Tools

C++ Broker Management Tools

- Management Tools Overview
- QMan - Qpid Management bridge
- Qpid Management Framework
- Manage anything with Qpid - QMF Python Console Tutorial
- Qpid Management Framework (QMF) Protocol

Java Broker Management Tools

- Qpid JMX Management Console
- MessageStore Tool
- Qpid Java Broker Management CLI

Developer Guides

- Qpid .Net Documentation
- Qpid Java Documentation
- Qpid 'C++' Documentation
- Qpid Python Test Framework

Development Tools

- Build Creator

DocumentationB

Qpid Documentation Index

General

- FAQ
- Quick start guide
- Example guide
- Qpid Interoperability Documentation

Qpid Brokers

- Java Broker
- C++ Broker

Qpid Clients

- JMS Client
- C++ Client
- Python Client
- Ruby Client
- .NET Client

Management Tools

- C++ Broker Management Tools
 - Management Tools Overview
 - QMan - Qpid Management bridge
 - Qpid Management Framework
 - Manage anything with Qpid - QMF Python Console Tutorial
 - Qpid Management Framework (QMF) Protocol
- Java Broker Management Tools
 - Qpid JMX Management Console
 - MessageStore Tool
 - Qpid Java Broker Management CLI

Developer Guides

- Qpid .Net Documentation
- Qpid Java Documentation
- Qpid 'C++' Documentation
- Qpid Python Test Framework

Development Tools

- Build Creator

.NET Client

General User Guides

- .NET client user guide
- .NET client Excel plug-in
- The WCF interface for the .NET client

C++ Broker

General User Guides

- [Feature Guide]
- Running an AMQP 0-10 C++ broker
- Getting Started
- Queue State Replication
- Understanding ACLs
- Using Broker Federation
- [Clustering Guide]

How Tos

- How to use SSL
- [RDMA How To]
- [Kerberos Support]

- [SASL Support]

Management

- Management Tools Overview
- QMan - Qpid Management bridge
- Qpid Management Framework
- Manage anything with Qpid - QMF Python Console Tutorial
- Qpid Management Framework (QMF) Protocol

Management Tools Overview

C++ Broker Management Tools Overview

- qpid-tool - telnet type tool to access data, view schema, issue command an and QMF resource
- qpid-config - tool to configure queues, exchanges, etc. all the details on the AMQP model
- qpid-route - tool to configure broker federation
- qpid-events - utility that will print to cmd line or syslog event from a broker like, userconnected, user created/deleted a queue.***
- qpid-stats * utility that will print out queue statistics to the cmd line or syslog like rate and message depth.
- QMan - accessing the above information via JMX or WS-DM (work in progress).

C++ Client

General User Guides

- [Feature Guide]
- C++ API Guide
- Configuring Queue Options
- Configuring Exchange Options
- Understanding Last Value Queues (LVQ)

How Tos

- How to use SSL
- [Message TTL, auto expire]
- [RDMA How To]
- [Kerberos Support]
- [SASL Support]

Example guide

Introduction

Qpid includes a set of examples using JMS, C++, Python and NET Clients (Ruby to be done soon.)
These examples are designed to interoperate with each other.

(Currently you can only use the c++ broker to demonstrate the interoperability as the java broker doesn't support AMQP 0-10 yet).

Example Structure

The examples demonstrates the following messaging use cases

- - Use of direct exchange
 - Use of topic exchange
 - Use of fanout exchange
 - Request/Reply pattern.

Getting the examples

You could get the examples by downloading the latest release from [here](#)
Or you could build them from source. Check the following URL's to browse the source code.

- C++ Examples: <https://svn.apache.org/repos/asf/qpid/trunk/qpid/cpp/examples/>
- Java JMS Examples: <https://svn.apache.org/repos/asf/qpid/trunk/qpid/java/client/example/>
- Python Examples: <https://svn.apache.org/repos/asf/qpid/trunk/qpid/python/examples/>
- Ruby Examples: <https://svn.apache.org/repos/asf/qpid/trunk/qpid/ruby/examples/>
- .NET Examples: <http://svn.apache.org/viewvc/qpid/trunk/qpid/dotnet/client-010/examples/>

Running the examples

todo

Java Broker

General User Guides

- Feature Guide
- FAQ
- Getting Started Guide
- Broker Environment Variables
- Troubleshooting Guide

How Tos

- Add New Users
- Configure ACLs
- Configure Java Qpid to use a SSL connection.
- Configure Log4j CompositeRolling Appender
- Configure the Broker via config.xml
- Configure the Virtual Hosts via virtualhosts.xml
- Debug using log4j
- How to Tune M3 Java Broker Performance
- Qpid Java Build How To
- Use Priority Queues

Management Tools

- Qpid JMX Management Console
- MessageStore Tool
- Qpid Java Broker Management CLI
- Management Design notes

IP Whitelisting

While using a properly configured firewall is the obvious way to restrict access to a broker, it's occasionally desireable to do this on the broker itself.

Configuration

The access restrictions apply either to the server as a whole or too a particular virtualhost. Rules are evaluated in the virtualhost first, then the server as a whole (most-specific to least-specific). This allows whole netblocks to be restricted from all but one virtualhost. A <firewall> element would appear in either the <broker><security><access> section or inside the equivalent <virtualhost> element.

Elements inside <firewall> would be <rule> or <include file="[path"]/>. <include> would read the file specified at path, which would contain an <firewall host="hostname"/>.

<firewall> would contain further <rule> entries, but not <include>. If the host attribute was specified the broker would check it's hostname against the attribute and cause a fatal error on startup if it did not match.

<rule> would have action, hostname and network attributes. Action and one of host or network would be mandatory. The action attribute would be either allow or deny. Host contains a comma seperated list of regexps against which it would match the reverse dns lookup of the connecting IP. Network contains a comma seperated list of of CIDR networks against which the IP would be matched.

The first <rule> which matched the connection would apply. If no rules applied, the default-action would apply.

For example, the following could appear in config.xml:

```
<firewall default-action="deny">
  <rule permission="allow" hostname="*.qpid.apache.org"/>
  <include file="/path/to/file" />
  <rule permission="allow" network="192.168.1.0/24" />
  <rule permission="allow" network="10.0.0.0/8" />
</firewall>
```

and /path/to/file could contain:

```
<firewall host="broker1.qpid.apache.org">
  <rule permission="deny" network="192.168.1.0/24" virtualhost="prod"/>
</firewall>
```

any machine in the qpid.apache.org domain could access dev.

Any machine in the 192.168.1.0/24 network would be allowed access to any virtualhost other than prod

Any machine in the 10.0.0.0/8 network would be allowed access to any virtual host

Any other machine would be denied access.

Changes would be possible while broker was running via commons-configuration magic when the file is edited. Existing connections would be unaffected by a new rule.

Implementation

An IPRestriction class would extend ACLPlugin which listens for ConnectionOpen and checks against the list of rules. It will use the mechanism described at <http://qpid.apache.org/java-authorization-plugins.html>.

IPRestriction would parse the config file, compiling into an ordered list of Rule classes, which would have two methods: boolean match(InetAddress IPAddress) and boolean allow(). During the authorization phase it would iterate through these Rules until match() returns true when it will authorize or not according to the value returned by allow().

Because of the way that Java pre-6 caches dns forever, a small value for networkaddress.cache.ttl is necessary.

QPID-1583

Java Broker Feature Guide

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

Upcoming features:

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

JMS Client

The Java Client supported by Qpid implements the JMS 1.1 specification.

General User Guides.

- [Feature Guide]
- [FAQ](#)
- [JMS 1.1 Specification](#)
- [System Properties](#)
- [Connection URL Format](#) - The format used to describe a connection.
- [BindingURLFormat](#) - The format used for creating bindings within and to a broker.
- [How to Use JNDI](#)
- [Using JMS client with RT Java]
- [JMS Client Tuning Guide]

Python Client

General User Guides

- [Python Client API Guide](#)
- [Python Test Framework](#)

Ruby Client

General User Guides

- Ruby Client API Guide (todo)

Java Broker Analysis Tools

Analysis Tools

This page contains details of the broker analysis tools available as part of the [Performance Test] package. The design for this work is located [here](#).

- [Overview](#)
- [Monitoring](#)
 - GC / Heap Usage
 - CPU Usage

- Scripting
- Processing
- processTests.py
- processAll.sh
- process.sh

Overview

To better understand the performance of the Java broker this collection of tools have been gathered to perform analysis on a variety of logging that the broker can produce. Looking solely at the throughput values from our performance suite is not sufficient to tell us that the broker's performance has increased.

Currently it the scripts monitor:

- Heap Usage via verbose GC logging
- GC Duration via verbose GC logging
- CPU Usage via batch mode top

Additional logging can be added to gather data as required. The processing of the resulting log files from the broker run can then processed and using [GnuPlot](#) graphs of the data are generated.

Monitoring

To better understand how the broker is performing there are some easy things we can start monitoring.

- Verbose GC/Heap Usage
- CPU Usage

GC / Heap Usage

Enabling verbose gc will allow the broker to provide us with a log file that details GC operation. SO we can get a better handle on the impact of GC on the performance of the broker. Enabling is done by providing a few additional values via QPID_OPTS:

```
-Xloggc:<gc log file> -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps
```

This will result in a gc log file that shows all GCs performed. Of initial interest is the extraction of:

- Heap Usage (Max Allocated, Pre/Post GC Usage)
- GC Count (Incremental, Full)
- GC Duration (Incremental, Full, Total)

Testing+Design++Java+Broker+CPU+GC+Monitoring

As we gather this information a better internal view of the broker in operation can be built. When changes are made to the broker this data should allow us to determine how the changes have affected the GC and memory profile.

CPU Usage

In addition to the GC and Memory profiling available via the verbose gc settings monitoring the CPU usage via top is a quick and easy way to view the cpu utilisation of the broker.

Using the following top command we can monitor a give broker and adjust the time interval in which we record the CPU usage.

```
$ top -d $monitor_rate -S -c -p $broker_pid -b > broker_cpu.log
```

Scripting

To make life easier and to allow for future automated testing the following monitoring scripts have been added to the 'perftest' package:

- monitor-broker.sh
- stop-monitored-broker.sh

monitor-broker.sh

The monitor-broker.sh script current starts the following processes:

- The broker with additional QPID_OPTS for gc logging
- Top to monitoring the CPU usage

To run this script a number of parameters are required:

```
Usage ./monitor-broker.sh <Path to Test Broker> <LOG DIR> <CPU Monitor Rate (s)> [Additional options to pass to Qpid broker startup]
```

The first parameter is the path to a qpid-broker package, or at least a directory that contains an executable broker. The <LOG DIR> is a path that *must not exist* (this is checked) for all the log files to be collected for this monitored startup process. The

verbose gc, broker log, std out/err and PID files will be written to this directory.
The rate at which top will run to monitor this broker is the third value, any value here that is valid for top's '-d' parameter is valid here.
Finally the remainder of the command line is passed directly to the qpid-server start up so additional log configuration or configuration can be provided here. NOTE: if providing custom log4j setup please ensure that the log file is written to QPID_WORK as this is set to the <LOG DIR> value. This will ensure that all the log files for the testing run are located in a single directory for easy later processing.

The pids of the broker and top are written to a *.pid file in the LOG_DIR. These are also used by the stop-monitored-broker.sh script to ensure clean shutdown.

If additional processes are desired to be run if they write a PID into LOG_DIR/*.pid then they will be shutdown with the stop script.

stop-monitored-broker.sh

This is a simple script that takes the <LOG DIR> as its only parameter.

It then looks in this directory for all '.pid' files and uses their contained pid value to execute a 'kill' command.

If all processes have not stopped within 3 seconds then a 'kill -9' is executed.

runTests.sh

The final script in the monitoring package currently is 'runTest.sh' this simplifies the execution of a suite of tests.

The script takes three parameters:

```
Usage ./runTests.sh <Path to Test Pack> <LOG DIR> <TEST LIST FILE>
```

As with the monitor-broker.sh the first parameter is the path to the qpid-test package that can be build using the [Build Creator](#). The test simply looks for a path that has a bin which will be used to execute your test lists.

The LOG_DIR path again is required *not to exist* on startup to ensure we have a clean result set. The results of each test run are instructed to be written here by adding '-o <LOG_DIR> --csv' to the executed test.

Finally the 'Test List File' is a plain text file containing a single command per line to execute. The entries here should at least respond to the '-o' parameter to ensure that their output is collected in the LOG_DIR.

An example test file might contain something link:

```
TQBT-AA-Qpid-01.sh -d10M  
TTBT-AA-Qpid-01.sh -d10M
```

The two test scripts are assumed to exist in thee <path to test pack>/bin directory.

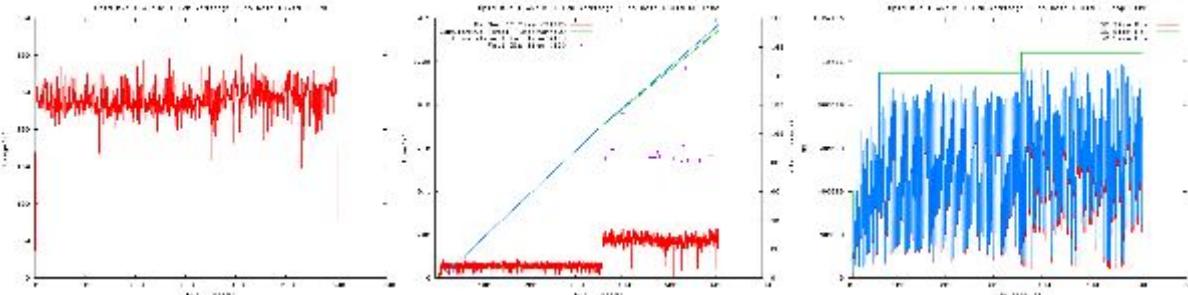
Monitoring

If you want to check the tests are running the standard out of the individual tests is redirected to <log dir>/TestRun.log

Processing

Monitoring is only the first stage to gather the data. It is the collection of processing tools that are responsible for turing the raw data into something more human understandable.

There are three scripts here that perform take the raw data from the monitoring phase and turn that in to three graphs such as these examples that were made during the design of these scripts:



The three scripts are:

- processTests.py
- processAll.sh
- process.sh

processTests.py

This is the first script written in python that takes the raw output from the monitoring stage and generates test packs.

The script has two parameters the two output directories (broker and test) from the monitoring phase:

```
Usage: processTests.py [-b|--broker-log-dir] <dir> [-t|--test-dir] <dir>
```

The tool currently looks for all the *.csv file the individual tests have generated and uses the gathered metadata to create a slice of each of the broker log files (gc, logging, cpu). In addition the script will gather details about the test run and broker used to form the title and filename for the graph.

processAll.sh

The processAll.sh script searches for 'graph.data' files and then runs the process.sh script on each of them to generate the graphs for that data.

The script takes a single argument, a directory to start searching in.

```
processAll.sh <search dir>
```

The graph images are then copied to a 'results' directory that is created in the current working directory.

process.sh

This is the main processing script for the collected data. It has been updated to work in conjunction with the processTests.py script. Further development of this script should be performed to allow the explicit naming of the various log file and parameter inputs that this uses.

This script is currently expected to be called from processAll.sh and as a result takes a single argument a graph.data file.

This graph.data file contains two text lines. The first is the title to give the graph, the second is the name of the file.
This is an example file of the automatic output from the processTests.py:

```
0.5:TQBT-AA-Qpid-01:256kb x 962 msg/sec using AutoAck  
0.5-TQBT-AA-Qpid-01-2009-06-19-17.04.25-timings
```

In generic terms it creates the following graph.data file:

```
<broker version>:<test name>:<messageSize>kb x <test volume as measured> msg/sec using <ackMode of  
test>  
<broker version>-<test name>
```

The process.sh script produces three graphs:

- GC Heap Usage
- GC Duration
- CPU Utilisation

NOTE GnuPlot is used to generate the graphs.

To generate these graphs it does a lot of data manipulation and extraction on the gc log file. Currently the script will process a ConcurrentMark and Sweep gc log file and the format used by the new G1 collector. The processing of these files extracts the recorded time for each gc and the instant count of minor and full GCs. This information is graphed on the GC Duration graph.

The GC log file also highlights the Allocated Heap, Pre and Post GC heap sizes. This is the data that is then graphed in the GC Heap Size graph.

The final graph, 'CPU Utilisation' is generated from the cpu data gathered using top. Here the script has been updated to work with the data output from processTests.py where the broker_cpu.log file contains a list of time-stamped entries. This is then used to show time on the x-axis.

This improvement is also due to be applied to the other two graphs. In addition to standardising the x-axis the y-axis scale for a given batch of tests, as processed by processAll.sh, will be standardised to allow for easy image comparison.

LVQ

Understanding LVQ

Last Value Queues are useful if you are only interested in the latest value entered into a queue. LVQ semantics are typically used for things like stock symbol updates when all you care about is the latest value for example.

Qpid C++ M4 or later supports two types of LVQ semantics:

- LVQ
- LVQ_NO_BROWSE

LVQ semantics:

LVQ uses a header for a key, if the key matches it replaces the message in-place in the queue except

- a.) if the message with the matching key has been acquired
- b.) if the message with the matching key has been browsed

In these two cases the message is placed into the queue in FIFO, if another message with the same key is received it will the 'un-accessed' message with the same key will be replaced

These two exceptions protect the consumer from missing the last update where a consumer or browser accesses a message and an update comes with the same key.

An example

```
[localhost tests]$ ./lvqtest --mode create_lvq
[localhost tests]$ ./lvqtest --mode write
Sending Data: key1=key1.0x7ffffdf3f3180
Sending Data: key2=key2.0x7ffffdf3f3180
Sending Data: key3=key3.0x7ffffdf3f3180
Sending Data: key1=key1.0x7ffffdf3f3180
Sending Data: last=last
[localhost tests]$ ./lvqtest --mode browse
Receiving Data:key1.0x7ffffdf3f3180
Receiving Data:key2.0x7ffffdf3f3180
Receiving Data:key3.0x7ffffdf3f3180
Receiving Data:last
[localhost tests]$ ./lvqtest --mode write
Sending Data: key1=key1.0x7ffffe4c7fa10
Sending Data: key2=key2.0x7ffffe4c7fa10
Sending Data: key3=key3.0x7ffffe4c7fa10
Sending Data: key1=key1.0x7ffffe4c7fa10
Sending Data: last=last
[localhost tests]$ ./lvqtest --mode browse
Receiving Data:key1.0x7ffffe4c7fa10
Receiving Data:key2.0x7ffffe4c7fa10
Receiving Data:key3.0x7ffffe4c7fa10
Receiving Data:last
[localhost tests]$ ./lvqtest --mode consume
Receiving Data:key1.0x7ffffdf3f3180
Receiving Data:key2.0x7ffffdf3f3180
Receiving Data:key3.0x7ffffdf3f3180
Receiving Data:last
Receiving Data:key1.0x7ffffe4c7fa10
Receiving Data:key2.0x7ffffe4c7fa10
Receiving Data:key3.0x7ffffe4c7fa10
Receiving Data:last
```

LVQ_NO_BROWSE semantics:

LVQ uses a header for a key, if the key matches it replaces the message in-place in the queue except

- a.) if the message with the matching key has been acquired

In these two cases the message is placed into the queue in FIFO, if another message with the same key is received it will the 'un-accessed' message with the same key will be replaced

Note, in this case browsed messages are not invalidated, so updates can be missed.

An example

```
[localhost tests]$ ./lvqtest --mode create_lvq_no_browse
[localhost tests]$ ./lvqtest --mode write
Sending Data: key1=key1.0x7fffce5fb390
Sending Data: key2=key2.0x7fffce5fb390
Sending Data: key3=key3.0x7fffce5fb390
Sending Data: key1=key1.0x7fffce5fb390
Sending Data: last=last
[localhost tests]$ ./lvqtest --mode write
Sending Data: key1=key1.0x7fff346ae440
Sending Data: key2=key2.0x7fff346ae440
Sending Data: key3=key3.0x7fff346ae440
Sending Data: key1=key1.0x7fff346ae440
Sending Data: last=last
[localhost tests]$ ./lvqtest --mode browse
Receiving Data:key1.0x7fff346ae440
Receiving Data:key2.0x7fff346ae440
Receiving Data:key3.0x7fff346ae440
Receiving Data:last
[localhost tests]$ ./lvqtest --mode browse
Receiving Data:key1.0x7fff346ae440
Receiving Data:key2.0x7fff346ae440
Receiving Data:key3.0x7fff346ae440
Receiving Data:last
[localhost tests]$ ./lvqtest --mode write
Sending Data: key1=key1.0x7fff606583e0
Sending Data: key2=key2.0x7fff606583e0
Sending Data: key3=key3.0x7fff606583e0
Sending Data: key1=key1.0x7fff606583e0
Sending Data: last=last
[localhost tests]$ ./lvqtest --mode consume
Receiving Data:key1.0x7fff606583e0
Receiving Data:key2.0x7fff606583e0
Receiving Data:key3.0x7fff606583e0
Receiving Data:last
[localhost tests]$
```

Example source

```
/*
 *
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements. See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership. The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License. You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied. See the License for the
 * specific language governing permissions and limitations
 * under the License.
 *
 */

#include <qpid/client/AsyncSession.h>
#include <qpid/client/Connection.h>
#include <qpid/client/SubscriptionManager.h>
#include <qpid/client/Session.h>
#include <qpid/client/Message.h>
#include <qpid/client/MessageListener.h>
#include <qpid/client/QueueOptions.h>
```

```

#include <iostream>

using namespace qpid::client;
using namespace qpid::framing;
using namespace qpid::sys;
using namespace qpid;
using namespace std;

enum Mode { CREATE_LVQ, CREATE_LVQ_NO_BROWSE, WRITE, BROWSE, CONSUME};
const char* modeNames[] = { "create_lvq", "create_lvq_no_browse", "write", "browse", "consume" };

// istream/ostream ops so Options can read/display Mode.
istream& operator>>(istream& in, Mode& mode) {
    string s;
    in >> s;
    int i = find(modeNames, modeNames+5, s) - modeNames;
    if (i >= 5) throw Exception("Invalid mode: "+s);
    mode = Mode(i);
    return in;
}

ostream& operator<<(ostream& out, Mode mode) {
    return out << modeNames[mode];
}

struct Args : public qpid::Options,
               public qpid::client::ConnectionSettings
{
    bool help;
    Mode mode;

    Args() : qpid::Options("Simple latency test options"), help(false), mode(BROWSE)
    {
        using namespace qpid;
        addOptions()
            ("help", optValue(help), "Print this usage statement")
            ("broker,b", optValue(host, "HOST"), "Broker host to connect to")
            ("port,p", optValue(port, "PORT"), "Broker port to connect to")
            ("username", optValue(username, "USER"), "user name for broker log in.")
            ("password", optValue(password, "PASSWORD"), "password for broker log in.")
            ("mechanism", optValue(mechanism, "MECH"), "SASL mechanism to use when
authenticating.")
            ("tcp-nodelay", optValue(tcpNoDelay), "Turn on tcp-nodelay")
            ("mode", optValue(mode, "'see below'"), "Action mode."
            "\ncreate_lvq: create a new queue of type lvq.\n"
            "\ncreate_lvq_no_browse: create a new queue of type lvq with no lvq on browse.\n"
            "\nwrit": write a bunch of data & keys.\n"
            "\nbrowse: browse the queue.\n"
            "\nconsume: consume from the queue.\n");
    }
};

class Listener : public MessageListener
{
private:
    Session session;
    SubscriptionManager subscriptions;
    std::string queue;
    Message request;
    QueueOptions args;
public:
    Listener(Session& session);
    void setup(bool browse);
    void send(std::string kv);
    void received(Message& message);
    void browse();
    void consume();
};

Listener::Listener(Session& s) :
    session(s), subscriptions(s),
    queue("LVQtester")
{}

```

```

void Listener::setup(bool browse)
{
    // set queue mode
args.setOrdering(browse?LVQ_NO_BROWSE:LVQ);

    session.queueDeclare(arg::queue=queue, arg::exclusive=false, arg::autoDelete=false,
arg::arguments=args;

}

void Listener::browse()
{
    subscriptions.subscribe(*this, queue, SubscriptionSettings(FlowControl::unlimited(),
ACCEPT_MODE_NONE, ACQUIRE_MODE_NOT_ACQUIRED));
    subscriptions.run();
}

void Listener::consume()
{
    subscriptions.subscribe(*this, queue, SubscriptionSettings(FlowControl::unlimited(),
ACCEPT_MODE_NONE, ACQUIRE_MODE_PRE_ACQUIRED));
    subscriptions.run();
}

void Listener::send(std::string kv)
{
    request.getDeliveryProperties().setRoutingKey(queue);

    std::string key;
args.getLVQKey(key);
    request.getHeaders().setString(key, kv);

    std::ostringstream data;
data << kv;
    if (kv != "last") data << "." << hex << this;
    request.setData(data.str());

    cout << "Sending Data: " << kv << "=" << data.str() << std::endl;
async(session).messageTransfer(arg::content=request);

}

void Listener::received(Message& response)
{
    cout << "Receiving Data:" << response.getData() << std::endl;
/*      if (response.getData() == "last"){
        subscriptions.cancel(queue);
    }
*/
}

int main(int argc, char** argv)
{
    Args opts;
    opts.parse(argc, argv);

    if (opts.help) {
        std::cout << opts << std::endl;
        return 0;
    }

    Connection connection;
try {
    connection.open(opts);
    Session session = connection.newSession();
    Listener listener(session);

    switch (opts.mode)
    {
    case CONSUME:
        listener.consume();
        break;
    case BROWSE:

```

```
    listener/browse();
    break;
case CREATE_LVQ:
    listener/setup(false);
    break;
case CREATE_LVQ_NO_BROWSE:
    listener/setup(true);
    break;
case WRITE:
    listener/send("key1");
    listener/send("key2");
    listener/send("key3");
    listener/send("key1");
    listener/send("last");
    break;
}
connection/close();
return 0;
} catch(const std::exception& error) {
    std::cout << error.what() << std::endl;
}
return 1;
```

}

QMan - Qpid Management bridge

QMan : Qpid Management Bridge

QMan is a management bridge for Qpid. It allows external clients to manage and monitor one or more Qpid brokers.

Please note: All WS-DM related concerns have to be considered part of M5 release.

QMan exposes the broker management interfaces using Java Management Extensions (JMX) and / or OASIS Web Services Distributed Management (WS-DM). While the first one is supposed to be used by java based clients only the latter is an interoperable protocol that enables management clients to access and receive notifications of management-enabled resources using Web Services.

QMan can be easily integrated in your preexisting system in different ways :

- As a standalone application : in this case it runs as a server. More specifically it enables communication via RMI (for JMX) or via HTTP (for WS-DM); Note that when the WS-DM adapter is used the JMX interface is not exposed;
- As a deployable unit : it is also available as a standard Java web application (war); This is useful when there's a preexisting Application Server in your environment and you don't want start another additional server in order to run QMan.

User Documentation

With "User Documentation" we mean all information that you need to know in order to use QMan from a user perspective. Those information include :

Section	Description
Get me up and running	How to install & start QMan.
QMan Administration Console	QMan (WS-DM version only) Administration Console.
JMX Interface Specification	Describes each JMX interface exposed by QMan.
WS-DM Interface Specification	Describes each WS-DM interface exposed by QMan.
QMan Messages Catalogue	Informational / Debug / Error / Warning messages catalogue.

Technical Documentation

If you are interested in technical details about QMan and related technologies this is a good starting point. In general this section provides information about QMan design, interfaces, patterns and so on...

Section	Description
System overview	A short introduction about QMan deployment context.
Components view	Describes QMan components, their interactions and responsibilities.

Get me up and running

Get me up and running

- Get me up...
 - Prerequisites
 - Installation
- ...and running
 - JMX Bridge (Standalone)
 - WS-DM Bridge
 - WS-DM Bridge with a preexisting Application Server

Get me up...

This section describes how to install QMan.

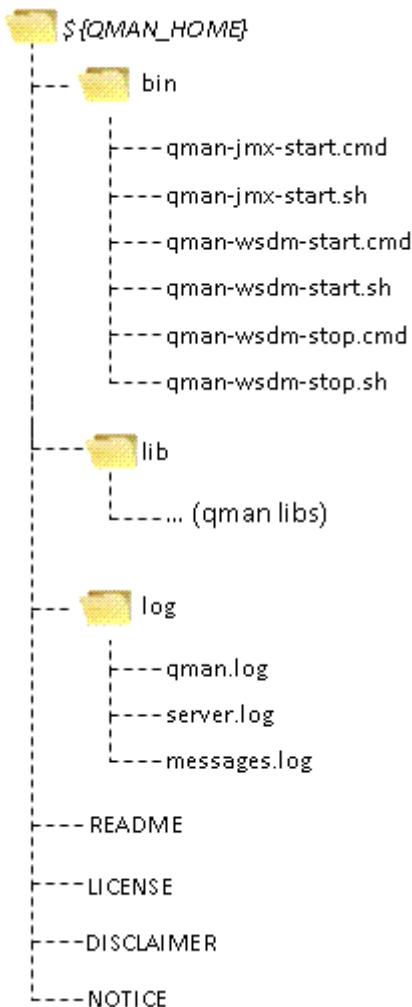
Prerequisites

QMan only runs with Java 5 or later. Java 6 is recommended for the best performance.

Installation

First of all, download the distribution from here and then unzip the archive into your chosen directory. Now you should have a directory

structure like this :



- bin : This contains the scripts needed to run QMan (see below for further details).
- lib : This contains jars needed to run QMan (core libraries and dependencies).
- log : This will contain the QMan log (see below for further details)

That's all! Please proceed to the next section in order to see how to run & configure QMan.

...and running

You can have QMan running in the following three different ways explained below; feel free to choose the way that best fits your needs...

JMX Bridge (Standalone)

This is a standalone RMI server that is able to expose Qpid broker management interface using JMX. As consequence of that, any Java based (management) client will be able, using JMX API, to remotely view the broker domain model, its properties, statistics and to invoke operations exposed by the management interface.

Configuration

QMan JMX Bridge needs two configuration files : qman-config.xml (optional) and log4j.xml. The first one is optional and must be used when if you wish to automatically connect to one or several brokers at startup; the log4j.xml configure logging.

qman-config.xml

Schema for this configuration file is very simple and for a better understanding let's start with an example :

```

<configuration>
  <brokers>
    <broker>
      <host>localhost</host>
      <port>5672</port>
      <virtual-host>test</virtual-host>
      <user>guest</user>
      <password>guest</password>
      <max-pool-capacity>4</max-pool-capacity>
      <initial-pool-capacity>0</initial-pool-capacity>
      <max-wait-timeout>-1</max-wait-timeout>
    </broker>
    <broker>
      <host>myhost</host>
      <port>5672</port>
      <virtual-host>test</virtual-host>
      <user>guest</user>
      <password>guest</password>
      <max-pool-capacity>4</max-pool-capacity>
      <initial-pool-capacity>0</initial-pool-capacity>
      <max-wait-timeout>-1</max-wait-timeout>
    </broker>
  </brokers>
</configuration>

```

- host : the hostname where the broker is running;
- port : the port where the broker is running;
- virtual-host : the virtual host as defined on the remote broker;
- user : the username used for establish connection;
- password : the password used for establish connection;
- max-pool-capacity : the maximum number of physical connections that the broker connection pool can contain;
- initial-pool-capacity : the number of physical connections to create when creating broker connection pool;
- max-wait-timeout : the maximum amount of time that a client will wait for obtaining a connection; a value of -1 means "Wait forever!".

The configuration in the example above specifies that QMan should connect to two brokers, one on localhost and one on myhost, both listening on port 5672. If you don't want connect QMan with any broker (at startup) simply leave this file empty (or without any <broker> declaration).

log4j.xml

For detailed information about how to configure log4j.xml please refer to <http://logging.apache.org/log4j/1.2/manual.html>

Run

To run QMan open up a shell / command prompt on 'bin' directory. After that execute ./qman-jmx-start.sh or qman-jmx.start.cmd (windows). You should see the following output on the /log/qman.log file :

```

2009-01-08 10:04:12,253 INFO  [QMan] <QMANT-000001> : Starting Q-Man...
2009-01-08 10:04:12,253 INFO  [QMan] <QMANT-000002> : Reading Q-Man configuration...
...
...
...
2009-01-08 10:04:12,847 INFO  [QMan] <QMANT-000023> : Q-Man service is now available on
MBeanServer.
2009-01-08 10:04:12,957 INFO  [QMan] <QMANT-000019> : Q-Man open for e-business.

```

Stop

Simply type "q" in the shell / command prompt from which QMan has been started.

Example : using JConsole as management client

The jconsole tool (<http://java.sun.com/javase/6/docs/technotes/guides/management/jconsole.html>) can acts as QMan client, allowing you to browse and interact with QMan exposed MBeans.

Remember, we are using RMI as communication protocol and therefore you need to add to JConsole classpath the QMan "client" classes. Those classes are contained on \$QMANT_HOME/lib/qpid-management-client-\$Version.jar. So at the end command line for running JConsole should look like this :

```
jconsole -J-Djava.class.path=$CLASSPATH:$JAVA_HOME/lib/jconsole.jar:$JAVA_HOME/lib/tools.jar
```

Where CLASSPATH contains the mentioned QMan jar and JAVA_HOME point on your JDK home.

When JConsole appears you can find QMan MBeans under the "MBeans" tab, Q-MAN domain.

WS-DM Bridge

QMan WS-DM Bridge is a HTTP server that acts as a WS-DM Adapter and enables broker remote management using this interoperable protocol. Briefly, let's say that broker management interface is exposed as a WS-Resources domain.

Configuration

QMan WS-DM Bridge needs three configuration files : qman-config.xml (optional), log4j.xml and jetty.xml. As the JMX version the qman-config.xml is optional while the other two are required. Specifically, jetty.xml is used for HTTP Server / Servlet Engine configuration and, unless you need an advanced configuration for web server, you can leave it as is.

qman-config.xml

For detailed information about this file please refer to the corresponding QMan JMX section before.

log4j.xml

For detailed information about how to configure log4j.xml please refer to <http://logging.apache.org/log4j/1.2/manual.html>

jetty.xml

QMan WS-DM uses Jetty as Web Server / Servlet Container. Detailed information on how to configure this module can be found at <http://docs.codehaus.org/display/JETTY/Configuring+Jetty>

Run

To run QMan open up a shell / command prompt on 'bin' directory. After that execute ./qman-wsdm-start.sh or qman-wsdm.start.cmd (windows).

You should see the following output on the /log/qman.log file :

```
2009-01-08 10:04:12,253 INFO  [QMan] <QMANT-000001> : Starting Q-Man...
2009-01-08 10:04:12,253 INFO  [QMan] <QMANT-000002> : Reading Q-Man configuration...
...
...
...
2009-01-08 10:04:12,847 INFO  [QMan] <QMANT-000023> : Q-Man service is now available on
MBeanServer.
2009-01-08 10:04:12,957 INFO  [QMan] <QMANT-000019> : Q-Man open for e-business.
2009-01-08 10:04:26,502 INFO  [WSDMAdapter] <QMANT-000026> : Initializing WS-DM Adapter
Environment...
2009-01-08 10:04:27,455 INFO  [WSDMAdapter] <QMANT-000027> : WS-DM Adapter ready for incoming
requests.
```

and this should be (moreless) the content of the /log/server.log:

```
2009-01-08 10:04:10,660 INFO  [log] Logging to org.slf4j.impl.Log4jLoggerAdapter(org.mortbay.log)
via org.mortbay.log.Slf4jLog
2009-01-08 10:04:10,832 INFO  [log] Extract jar:file:/.../qman.war!/ to
/.../Jetty_0_0_0_0_8080_qman.war__qman__j84idd/webapp
2009-01-08 10:04:11,957 WARN  [log] Unknown realm: default
2009-01-08 10:04:13,191 INFO  [log] Started SelectChannelConnector@0.0.0.0:8080
```

Stop

To run QMan open up a shell / command prompt on 'bin' directory and execute ./qman-wsdm-stop.sh or qman-wsdm-stop.cmd (windows). You should see the following output on the shell :

```
QMan WS-DM Adapter shut down successfully.
```

the following on the /log/qman.log:

```
2009-01-09 15:37:32,589 INFO  [QMan] <QMANT-000020> : Shutting down Q-Man...
2009-01-09 15:37:32,589 INFO  [QMan] <QMANT-000021> : Q-Man shut down.
```

and the following on /log/server.log:

```
2009-01-09 15:36:15,444 INFO  [log] Shutdown hook executing
2009-01-09 15:36:15,444 INFO  [log] Shutdown hook complete
```

WS-DM Bridge with a preexisting Application Server

QMan WS-DM Adapter is basically a JEE standard web application. You can find the archive under the \$QMAN_HOME/lib directory; it is the qman.war file.

So, generally speaking, you can get that archive and deploy it on your preferred Application Server that is J2EE 1.4 (or later) compliant. This kind of installation (deployment) is in general useful when there's already a Web Server in your environment.

The following is a list of servers where QMan can be deployed :

- JBoss 4.2.x;
- BEA Weblogic Server 9.x;
- BEA Weblogic Server 10;
- IBM WebSphere 7.x;
- Apache Tomcat 5.x;
- Jetty 6.x

Configuration

In this scenario we are working with a third-party middleware so basically the only thing that you can configure is the qman-config.xml (see above for details).

Note that the location of the configuration file must be passed as a JVM parameter from the start command line of the server:

```
java -Dqman-config=<complete path to qman-config.xml> ...
```

If you don't have permission to edit the server startup script please read the User Guide in order to see how to connect QMan with broker(s) at runtime.

JMX Interface Specification

JMX Interface Specification

This section contains QMan JMX Interface Specification.

Note that what is explained in this section refers to what is commonly addressed as "Instrumentation Layer".

That means the "Agent Layer" is not part of QMan and therefore it relies on standard JMX API.

So, strictly speaking, the described interface is not directly exposed but rather intermediated by the agent layer and specifically by the MBeanServer.

The following table describes the three entity types that are part of QMan management domain model.

Resource Type	Description	Multiplicity
QMan MBean	QMan is exposed itself as an MBean.	1
Object MBean	QMan JMX representation of a Qpid domain object exposed for management / monitoring.	0...*
Event MBean	QMan JMX representation of a Qpid domain event exposed for monitoring.	0...*

Event MBean

Event MBean

Events doesn't have operations because are representation of transient entities created on broker side.

- Description
- Object Name
- Attributes
- Operations
 - getAttribute
- Notifications

Description

QMan JMX representation of a Qpid event exposed for monitoring. Note that there will be an event MBean instance for each event produced on Qpid side.

This chapter refers to the abstract interface that object will have.

Object Name

Q-MAN:brokerId=<BROKER_ID>,type=Event,package=<PACKAGE_NAME>,class=<CLASS_NAME>,objectId=<OBJECT_ID>

where :

Name	Description	Example

Q-MAN	QMan management domain. This is a fixed value.	N.A.
BROKER_ID	Broker identifier. This is a UUID assigned to each connected broker. Basically it indicates the (broker) owner of this event.	5004341d-7f3e-444a-b240-7d48030599f9
PACKAGE_NAME	The package name. A package is a grouping of class definitions that are related to a single software component. The package concept is used to extend the management schema beyond just the QPID software components.	org.apache.qpid.broker
CLASS_NAME	The class name. A class is a type definition for a manageable event.	bind, subscribe

Attributes

An Event MBean is a JMX representation of something that happened on Qpid side. Its state is basically composed by the arguments

The JMX interface of an event MBean lets you retrieve attributes metadata using the standard JMX API. The following example is showing that.

Example : retrieving attributes metadata

```

public class Example
{
    public static void main(String[] args) throws Exception
    {
        MBeanServer server = ManagementFactory.getPlatformMBeanServer();

        // Suppose the following is an object name associated with an existing managed domain instance.
        ObjectName objectName = ...;

        MBeanInfo mbeanMetadata = server.getMBeanInfo(objectName);

        // List all attributes (metadata, not values)
        for (MBeanAttributeInfo attribute : mbeanMetadata.getAttributes())
        {
            System.out.println("Name : "+attribute.getName());
            System.out.println("Description : "+attribute.getDescription());
            System.out.println("Type : "+attribute.getType());
            System.out.println("Is Readable : "+attribute.isReadable());
            System.out.println("Is Writable : "+attribute.isWritable());
        }
    }
}

```

Operations

getAttribute

Operation Name	Description	Return Type
getAttribute	This operation allows client to retrieve value of an mbean attribute (argument).	java.lang.Object

Argument Name	Description	Type	Nullable	Note
objectName	This is the name of the target MBean.	javax.management.ObjectName	No	N.A.
attributeName	This is the name of the requested attribute.	java.lang.String	No	N.A.

Example

```
public class Example
{
    public static void main(String[] args) throws Exception
    {
        MBeanServer server = ManagementFactory.getPlatformMBeanServer();

        // Suppose that this is an object name corresponding to a valid managed domain instance.
        ObjectName objectName = ...;

        // Suppose the mbean has an attribute (a statistic in this case) PendingMessagesCount
        Long attributeValue = (Long) server.getAttribute(objectName, "PendingMessagesCount");

        System.out.println("Attribute Value : "+attributeValue);
    }
}
```

Notifications

N.A.

Object MBean

Object MBean

- Description
- Object Name
- Attributes
- Operations
 - `getAttribute`
 - `setAttribute`
 - `invoke`
- Notifications

Description

QMan JMX representation of a Qpid domain object exposed for management / monitoring. Note that there will be an object MBean for each management object on Qpid.

This chapter refers to the abstract interface that object will have.

Object Name

Q-MAN:`brokerId=<BROKER_ID>,type=Object,package=<PACKAGE_NAME>,class=<CLASS_NAME>,objectId=<OBJECT_ID>`

where :

Name	Description	Example
Q-MAN	QMan management domain. This is a fixed value.	N.A.
BROKER_ID	Broker identifier. This is a UUID assigned to each connected broker. Basically it indicates the (broker) owner of this object.	5004341d-7f3e-444a-b240-7d48030599f9
PACKAGE_NAME	The package name. A package is a grouping of class definitions that are related to a single software component. The package concept is used to extend the management schema beyond just the QPID software components.	org.apache.qpid.broker
CLASS_NAME	The class name. A class is a type definition for a manageable object.	queue, session, connection

Attributes

Object MBean attributes can be classified under two categories :

- **Properties** : typed members of object (that is, of its class definition) which represent a configurable attribute of the class. In general, properties don't change frequently or may not change at all;
- **Statistics** : typed members of object(that is, of its class definition) which represents an instrumentation attribute of the class. Statistics are always read-only in nature and tend to change rapidly.

The JMX interface of an object MBean lets you retrieve attributes metadata using the standard JMX API. The following example is showing that.

Example : retrieving attributes metadata

```

public class Example
{
    public static void main(String[] args) throws Exception
    {
        MBeanServer server = ManagementFactory.getPlatformMBeanServer();

        // Suppose the following is an object name associated with an existing managed domain instance.
        ObjectName objectName = ...;

        MBeanInfo mbeanMetadata = server.getMBeanInfo(objectName);

        // List all attributes (metadata, not values)
        for (MBeanAttributeInfo attribute : mbeanMetadata.getAttributes())
        {
            System.out.println("Name : "+attribute.getName());
            System.out.println("Description : "+attribute.getDescription());
            System.out.println("Type : "+attribute.getType());
            System.out.println("Is Readable : "+attribute.isReadable());
            System.out.println("Is Writable : "+attribute.isWritable());
        }
    }
}

```

Operations



Note that all operations that are part of the Object MBean interface are not exposed directly. According to JMX API specs, the invocation of those operations needs to be done using the MBeanServer Agent.

getAttribute

Operation Name	Description	Return Type
getAttribute	This operation allows client to retrieve value of an mbean attribute (property or statistic).	java.lang.Object

Argument Name	Description	Type	Nullable	Note
objectName	This is the name of the target MBean.	javax.management.ObjectName	No	N.A.
attributeName	This is the name of the requested attribute.	java.lang.String	No	N.A.

Example

```

public class Example
{
    public static void main(String[] args) throws Exception
    {
        MBeanServer server = ManagementFactory.getPlatformMBeanServer();

        // Suppose that this is an object name corresponding to a valid managed domain instance.
        ObjectName objectName = ...;

        // Suppose the mbean has an attribute (a statistic in this case) PendingMessagesCount
        Long attributeValue = (Long) server.getAttribute(objectName, "PendingMessagesCount");

        System.out.println("Attribute Value : "+attributeValue);
    }
}

```

setAttribute

Operation Name	Description	Return Type
setAttribute	This operation allows client to set the value of an mbean attribute (property). Note that it will be possible only if the attribute is writable. You can get that information on the corresponding attribute metadata.	void

Argument Name	Description	Type	Nullable	Note
objectName	This is the name of the target MBean.	javax.management.ObjectName	No	N.A.
attribute	This is a data transfer object representing the attribute with its new value.	javax.management.Attribute	Yes	N.A.

Example

```
public class Example
{
    public static void main(String[] args) throws Exception
    {
        MBeanServer server = ManagementFactory.getPlatformMBeanServer();

        // Suppose that this is an object name corresponding to a valid managed domain instance.
        ObjectName objectName = ...;

        // Suppose we want to set a value of 30000 for "MgmtPubInterval" attribute.
        Attribute attribute = new Attribute("MgmtPubInterval",new Long(3000));
        server.setAttribute(objectName, attribute);
    }
}
```

invoke

Operation Name	Description	Return Type
invoke	Invokes an operation on an object MBean	org.apache.qpid.management.domain.handler.impl.InvocationResult

Argument Name	Description	Type	Nullable	Note
objectName	The object name of the target object MBean	javax.management.ObjectName	No	N.A.
operationName	This is the operation to be invoked on the target MBean	java.lang.String	No	N.A.
parameters	These are the input parameters of the operation	java.lang.Object[]	No	N.A.
signature	The operation signature	java.lang.String []	No	N.A.

While mostly the interface follows the same rules of javax.management.MBeanServer.invoke() the only difference resides on return type. The mentioned JMX interface generally returns java.lang.Object. While this is the type that the management client see, the underlying object that is returned as result of an operation invocation on QMan is ALWAYS one of the following :

- org.apache.qpid.management.domain.handler.impl.InvocationResult : This is a simple data transfer object wrapping a java.util.Map<String, Object> that contains (optional) output parameters;
- org.apache.qpid.management.domain.services.MethodInvocationException : An exception containing a status text and a status code that indicate whether or not the method was successful and if not, what the error was.

Example

```
public class Example
{
    public static void main(String[] args) throws Exception
    {
        MBeanServer server = ManagementFactory.getPlatformMBeanServer();

        // Suppose that this is an object name corresponding to a valid managed domain instance.
        ObjectName objectName = new ObjectName("A:N=1");

        // Suppose the mbean has an operation
        // public int purge(int request)
        try
        {
            String outputParameterName = "result";

            String operationName = "purge";
            Object [] parameters = new Object[]{1235};
            String [] signature = new String[]{int.class.getName()};

            InvocationResult result = (InvocationResult)server.invoke(
                objectName,
                operationName,
                parameters,
                signature);

            // Output parameters map
            Map<String, Object> outputSection = result.getOutputSection();

            // Output parameter
            Integer outputParameter = (Integer) outputSection.get(outputParameterName);

            System.out.println("Output parameter : "+outputParameter);

        } catch (MBeanException exception)
        {
            Exception nested = exception.getTargetException();
            if (nested instanceof MethodInvocationException)
            {
                MethodInvocationException invocationException = (MethodInvocationException) nested;
                System.out.println("Status Code : "+invocationException.getReturnCode());
                System.out.println("Status Text : "+invocationException.getStatusText());
            }
        }
    }
}
```

Notifications

N.A.

QMan MBean

QMan MBean

- Description
- Object Name
- Attributes
- Operations
 - void addBroker
- Notifications

Description

QMan is exposed as MBean itself. That means its public interface will be available to any connected management client.

Object Name

Q-MAN:Name=QMan,Type=Service

Attributes

N.A.

Operations

void addBroker

Operation Name	Description	Return Type
addBroker	Connects QMan with a broker using the given connection data.	void

Argument Name	Description	Type	Nullable	Note
host	The IP address or DNS name where Qpid Broker is running.	java.lang.String	No	N.A.
port	The port number where Qpid broker is running.	int	No	N.A.
username	The username used for establishing connection with Qpid broker	java.lang.String	No	N.A.
password	The password used for establishing connection with Qpid broker	java.lang.String	No	N.A.
virtualHost	The virtual host name	java.lang.String	No	N.A.
initialPoolCapacity	The number of physical connections (between 0 and a positive 32-bit integer) to create when creating the (broker) connection pool.	int	No	N.A.
maxPoolCapacity	The maximum number of physical database connections (between 0 and a positive 32-bit integer) that the (Qpid) connection pool can maintain.	int	No	N.A.
maxWaitTimeout	The maximum amount of time to wait for an idle connection	long	No	A value of -1 means "Wait forever"

Example

```
import java.lang.management.ManagementFactory;

import javax.management.MBeanServer;
import javax.management.ObjectName;

public class Example
{
    public static void main(String[] args) throws Exception
    {
        MBeanServer server = ManagementFactory.getPlatformMBeanServer();

//    ObjectName objectName = new ObjectName("Q-MAN:Name=QMan,Type=Service");
ObjectName objectName = Names.QMAN_OBJECT_NAME;

        String host = "qpidd.host.com";
        int port = 2005;

        String username ="qpidd_username";
        String password ="qpidd_password";

        String virtualHost = "qpidd_virtualhost";

        int initialPoolCapacity = 3; // Open 3 connections immediately.
int maxPoolCapacity = 4; // another on-demand additional connection.
int maxWaitTimeout = 2000;

        server.invoke(
            objectName,
            "addBroker",
            new Object []{
                host,
                port,
                username,
                password,
                virtualHost,
                initialPoolCapacity,
                maxPoolCapacity,
                maxWaitTimeout},
            new String[] {
                String.class.getName(),
                int.class.getName(),
                String.class.getName(),
                String.class.getName(),
                String.class.getName(),
                int.class.getName(),
                int.class.getName(),
                long.class.getName()
            });
    }
}
```

Notifications

Type	Class	Description
org.apache.qpid.management.lifecycle.entity.schema.requested	org.apache.qpid.management.jmx.EntityLifecycleNotification	A schema request for QMan entity has been sent.
org.apache.qpid.management.lifecycle.entity.schema.injected	org.apache.qpid.management.jmx.EntityLifecycleNotification	A schema has been injected on QMan entity
org.apache.qpid.management.lifecycle.error.schema	org.apache.qpid.management.jmx.EntityLifecycleNotification	Qman has received a malformed schema.

qman.lifecycle.entity.instance.created	org.apache.qpid.management.jmx.EntityLifecycleNotification	A new instance (event or object) has been created on QMan management domain.
qman.lifecycle.entity.instance.removed	org.apache.qpid.management.jmx.EntityLifecycleNotification	An object instance has been removed from QMan management domain.



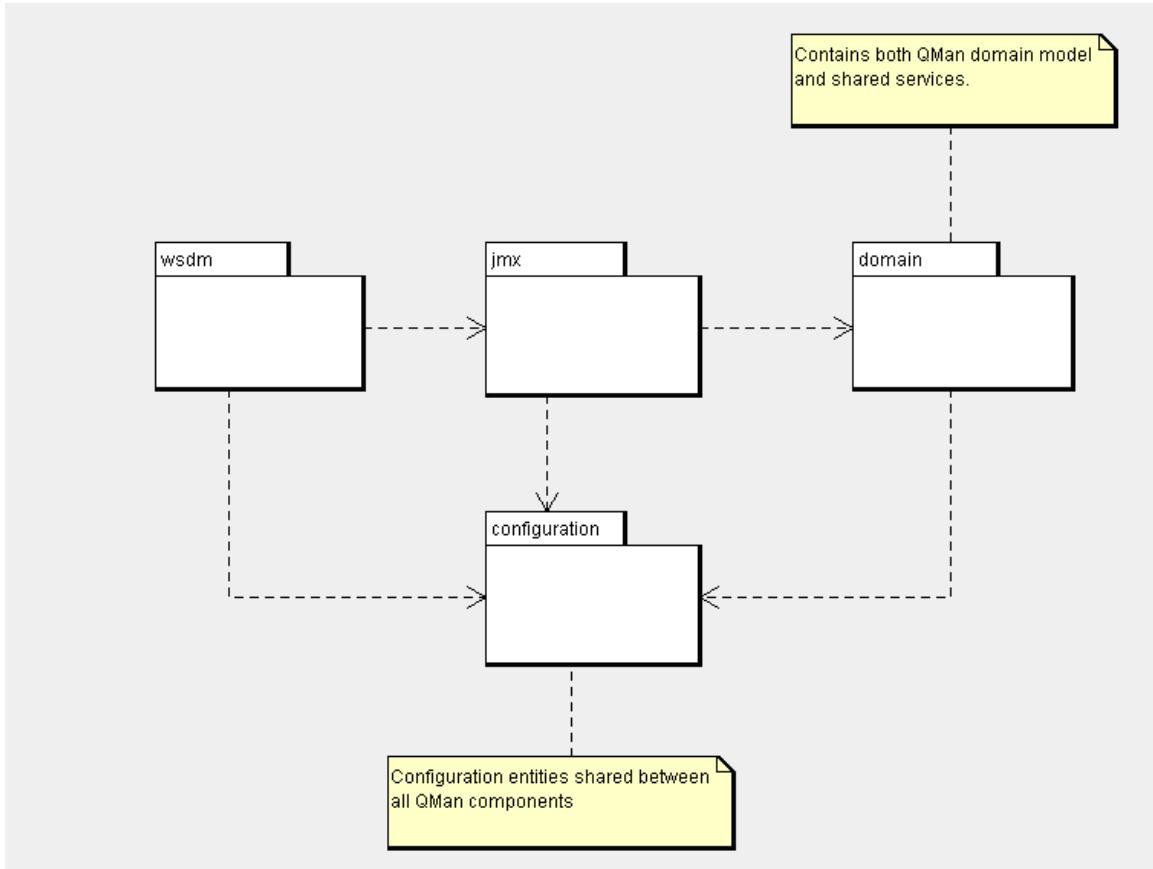
Consider that notifications are sent asynchronously so QMan is not waiting for completion of receiver task.

QMan Components View

Components View

- Package configuration
 - Configurator
 - Configuration
 - Parser
 - WorkerManagerConfigurationData
 - BrokerConfigurationData
 - AccessModes
 - Message Handlers
- Package domain
- Package jmx
- Package wsdm
 - QManLifecycleManager
 - WsDmAdapter
 - QMan
 - QManWsResource
 - QManWsAdapter
 - SubscriptionManager
 - MBeanWSResource
 - Other WS interfaces

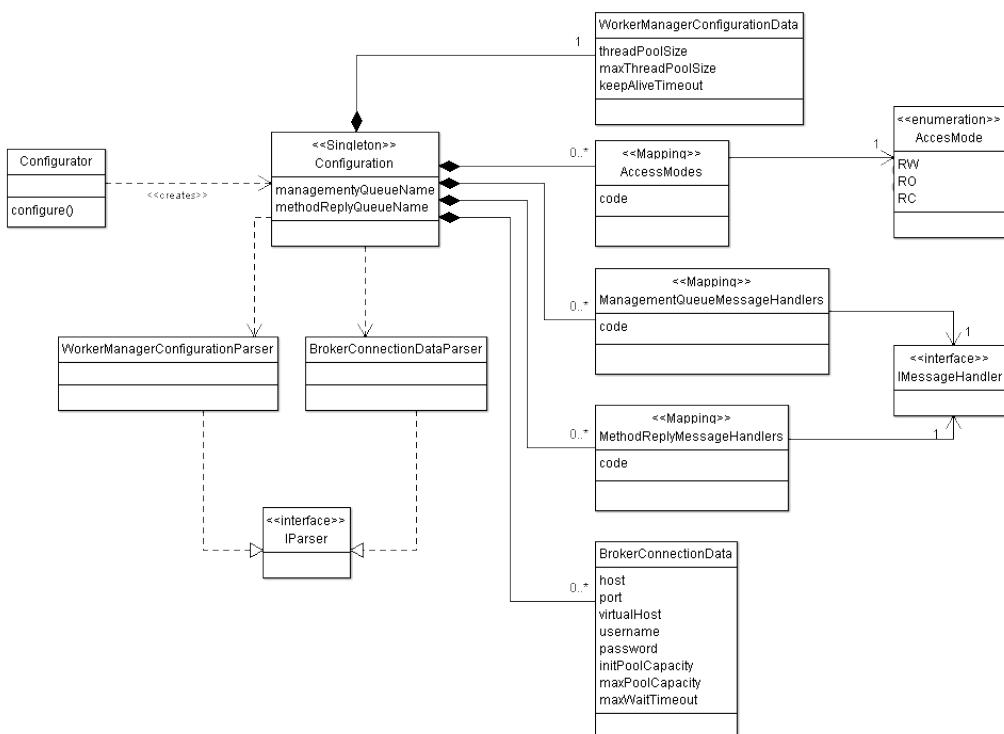
At package level, QMan is composed by 4 core packages illustrated in the picture below :



Package configuration

This package contains all configuration entities / items shared between all QMan components. Those includes both configuration items that are directly under the user control and not-public items used internally to configure (sub) components.

As you can see, the configuration package is a central grouping of components that serves the remaining three modules.



Configurator

When QMan starts, this component is responsible to build the configuration for that instance. As you can see it uses several built-in builders in order to create and populate the configuration instance.

Configuration

Encapsulates a configuration for a QMan instance. Note that this is a singleton because it must be accessed from everywhere inside QMan sub-modules.

Parser

Active participants of the configuration build process. Basically they are responsible to parse a specific section of the (optional) qman-configuration file that is given at QMan startup. At the moment we have two implementation of this interface :

- **WorkerManagerConfigurationParser** : parses the configuration data of the internal worker manager; It creates a WorkerManagerConfigurationData instance.
- **BrokerConnectionDataParser** : parses the connection data of declared brokers; For each configured broker a corresponding BrokerConnectionData instance is created.

WorkerManagerConfigurationData

A value object encapsulating configuration data for the work manager.

BrokerConfigurationData

A value object encapsulating connection data of a remote broker.

AccessModes

A map associating a code with an access mode. At the moment we have three access modes :

- **RO** : Read only;
- **RW** : Read / Write;
- **RC** : Read create;

Message Handlers

Each time a message is received from a remote broker there will be a specific message handler that is responsible for processing that message.

On top of that, message handlers mapping associates an operation code (a character) with a message handler instance.

As you can see, there are two distinct collections of mappings. The first one contains message handlers associated with management queue, while the second one is referred to message handlers associated with method reply queue.

Package domain

Package jmx

Contains all services that are part of QMan JMX adapter. A special note should be done for this package because even from the picture is not clear, the JMX core could be used independently from the WSDM adapter.

That means this package contains all services and interfaces needed for expose a Qpid management domain model using JMX.

That fits a management scenario where there's a java agent layer that wants to monitor / instrument / manage one or more remote brokers. Each managed entity will have its own JMX representation as an MBean with common and specific features (depending on the resource type).

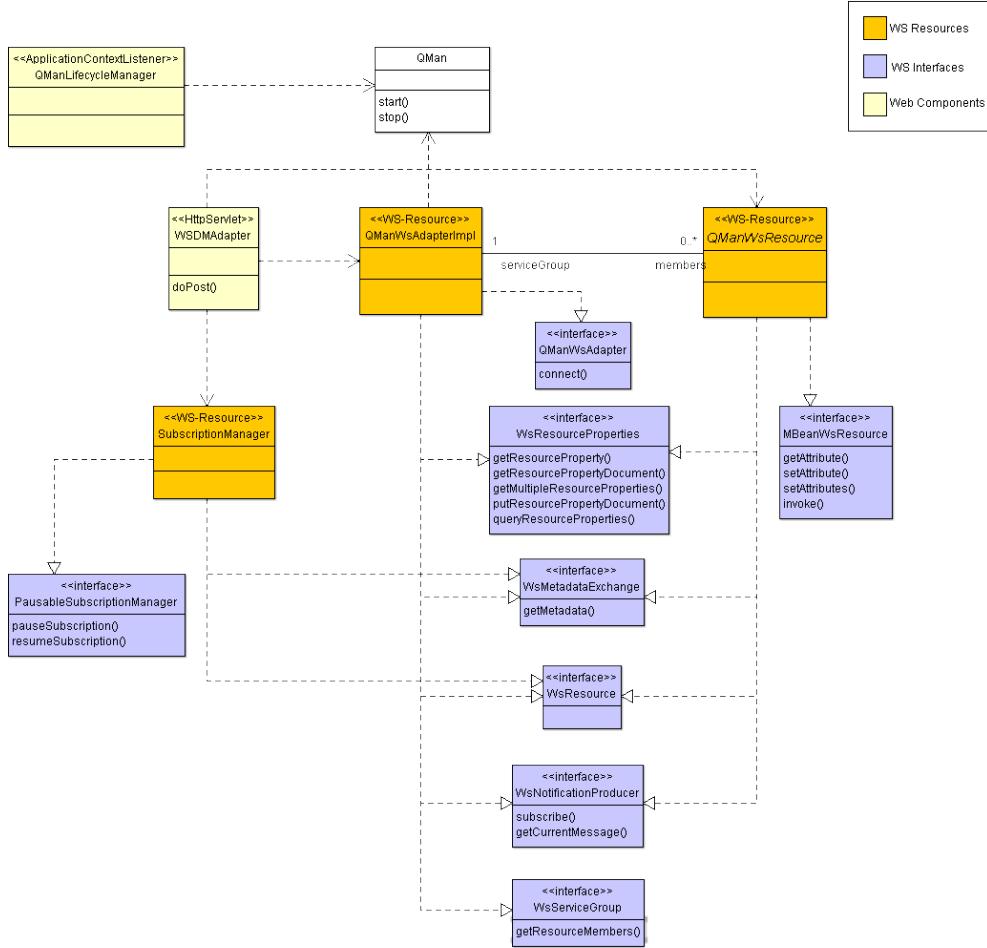
JMX interfaces and entities are detailed [here](#)

Package wsdm

Components that belong to this package enable QMan interfaces to be exposed using WS-DM / HTTP. As part of that, this package contains

- standard JEE Web components (Servlets and Context Listeners).
- WS-DM specific components

Note that this is an additional layer over the previous JMX core so the basically the same considerations apply management resources (in this case we call them WS-Resources).



QManLifecycleManager

Simply speaking, we could say that the whole WSDM Layer acts as a facade of the JMX Adapter, so it should be able to control the lifecycle of a that adapter. This component is a web component that (as the name suggests) provides a lifecycle management (startup & shutdown) of a JMX adapter instance.

WsDmAdapter

An HTTP Servlet that listens for incoming WS-DM / HTTP requests and dispatches those requests to the appropriate handler.

QMan

A front controller of the JMX instrumentation layer exposed itself for management (as an MBean).

QManWsResource

QMan object representation of a WS-Resource. A WS-Resource is a composition of a management resource and a Web service through which the resource can be accessed.

QManWsAdapter

A static WS-Resource that acts as a controller / facade of QMan WS-DM management domain model. Basically it provides the following features :

- WS-Resource creation : When a new resource is built on JMX layer, it builds the corresponding WS-Resource representation (WS-Resource instance, WSDL, RMD and capabilities)
- WS-Resource deletion : When a resource is deleted on JMX layer (i.e. a connection that has been closed, a session that has been destroyed), it deletes the corresponding WS-Resource
- Notifications : Acting as a notification producer, each time a WS-Resource is created / destroyed a dedicated message is published on a lifecycle topic.

SubscriptionManager

A WS-Resource that enables QMan notifications. It provides operations that allow a requestor to query and manipulate subscription

resources that it manages. For example it is possible (on requestor side) to pause and resume a subscription.

MBeanWSResource

This is the interface that all QMan WS-Resources have. Although a concrete implementation of this interface is built at runtime, it indicates that the resources will have a common set of features (attribute retrieval, operation invocation).

Other WS interfaces

The other interfaces on the diagram simply enumerates all the WS-DM interfaces that will be implemented by the QMan WS-Resources. More information about those interfaces are found [here](#).

QMan Messages Catalogue

QMan Messages Catalogue

Each message produced by Qman has the following format :

<DATE> <PRIORITY> <MODULE> <MESSAGE-ID> : <MESSAGE>

Where :

- <DATE> : The date and time at which the message was added;
- <PRIORITY> : The priority level. Could be one of the following : DEBUG, WARN, INFO, ERROR, FATAL;
- <MODULE> : The component which generated the log;
- <MESSAGE-ID> : A message identifier. Each message has a unique id. It has the following format <QMAN-XXXXXXX>;
- <MESSAGE> : The log message. Data recorded in this field should be ignored by analysis tools because it could change.

For example :

```
2009-02-12 15:20:44,971 INFO QMan <QMAN-000019> : Q-Man open for e-business.
```

Note that the log format described above is not fixed but could be configured using log4j. Obviously what mentioned above is referred to the default configuration shipped with QMan distribution.

The following is a list of all QMan messages divided by priority level.

Messages	Description
QMAN-000001 - QMAN-000029	Informational messages
QMAN-100001 - QMAN-100026	Error messages
QMAN-200001 - QMAN-200041	Debug messages
QMAN-300001 - QMAN-300004	Warning messages



Be Careful

QMan is under development so the list above could change easily...

QMan System Overview

System Overview

- [Introduction](#)

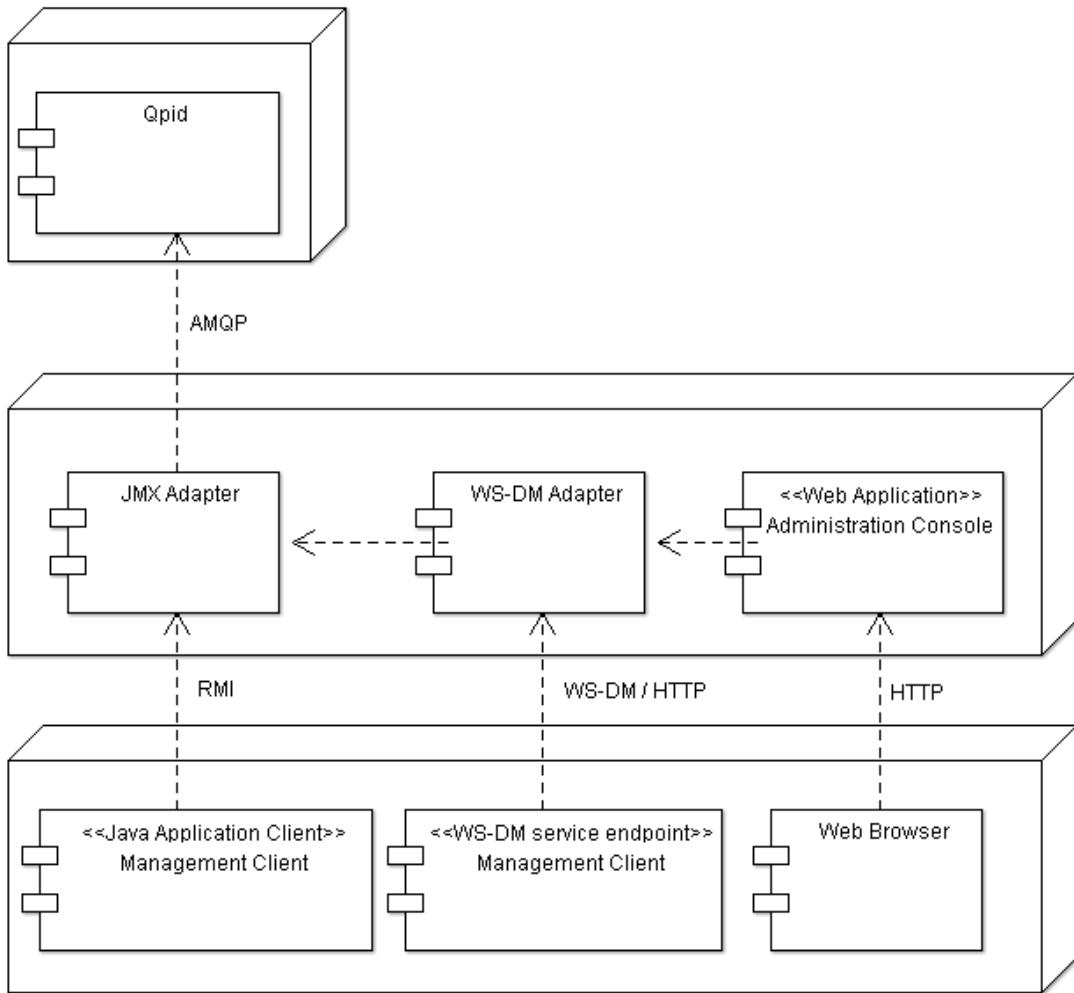
Introduction

QMan is a bridge that enables remote management of one or more brokers.

Depending on your needs, remote management can be exposed using JMX or WS-DM. The difference between two approaches is that in the first one only a Java client is supported, while WS-DM is language independent and therefore the corresponding management entity (client) should only able to speak WS-DM dialect.

As consequence of that we will divide the following documentation under two perspectives.

In the picture below you can have a big picture about the working environment where QMan runs :



As you can see QMan supports three different clients :

- a Java (Application) Client which is using Java Management Extensions (JMX) API for interaction; Note that this could be a standalone application client or a JEE application (Web & Business components);
- a WS-DM enabled management client which could be an application / middleware / command line tool that is able to communicate with WS-DM dialect;
- the third client (Web Browser) is not really a management client but it refers to administration console of QMan WS-DM. This is part of QMan Web Application and allows administration / management of QMan itself.

QMan User Guide

QMan Admin Console

Created by [Andrea Gazzarini](#)
 On Thu Feb 12 09:11:48 CET 2009
 Using [TimTam](#)

QMan Debug Messages

Debug messages

QMAM-200001

- **Message** : New incoming message has been received. Message content is *<content>*.
- **Description** : A new message is received. Its content is *<content>*.
- **Cause** : This is a debug message so there's not a cause
- **Action** : No action is required.

QMAM-200002

- **Message** : *<opcode>* opcode is associated to handler *<handler>*.
- **Description** : Informs that operation code *<opcode>* has been associated with the handler *<handler>*.

- **Cause** : This is a debug message so there's not a cause
- **Action** : No action is required.

QMAN-200003

- **Message** : Incoming message with `<opcode>` as opcode will be forwarded to `<handler>` for processing.
- **Description** : This is a runtime information regarding the association between an operation code and the associated handler.
- **Cause** : This is a debug message so there's not a cause
- **Action** : No action is required.

QMAN-200004

- **Message** : Management queue name : `<name>`.
- **Description** : Informs about the name of the management queue that will be declared on the broker.
- **Cause** : This is a debug message so there's not a cause
- **Action** : No action is required.

QMAN-200005

- **Message** : Method-Reply queue name : `%s`.
- **Description** : Informs about the name of the method-reply queue that will be declared on the broker.
- **Cause** : This is a debug message so there's not a cause
- **Action** : No action is required.

QMAN-200006

- **Message** : Connection `<id>` returned to the pool.
- **Description** : Broker connection lifecycle debug message. A connection has been released.
- **Cause** : This is a debug message so there's not a cause
- **Action** : No action is required.

QMAN-200007

- **Message** : Test connection on reserve. Is valid? `<is_valid>`
- **Description** : Broker connection lifecycle debug message. A connection has been tested.
- **Cause** : This is a debug message so there's not a cause
- **Action** : No action is required.

QMAN-200008

- **Message** : Connection has been destroyed.
- **Description** : Broker connection lifecycle debug message. A connection has been destroyed.
- **Cause** : This is a debug message so there's not a cause
- **Action** : No action is required.

QMAN-200009

- **Message** : Unable to destroy a connection object.
- **Description** : Broker connection lifecycle debug message. There was been a failure while destroying a connection.
- **Cause** : This is a debug message so there's not a cause
- **Action** : No action is required.

QMAN-200010

- **Message** : Event instance `<broker_id>:<package_name>:<object_id>` successfully registered with MBean Server with name `_<event_name>`.
- **Description** : A new event instance has been registered on QMan domain with the name `<event_name>`.
- **Cause** : This is a debug message so there's not a cause
- **Action** : No action is required.

QMAN-200011

- **Message** :
- **Description** :
- **Cause** : This is a debug message so there's not a cause
- **Action** : No action is required.

QMAN-200012

- **Message** :
- **Description** :
- **Cause** : This is a debug message so there's not a cause
- **Action** : No action is required.

QMAN-200013

- **Message :**
- **Description :**
- **Cause :** This is a debug message so there's not a cause
- **Action :** No action is required.

QMAM-200014

- **Message :**
- **Description :**
- **Cause :** This is a debug message so there's not a cause
- **Action :** No action is required.

QMAM-200015

- **Message :**
- **Description :**
- **Cause :** This is a debug message so there's not a cause
- **Action :** No action is required.

QMAM-200016

- **Message :**
- **Description :**
- **Cause :** This is a debug message so there's not a cause
- **Action :** No action is required.

QMAM-200017

- **Message :**
- **Description :**
- **Cause :** This is a debug message so there's not a cause
- **Action :** No action is required.

QMAM-200018

- **Message :**
- **Description :**
- **Cause :** This is a debug message so there's not a cause
- **Action :** No action is required.

QMAM-200019

- **Message :**
- **Description :**
- **Cause :** This is a debug message so there's not a cause
- **Action :** No action is required.

QMAM-200020

- **Message :**
- **Description :**
- **Cause :** This is a debug message so there's not a cause
- **Action :** No action is required.

QMan Error Messages

Error Messages

QMAM-100001

- **Message :** Message processing failure : incoming message contains a bad magic number <bad_magic_number> and therefore will be discarded.
- **Description:** Incoming Qpid message contains a bad magic number.
- **Cause :** The incoming message contains a wrong magic number (it differs from the expected value).
- **Action:** QMan has nothing to do with this problem; Check on broker side in order to see if some protocol change occurred.

QMAM-100002

- **Message :** Message I/O failure : unable to read byte message content and therefore it will be discarded.
- **Description:** QMan is not able to read the incoming message stream.
- **Cause :** Management messages have a predefined format and adhere to specific rules so cause for this problem can be a malformed message or a general I/O failure.
- **Action:** See QMan logs for a better understanding of the exact cause and check if there are network issues and if something

changed on broker side (especially protocol change due to an upgrade).

QMAN-100003

- **Message :** Message processing failure : unknown exception; see logs for more details.
- **Description:** QMan is not able to read and /or process the incoming message.
- **Cause :** Not well specified cause : as suggested you should see the log for further details.
- **Action:** See QMan logs for a better understanding of the exact cause.

QMAN-100005

- **Message :** Q-Man was unable to process the schema response message.
- **Description:** An incoming schema message response for a class cannot be processed.
- **Cause :** This could only happen if a protocol change occurred in the schema messages.
- **Action:** See QMan logs for a better understanding of the exact cause and check on broker side in order to see if some protocol change occurred.

QMAN-100006

- **Message :** Q-Man was unable to process the schema response message.
- **Description:** An incoming schema message response for an event cannot be processed.
- **Cause :** This could only happen if a protocol change occurred in the schema messages.
- **Action:** See QMan logs for a better understanding of the exact cause and heck on broker side in order to see if some protocol change occurred.

QMAN-100007

- **Message :** Unable to connect with broker located on <broker_id>. This broker will be ignored.
- **Description:** QMan has been requested to connect with a broker but the connection cannot be established.
- **Cause :** Newtork issues or simply the broker is not running.
- **Action:** See QMan logs for a better understanding of the exact cause and check if the broker is running.

QMAN-100008

- **Message :** Management Message Handler configured for opcode %s is not available and therefore will be discarded.
- **Description:** QMan has been requested to connect with a broker but the connection cannot be established.
- **Cause :** Newtork issues or simply the broker is not running.
- **Action:** See QMan logs for a better understanding of the exact cause and check if the broker is running.

QMAN-100010

- **Message :** An exception occurred while storing the result of a method invocation. Sequence number was <sequence_number>.
- **Description:** As consequence of a method invocation, a response is gathered from the broker. This message indicates a failure that occurred while handling that result;
- **Cause :** Method invocation response message must adhere to a specific (protocol defined) format. Probably the message couldn't be parsed and / or handled because it is malformed.
- **Action:** See QMan logs for a better understanding of the exact cause.

QMAN-100011

- **Message :** Unknwon class kind : <class_kind>
- **Description:** Incoming schema messages contains an unknown class kind.
- **Cause :** Allowed class kind at the moment are Object and Event. Probably the message was malformed or a protocol change occurred on the broker side.
- **Action:** See QMan logs for a better understanding of the exact cause and check on the broker side in order to see if some protocol change occurred.

QMAN-100012

- **Message :** Q-Man was unable to process the schema response message.
- **Description:** Not well error occurs while parsing the incoming schema message.
- **Cause :** It's hard to determine. Check the QMan logs.
- **Action:** See QMan logs for a better understanding of the exact cause and check on the broker side in order to see if some protocol change occurred.

QMAN-100013

- **Message :** Unable to unregister object instance <object_name>.
- **Description:** A new event or object instance deletion has been notified; The corresponding JMX entity unregistration failed.
- **Cause :** This is probably due to the fact that the requested instance no longer exists on QMan and therefore was previously unregistered.
- **Action:** See QMan logs in order to see the name of the object that caused the failure. After that, on the same log, have a look all lifecycle events related with this entity. This should give you a clear idea of what happened.

QMAN-100014

- **Message** : Unable to decode value for attribute <attribute_name>.
- **Description**: A content indication message arrived but QMan is not able to decode correctly its data.
- **Cause** : This message appears under two circumstances : a malformed / corrupted message or a protocol change occurred on the broker side.
- **Action**: See QMan logs in order to see the exact cause of the problem and after that ensure that no protocol changes occurred on broker side.

QMAN-100015

- **Message** : Unable to send a schema request schema for <package_name>.<class_name>
- **Description**: QMan is not able to send a Schema request to Qpid.
- **Cause** : Probably there's some problem at I/O level (i.e. network).
- **Action**: Ensure that connection with broker is established and working.

QMAN-100016

- **Message** : Unable to decode value for <broker_id>:<package_name>:<class_name>.
- **Description**: Basically we are talking about the same thing reported on QMAN-100014. The difference is that here we have the FQN of the entity.
- **Cause** : see QMAN-100014.
- **Action**: see QMAN-100014.

QMAN-100017

- **Message** : Cannot connect to broker <broker_id> on <connection_data>
- **Description**: QMan is not able to establish a connection with broker <broker_id>, whose connection data is <connection_data>.
- **Cause** : Network issue or the target broker is not running.
- **Action**: Check network and target broker.

QMAN-100018

- **Message** : Q-Man was unable to startup correctly : see logs for further details.
- **Description**: QMan startup procedure fails.
- **Cause** : There could be many reasons for that. Probably the cause is an error in the configuration.
- **Action**: Check QMan configuration files and of course, have a look at logs...there should be written the exact cause of the problem.

QMAN-100019

- **Message** : Unexpected exception occurred on WSDM adapter layer : probably request or response was malformed.
- **Description**: Soap Request or response is malformed. Most probably issue occurs in the request (because response is automatically generated by QMan)
- **Cause** : The WS-DM adapter deals with Soap messages, which must be valid XML documents. If one of them (request or response) is malformed you get this error.
- **Action**: Using QMan administration console enable a fine level for WS-DM adapter log. You should see all XML messages that are exchanged during conversations.

QMAN-100020

- **Message** : WS-Action not supported.
- **Description**: An action request has been made to a QMan but the requested action is not supported by the target WS-Resource.
- **Cause** : Each managed WS-Resource has capabilities that include several actions (it depends by the resource interface). Probably the request has been sent to the wrong resource.
- **Action**: Using QMan administration console inspect the requested resource in order to see all the supported actions. Check the incoming request message in order to see if it is correct.

QMAN-100021

- **Message** : Unable to build RDM for resource <resource_id>.
- **Description**: Resource Metadata Descriptor is an artifact that is built when a WS-Resource is created. The RDM for the reported WS-Resource cannot be created.
- **Cause** : RDM is basically a descriptor reporting the metadata for a WS-Resource so probably the problem is in the structure of the resource (properties, operations). It is also possible that a resource interface contains a datatype that is not supported.
- **Action**: Using QMan administration console enable a fine level for WS-DM adapter log. You should see detailed information about the lifecycle of each managed resources including generated artifacts.

QMAN-100023

- **Message** : Unable to build WS artifacts.
- **Description**: When a WS-Resource is created, its WSDL and RDM is generated too. The engine wasn't able to build those artifacts.
- **Cause** : This is, as QMAN-100021, a problem with the interface / definition of the resource that needs to be created.
- **Action**: Using QMan administration console enable a fine level for WS-DM adapter log. You should see detailed information about the lifecycle of each managed resources including generated artifacts.

QMAN-100024

- **Message** : Unable to instantiate generated capability class for <resource_id>.
- **Description**: When a WS-Resource is created, QMan is building several classes that represent its capabilities. This error indicates that at least one capability class failed to initialize.
- **Cause** : This is, as QMAN-100021, a problem with the interface / definition of the resource that needs to be created.
- **Action**: Using QMan administration console enable a fine level for WS-DM adapter log. You should see detailed information about the lifecycle of each managed resources including generated artifacts.

QMAN-100024

- **Message** : Resource manager raised an exception while creating capability for <resource_id>.
- **Description**: A resource manager is an internal component that handles resources lifecycle including creation and destruction. This error occurs when the resource cannot be created.
- **Cause** : This is, as QMAN-100021, a problem with the interface / definition of the resource that needs to be created.
- **Action**: Using QMan administration console enable a fine level for WS-DM adapter log. You should see detailed information about the lifecycle of each managed resources including generated artifacts.

QMAN-100024

- **Message** : Exception occurred while replacing the placeholder soap address with resource actual location.
- **Description**: Each WS-Resource has a WSDL that represents itself as a Web service. Service description is built starting from a template WSDL that contains a fake address replaced with the actual location when the resource is built.
- **Cause** : This is an internal QMan error and should never appear because is due to a malformed WSDL template.
- **Action**: Using QMan administration console enable a fine level for WS-DM adapter log. You should see detailed information about the lifecycle of each managed resources including generated artifacts.

QMAN-100027

- **Message** : Shutdown failure while destroying resource <resource_id>.
- **Description**: QMan was not able to destroy the resource <resource_id>.
- **Cause** : Most probably the resource was busy when the shutdown has been requested.
- **Action**: Using QMan administration console enable a fine level for WS-DM adapter log.

QMAN-100029

- **Message** : Unable to define URI for QMan resources using <uri>. It violates RFC 2396
- **Description**: Each WS-Resource is associated with an URI generated a runtime and based on the running environment (host, port).
- **Cause** : Most probably some information is missing and therefore QMan is not able to create a valid URI.
- **Action**: See installation instructions and ensure that all needed environment properties are correctly set.

QMAN-100030

- **Message** : QMan JMX core Unexpected failure while starting up.
- **Description**: Startup failure has occurred on JMX core.
- **Cause** : This indicates a configuration error that occurred at startup.
- **Action**: Check QMan configuration files.

QMAN-100031

- **Message** : Bad request has been received on this WS-Resource : Initialization is not possible because the resource has already been initialized.
- **Description**: An initialization request has been issued to an already initialized WS-Resource.
- **Cause** : It is not possible to initialize a WS-Resource twice. Something was wrong on WS-DM Adapter.
- **Action**: Using QMan administration console enable a fine level for WS-DM adapter log. You should see detailed information about the lifecycle of each managed resources including generated artifacts.

QMAN-100032

- **Message** : Bad request has been received on this WS-Resource : Shutdown is not possible because the resource hasn't yet been initialized.
- **Description**: A shutdown request has been issued to a resource that has not been initialized.
- **Cause** : It is not possible to shutdown a WS-Resource that isn't initialized. Something was wrong on WS-DM Adapter.
- **Action**: Using QMan administration console enable a fine level for WS-DM adapter log. You should see detailed information about the lifecycle of each managed resources including generated artifacts. It is also possible that a wrong deletion message (content indication message) arrived from Qpid broker.

QMAN-100033

- **Message** : Bad request has been received on this WS-Resource : Shutdown is not possible because the resource has already been shutdown.
- **Description**: A shutdown request has been issued to an already shutdown WS-Resource.
- **Cause** : It is not possible to shutdown a WS-Resource twice. Something was wrong on WS-DM Adapter.
- **Action**: Using QMan administration console enable a fine level for WS-DM adapter log. You should see detailed information about the lifecycle of each managed resources including generated artifacts. It is also possible that a wrong deletion message (content indication message) arrived from Qpid broker.

QMAN-100034

- **Message** : Unable to get via XPath the schema section in WSDL.
- **Description**: QMan is not able to deal with WSDL template for a specific WS-Resource.
- **Cause** : This is an internal QMan error and should never appear because is due to a malformed WSDL template.
- **Action**: Using QMan administration console enable a fine level for WS-DM adapter log. You should see detailed information about the lifecycle of each managed resources including generated artifacts.

QMan Informational Messages

Informational Messages

QMAM-000001

- **Message** : Starting QMan...
- **Description**: Application is starting.
- **Cause** : This is an informational message only.
- **Action**: No action required.

QMAM-000002

- **Message** : Reading Q-Man configuration...
- **Description**: Application is reading configuration. This doesn't include the (eventual) qman-config.xml configuration file supplied from command line.
- **Cause** : This is an informational message only.
- **Action**: No action required.

QMAM-000003

- **Message** : Creating management client(s)...
- **Description**: Each time a broker is defined and connected on QMan a dedicated (internal) management client module is created;
- **Cause** : This is an informational message only.
- **Action**: No action required.

QMAM-000004

- **Message** : Management client for broker <broker_id> successfully connected.
- **Description**: A management client for broker #<broker_id> has been created and successfully connected.
- **Cause** : This is an informational message only.
- **Action**: No action required.

QMAM-000005

- **Message** : Type mapping : code = <code> associated to <AMQP type> (validator class is <validation_class>)
- **Description**: New mapping between a Java type and an AMQP type has been created. This type will be validated using
- **Cause** : This is an informational message only.
- **Action**: No action required.

QMAM-000006

- **Message** : Access Mode mapping : code = <code> associated to <access_mode>
- **Description**: New access mode mapping has been created. The code <code> has been associated with access mode <access_mode>.
- **Cause** : This is an informational message only.
- **Action**: No action required.

QMAM-000007

- **Message** : Management Queue Message Handler Mapping : opcode = <code> associated with <class>
- **Description**: New Management Queue Message Handler instance (class name is <class>) has been created and associated with code <code>.
- **Cause** : This is an informational message only.
- **Action**: No action required.

QMAM-000008

- **Message** : Method-Reply Queue Message Handler Mapping : opcode = <code> associated with <class>
- **Description**: New Method-Reply Queue Message Handler instance (class name is <class>) has been created and associated with code <code>.
- **Cause** : This is an informational message only.
- **Action**: No action required.

QMAM-000009

- **Message** : Broker configuration <broker_id>: <connection_data>

- **Description:** Broker <broker_id> as been connected using <connection_data>.
- **Cause :** This is an informational message only.
- **Action:** No action required.

QMAN-000010

- **Message :** Incoming schema for <broker_id>::<package_name>.<class_name>.
- **Description:** Schema message response for class <broker_id>::<package_name>.<class_name> has been received.
- **Cause :** This is an informational message only.
- **Action:** No action required.

QMAN-000011

- **Message :** The shutdown sequence has been initiated for management client connected with broker <broker_id>
- **Description:** Management client connected to broker <broker_id> is closing.
- **Cause :** This is an informational message only.
- **Action:** No action required.

QMAN-000012

- **Message :** Management client connected with broker <broker_id> shut down successfully.
- **Description:** Management client connected to broker <broker_id> has been closed.
- **Cause :** This is an informational message only.
- **Action:** No action required.

QMAN-000013

- **Message :** Method-reply queue consumer has been successfully installed and bound on broker <broker_id>.
- **Description:** Method-reply queue consumer associated with broker <broker_id> has been created.
- **Cause :** This is an informational message only.
- **Action:** No action required.

QMAN-000014

- **Message :** Management queue consumer has been successfully installed and bound on broker <broker_id>.
- **Description:** Management queue consumer associated with broker <broker_id> has been created.
- **Cause :** This is an informational message only.
- **Action:** No action required.

QMAN-000015

- **Message :** Management queue with name <queue_name> has been successfully declared and bound on broker <broker_id>.
- **Description:** Management queue with name <queue_name> has been created on broker #<broker_id>
- **Cause :** This is an informational message only.
- **Action:** No action required.

QMAN-000016

- **Message :** Method-Reply queue with name <queue_name> has been successfully declared and bound on broker <broker_id>.
- **Description:** Method-Replyqueue with name <queue_name> has been created on broker #<broker_id>.
- **Cause :** This is an informational message only.
- **Action:** No action required.

QMAN-000017

- **Message :** Consumer <consumer_name> has been removed from broker <broker_id>.
- **Description:** A consumer with name <consumer_name> _has been removed from broker #<broker_id>._
- **Cause :** This is an informational message only.
- **Action:** No action required.

QMAN-000018

- **Message :** Queue <queue_name> has been removed from broker <broker_id>.
- **Description:** Queue <queue_name> _has been undeclared from broker #<broker_id>._
- **Cause :** This is an informational message only.
- **Action:** No action required.

QMAN-000019

- **Message :** QMan open for e-business.
- **Description:** QMan is up and running._
- **Cause :** This is an informational message only.
- **Action:** No action required.

QMAN-000020

- **Message** : Shutting down Q-Man...
- **Description**: The shutdown sequence has been initiated for QMan module.
- **Cause** : This is an informational message only.
- **Action**: No action required.

QMAN-000021

- **Message** : Q-Man shut down.
- **Description**: The shutdown sequence has been terminated for QMan module.
- **Cause** : This is an informational message only.
- **Action**: No action required.

QMAN-000022

- **Message** : Q-Man has no configured broker : in order to connect with a running one use Q-Man Administration interface.
- **Description**: QMan started without any configured broker (qman-config.xml is empty or it contains invalid broker connection). A new connection must be established at runtime.
- **Cause** : This is an informational message only.
- **Action**: In order to connect QMan with one or more broker you must use its administration interface.

QMAN-000023

- **Message** : Q-Man service is now available on MBeanServer.
- **Description**: QMan is itself a JMX MBean. This message informs that QMan was successfully registered with Management Server.
- **Cause** : This is an informational message only.
- **Action**: No action required.

QMAN-000026

- **Message** : Initializing WS-DM Adapter Environment...
- **Description**: WS-DM Adapter Environment is starting.
- **Cause** : This is an informational message only.
- **Action**: No action required.

QMAN-000027

- **Message** : WS-DM Adapter ready for incoming requests.
- **Description**: WS-DM Adapter successfully started.
- **Cause** : This is an informational message only.
- **Action**: No action required.

QMAN-000028

- **Message** : Qpid emulator not found. Test notifications are disabled.
- **Description**: Qpid emulator module was not found and therefore test notifications will be disabled.
- **Cause** : This is an informational message only.
- **Action**: No action required.

QMAN-000028

- **Message** : Default URI will be set to <default_uri>
- **Description**: WS-Resources default URI (used for endpoint references) will be set to <default_uri>
- **Cause** : This is an informational message only.
- **Action**: No action required.

QMan Warning Messages

Warning messages

QMAN-300001

- **Message** : No handler has been configured for processing messages with <op_code> as opcode. Message will be discarded.
- **Description** : QMan has no handler configured for operation code <op_code> and therefore the incoming message will be discarded.
- **Cause** : Probably something changed on broker side; a new operation code has been added and QMan doesn't know how to handle it.
- **Action** : No action is required as it's not possible to configure additional message handler.

QMAN-300002

- **Message** : Unable to deal with incoming message because it contains an unknown sequence number <sequence_number>.
- **Description** : The incoming message contains an unknown sequence number.
- **Cause** : After a method invocation timed out, sequence number used as correlation ID between request and response is discarded.

If after that, the response for that invocation is received, it will contain an unknown sequence number and therefore the message will be discarded.

- **Action** : If a method invocation times out there could be a lot of underlying causes. Basically we suggest to check if there are network issues.

QMAN-300003

- **Message** : Unable to enlist given broker connection data : QMan is already connected with broker *<broker_id>*.
- **Description** : A request has been made in order to connect QMan with an already connected broker.
- **Cause** : For each connected broker QMan creates a dedicated module called management client. It is not allowed to have two management clients connected with the same broker.
- **Action** : No action is required. The requested broker is already connected and a management client has already been created for it.

QMAN-300004

- **Message** : The given configuration file *<file>* is not valid (it doesn't exist or cannot be read)
- **Description**: the Path supplied at startup using system property qman-config refers to an invalid configuration file.
- **Cause** : The qman-config property points to a malformed or non-existing configuration file.
- **Action**: Check the value of the mentioned property and ensure that it is pointing to a correct configuration file.

QMAN-300005

- **Message** : Unable to initialize QEmu module and therefore emulation won't be enabled...
- **Description**: The QEmu sub-module cannot startup.
- **Cause** : This is a problem with the registration of the QEmu module with Management Server.
- **Action**: Check the logs in order to see the cause of the problem.

WS-DM Interface Specification

WS-DM Interface Specification

This section contains QMan WS-DM Interface Specification.

Each interface will be detailed using a dedicated subsection with the following template:

Name	Description
Description	A brief description of the interface purpose
Request	Analysis of a sample request that the interface expects
Response	Analysis of a sample response that the interface produces
Faults	Enumeration of all faults that are generated in error scenarios
Quick links	Useful external links about the interface (OASIS specs, etc...)

QMan has three different kinds of resources :

- 1 WSDM Adapter resource, which is the a facade that handles lifecycle of all managed WS-Resource;
- 0...* QMan WS Resource, which is a web service representation of a managed resource; This is directly part of QMan / Qpid management domain model;
- 1 Subscription Manager : the WS-Resource that enables notifications and allows clients to register themselves as notification listeners.

WS-DM Adapter WS-Resource

A stateful web service facade that acts as a front-end resource from a requestor endpoint perspective.

Interface Name	Description
MetadataExchange	Defines messages to retrieve metadata associated with the adapter endpoint.
Connect	Allows a requestor to connect QMan with a broker.
GetResourceMembers	Allows a requestor to retrieve the catalogue of all managed resources.
Subscribe	Allows a requestor to be registered as a notification listener.
GetCurrentMessage	Allows a requestor to retrieve the last notification published on a given topic.

QMan WS-Resource

A stateful Qpid entity (that is part of Qpid management domain model) represented as a web service (stateful) instance on QMan side and therefore exposed for management.

Interface Name	Description
MetadataExchange	Defines messages to retrieve metadata associated with a resource.
GetResourcePropertyDocument	Allows a requestor to retrieve the values of all resource properties associated with the resource.
PutResourcePropertyDocument	Allows a requestor to completely replace the values of a resource's properties with an entirely new resource property document.
GetResourceProperty	Allows a requestor to retrieve the value of a single resource property of a resource.
SetResourceProperties	Allows a requestor to modify the values of multiple resource properties of a resource.
GetMultipleResourceProperties	Allows a requestor to retrieve the values of multiple resource properties of a resource.
QueryResourceProperties	Allows a requestor to query the resource properties document of a resource using a query expression.
Operation invocation	Allows a requestor to invoke an operation exposed on the resource public interface.

Subscription Manager WS-Resource

A web service endpoint that provides operations that allow a service requestor to manage subscription resources.

Interface Name	Description
MetadataExchange	Defines messages to retrieve metadata associated with the subscription manager.
PauseSubscription	Allows a requestor to suspend the production of notifications on the given subscription.
ResumeSubscription	Allows a requestor to resume a previously suspended subscription.

Connect

Connect

- Description
- Request
- Response
- Faults
 - UnableToConnectFault
- Quick links

Description

The Connect interface allows to connect QMan with a Qpid broker.

Two categories of parameters need to be sent in order to make a connect request :

- connection parameters : host, port, username, password and virtual host name;
- connection pool parameters : for each connected broker a dedicated connection pool is created too. Those parameters allows a requestor to configure that pool.

Request

```

01. <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
02.   <soap:Header>
03.     <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
04.       http://localhost:8080/qman/services/adapter
05.     </wsa:To>
06.     <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
07.       http://amqp.apache.org/qpid/management/qman/Connect
08.     </wsa:Action>
09.     <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
10.       uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
11.     </wsa:MessageID>
12.     <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
13.       <wsa:Address>
14.         http://www.w3.org/2005/08/addressing/role/anonymous
15.       </wsa:Address>
16.     </wsa:From>
17.   </soap:Header>
18.   <soap:Body>
19.     <qman:Connect xmlns:qman="http://amqp.apache.org/qpid/management/qman">
20.       <qman:host>sofia.gazzax.com</qman:host>
21.       <qman:port>5672</qman:port>
22.       <qman:username>a.gazzarini</qman:username>
23.       <qman:password>plssw9rd</qman:password>
24.       <qman:virtualHost>test</qman:virtualHost>
25.       <qman:initialPoolCapacity>1</qman:initialPoolCapacity>
26.       <qman:maxPoolCapacity>4</qman:maxPoolCapacity>
27.       <qman:maxWaitTimeout>2000</qman:maxWaitTimeout>
28.     </qman:Connect>
29.   </soap:Body>
30. </soap:Envelope>

```

Line(s)	Description
01	The SOAP <Envelope> is the root element in every SOAP message, and contains two child elements, <Header> and <Body>.
02	The SOAP Header will contain all metadata used for identifying the conversation participants (requestor and provider).
03 - 05	Convey the target endpoint also known (in the request phase) as service provider.
06 - 08	Indicate this is a Connect request. This is done using a wsa:Action that is part of WS-Addressing specification.
09 - 11	Convey a unique identifier associated with the current message. This will be used for request / response messages correlation.
12 - 15	Provide the address of the source endpoint also known (in the request phase) as service requestor.
17 - 26	The connect request. Subsequent children specify connection parameters.
18	The host name / IP address where the broker is running.
19	The port number on which the broker is listening.
20	Username used for establishing the connection.
21	Password used for establishing the connection.
22	The virtual host name.
23	The initial size of broker dedicated connection pool. That means the number of connections that will be immediately created.
24	The maximum allowed size of broker dedicated connection pool.
25	The maximum wait timeout for retrieving connections from connection pool. A value of -1 means "Waits forever!"

Response

```

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
01.   <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
02.     http://www.w3.org/2005/08/addressing/role/anonymous
03.   </wsa:To>
04.   <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
05.     http://amqp.apache.org/qpid/management/qman/ConnectResponse
06.   </wsa:Action>
07.   <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
08.     uuid:980617c8-e3a0-ebf1-8f5a-2b43d3d6d416
09.   </wsa:MessageID>
10.   <wsa:RelatesTo RelationshipType="wsa:Reply" xmlns:wsa="http://www.w3.org/2005/08/addressing"
>
11.     uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
12.   </wsa:RelatesTo>
13.   <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
14.     <wsa:Address>
15.       http://localhost:8080/qman/services/adapter
16.     </wsa:Address>
17.   </wsa:From>
  </soap:Header>
  <soap:Body>
    <qman:ConnectResponse xmlns:qman="http://amqp.apache.org/qpid/management/qman" />
  </soapBody>
</soap:Envelope>

```

Line(s)	Description
01 - 03	Convey the recipient of the response message. Note that this time we are talking about the service requestor; The address matches the <wsa:From> previously found in the corresponding request
04 - 06	Indicate this is a Connect response. This is done as usual using a wsa:Action that is part of WS-Addressing specification
07 - 09	Convey a unique identifier associated with the current response message
10 - 12	This element provides the identifier of the correlated (request) message
13	The <wsa:From> element (part of WS-Addressing specs too) identifies the source endpoint, the originator of this response message
14 - 16	This is the address of the source service endpoint. As said for lines 01-03 this time this is referred to service provider (the message originator)
17	This is the connect response. Note that this is an empty element because this operation is void.

Faults

UnableToConnectFault

This is the only fault that could be returned as consequence of a connect request. That means QMan was unable to connect with the requested broker.

```

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
    <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
      http://www.w3.org/2005/08/addressing/role/anonymous
    </wsa:To>
    <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
      http://amqp.apache.org/qpid/management/qman/ConnectResponse
    </wsa:Action>
    <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
      uuid:980617c8-e3a0-ebf1-8f5a-2b43d3d6d416
    </wsa:MessageID>
    <wsa:RelatesTo RelationshipType="wsa:Reply" xmlns:wsa="http://www.w3.org/2005/08/addressing">
      uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
    </wsa:RelatesTo>
    <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
      <wsa:Address>
        http://localhost:8080/qman/services/adapter
      </wsa:Address>
    </wsa:From>
  </soap:Header>
  <soap:Body>
    01.   <soap:Fault>
    02.     <soap:Code xmlns:qman="http://amqp.apache.org/qpid/management/qman">
        <soap:Value>qMan</soap:Value>
    03.   </soap:Code>
    04.   <soap:Reason>
        <soap:Text>Unable to connect with the requested broker.</soap:Text>
    </soap:Reason>
    05.   <soap:Detail>
    06.     <qman:UnableToConnectFault xmlns:qman="http://amqp.apache.org/qpid/management/qman">
    07.       <wsrf-bf:Timestamp xmlns:wsrf-bf="http://docs.oasis-open.org/wsrf/bf-2">
            2009-02-17T10:37:08+01:00
          </wsrf-bf:Timestamp>
    08.       <wsrf-bf:OriginatorReference xmlns:wsrf-bf="http://docs.oasis-open.org/wsrf/bf-2">
        <wsa:ReferenceParameters xmlns:wsa="http://www.w3.org/2005/08/addressing"/>
        <wsa:Address xmlns:wsa="http://www.w3.org/2005/08/addressing">
          http://romagazzarini:8080/qman/services/adapter
        </wsa:Address>
    09.       <wsrf-bf:OriginatorReference>
    10.         <qman:host>sofia.gazzax.com</qman:host>
    11.         <qman:port>5672</qman:port>
    12.         <qman:username>a.gazzarini</qman:username>
    13.         <qman:virtualHost>plssw9rd</qman:virtualHost>
        </qman:UnableToConnectFault>
    </soap:Detail>
  </soap:Fault>
</soap:Body>
</soap:Envelope>

```

Line(s)	Description
01	This is a sub-element which is used on SOAP for reporting errors
02 - 03	Indicate the module that is throwing the exception
04 - 05	A human-readable text message which contains the reason of the failure.
05	An additional detail element of the current failure.
06	Custom fault element. Its name if UnableToConnectFault because we were unable to establish a connection.
07	The timestamp of the connection failure.
08 - 09	Reference data of the originator of this failure (QMan WS-DM Adapter)
10	host name of qpid broker that was passed as input arguments.
11	port number of qpid broker that was passed as input arguments.
12	username that was passed as input arguments.
13	password that was passed as input arguments.

N.A.

GetCurrentMessage

GetCurrentMessage

- Description
- Request
- Response
- Faults
- Quick links

Description

Allows a requestor to retrieve the last published message on a given topic. Note that this will be a non destructive read of the message, which won't be dequeued and therefore will be available to other requestors.

Request

```
01. <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
02.   <soap:Header>
03.     <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
04.       http://localhost:8080/qman/services/adapter
05.     </wsa:To>
06.       <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
07.         http://docs.oasis-open.org/wsn/bw-2/NotificationProducer/GetCurrentMessageRequest
08.       </wsa:Action>
09.         <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
10.           uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
11.         </wsa:MessageID>
12.         <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
13.           <wsa:Address>
14.             http://www.w3.org/2005/08/addressing/role/anonymous
15.           </wsa:Address>
16.         </wsa:From>
17.         </qman:ResourceId>
18.       </soap:Header>
19.       <soap:Body>
20.         <wsnt:GetCurrentMessage>
21.           <wsnt:Topic Dialect="http://docs.oasis-open.org/wsn/t-1/TopicExpression/Concrete">
22.             qman:EventsLifeCycleTopic
23.           </wsnt:Topic>
24.         </wsnt:GetCurrentMessage>
25.       </soap:Body>
26.     </soap:Envelope>
```

Line(s)	Description
01	The SOAP <Envelope> is the root element in every SOAP message, and contains two child elements, <Header> and <Body>.
02	The SOAP Header will contain all metadata used for identifying the conversation participants (requestor and provider).
03 - 05	Convey the target endpoint also known (in the request phase) as service provider.
06 - 08	Indicate this is a GetCurrentMessage request.
09 - 11	Convey a unique identifier associated with the current message. This will be used for request / response messages correlation.
12 - 15	Provide the address of the source endpoint also known (in the request phase) as service requestor.
18 - 22	The GetCurrentMessage request. As you can see requested the example shows how to request the current message of the qman:EventsLifeCycleTopic topic. This is the topic where event lifecycle notifications are published.

Response

```

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
01.    <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
02.      http://www.w3.org/2005/08/addressing/role/anonymous
03.    </wsa:To>
04.    <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
05.      http://docs.oasis-open.org/wsn/bw-2/NotificationProducer/GetCurrentMessageResponse
06.    </wsa:Action>
07.    <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
08.      uuid:980617c8-e3a0-ebf1-8f5a-2b43d3d6d416
09.    </wsa:MessageID>
10.    <wsa:RelatesTo RelationshipType="wsa:Reply" xmlns:wsa="http://www.w3.org/2005/08/addressing"
>
11.      uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
12.    </wsa:RelatesTo>
13.    <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
14.      <wsa:Address>
15.        http://localhost:8080/qman/services/adapter
16.      </wsa:Address>
20.      </wsa:From>
    </soap:Header>
    <soap:Body>
21.      <wsnt:GetCurrentMessageResponse>
22.        <wsnt:Message>
          <qman:LifeCycleEvent
            xmlns:qman="http://amqp.apache.org/qpid/management/qman"
            TimeMillis="1234295015000" Type="CREATED">
              <qman:Resource>
                <qman-wsa:ResourceId
                  xmlns:qman-wsa="http://amqp.apache.org/qpid/management/qman/addressing">
                    aff2f6ec-2e5c-4768-ae87-6da2c8a005ff
                  </qman-wsa:ResourceId>
                  <qman:PackageName>org.apache.qpid.broker
                </qman:PackageName>
                <qman:Name>connection</qman:Name>
              </qman:Resource>
            </qman:LifeCycleEvent>
23.        </wsnt:Message>
24.      </wsnt:GetCurrentMessageResponse>
    </soapBody>
  </soap:Envelope>

```

Line(s)	Description
01 - 03	Convey the recipient of the response message. Note that this time we are talking about the service requestor; The address matches the <wsa:From> previously found in the corresponding request.
04 - 06	Indicate this is a GetCurrentMessage response. This is done as usual using a wsa:Action that is part of WS-Addressing specification.
07 - 09	Convey a unique identifier associated with the current response message.
10 - 12	This element provides the identifier of the correlated (request) message.
13 - 20	The <wsa:From> element (part of WS-Addressing specs too) identifies the source endpoint, the originator of this response message.
14 - 16	This is the address of the source service endpoint. As said for lines 01-03 this time this is referred to service provider (the message originator).
17 - 20	As part of wsa:From element, this contains (specifically on line 18) additional information needed for identifying the originator of this message.
21 - 24	This is the GetCurrentMessage response. That will contain as a nested child the last published message (lines 22 - 23).

Faults

- **ResourceUnknownFault** : There's no resource on QMan associated with the given reference information (soap address and identifier).
- **ResourceUnavailableFault** : The requested resource is unavailable. This fault should indicate a transient condition. That means a requester might resend the message.
- **TopicExpressionDialectUnknownFault** : The topic expression dialect is unknown.
- **InvalidTopicExpressionFault** : The topic expression is not valid for the specified dialect.
- **TopicNotSupportedFault** : The requested topic is not supported.

- **MultipleTopicsSpecifiedFault** : The topic expression is ambiguous because is referring to multiple topic.
- **NoCurrentMessageOnTopicFault** : There's no message available on the requested topic.

Quick links

[Web Services Base Notification](#)
[Web Services Topics](#)

GetMultipleResourceProperties

GetMultipleResourceProperties

- Description
- Request
- Response
- Faults
- Quick links

Description

This interface allows a requestor to retrieve the values of multiple resource properties of a WS-Resource.

Request

```

01. <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
02.   <soap:Header>
03.     <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
04.       http://localhost:8080/qman/services/QManWsResource
05.     </wsa:To>
06.       <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
07.
http://docs.oasis-open.org/wsrf/rpw-2/GetMultipleResourceProperties/GetMultipleResourcePropertiesRequ
</wsa:Action>
09.           <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
10.             uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
11.           </wsa:MessageID>
12.           <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
13.             <wsa:Address>
14.               http://www.w3.org/2005/08/addressing/role/anonymous
15.             </wsa:Address>
16.           </wsa:From>
17.           <qman:ResourceId
      xmlns:wsa="http://www.w3.org/2005/08/addressing"
      wsa:IsReferenceParameter="true"
      xmlns:qman="http://amqp.apache.org/qpid/management/qman">
18.             781f4ad7-4c96-4caa-b69d-291461cdb1fc
19.           </qman:ResourceId>
</soap:Header>
<soap:Body>
20.   <wsrf-rp:GetMultipleResourceProperties
      xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
      xmlns:qman="http://amqp.apache.org/qpid/management/qman">
21.     <wsrf-rp:ResourceProperty>qman:MgmtPubInterval</wsrf-rp:ResourceProperty>
22.     <wsrf-rp:ResourceProperty>qman:Name</wsrf-rp:ResourceProperty>
23.     <wsrf-rp:ResourceProperty>qman:MsgTotalEnqueues</wsrf-rp:ResourceProperty>
24.     <wsrf-rp:ResourceProperty>qman:Arguments</wsrf-rp:ResourceProperty>
25.     <wsrf-rp:ResourceProperty>qman:VhostRef</wsrf-rp:ResourceProperty>
26.     <wsrf-rp:ResourceProperty>qman:ExpireTime</wsrf-rp:ResourceProperty>
27.     <wsrf-rp:ResourceProperty>qman:Durable</wsrf-rp:ResourceProperty>
28.     <wsrf-rp:ResourceProperty>qman:ConsumerCount</wsrf-rp:ResourceProperty>
29.     <wsrf-rp:ResourceProperty>qman:Type</wsrf-rp:ResourceProperty>
30.   </wsrf-rp:GetMultipleResourceProperties>
</soap:Body>
</soap:Envelope>
```

Line(s)	Description
01	The SOAP <Envelope> is the root element in every SOAP message, and contains two child elements, <Header> and <Body>.
02	The SOAP Header will contain all metadata used for identifying the conversation participants (requestor and provider).
03 - 05	Convey the target endpoint also known (in the request phase) as service provider.

06 - 08	Indicate this is a GetMultipleResourceProperties request.
09 - 11	Convey a unique identifier associated with the current message. This will be used for request / response messages correlation.
12 - 15	Provide the address of the source endpoint also known (in the request phase) as service requestor.
17 - 19	This indicates the target resource of this request. Specifically the line 18 contains the resource identifier.
20 - 30	The GetMultipleResourceProperties request. As you can see requested properties are nested children (line 21 - 29).
21 - 29	Each wsrf-rp:ResourceProperty contains a resource property QName.

Response

```

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
01.    <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
02.      http://www.w3.org/2005/08/addressing/role/anonymous
03.    </wsa:To>
04.    <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
05.
http://docs.oasis-open.org/wsrf/rpw-2/GetMultipleResourceProperties/GetMultipleResourcePropertiesRespon
</wsa:Action>
07.    <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
08.      uuid:980617c8-e3a0-ebf1-8f5a-2b43d3d6d416
09.    </wsa:MessageID>
10.    <wsa:RelatesTo RelationshipType="wsa:Reply" xmlns:wsa="http://www.w3.org/2005/08/addressing">
>
11.      uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
12.    </wsa:RelatesTo>
13.    <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
14.      <wsa:Address>
15.        http://localhost:8080/qman/services/QManWsResource
16.      </wsa:Address>
17.      <wsa:ReferenceParameters>
<qman:ResourceId
  xmlns:wsa="http://www.w3.org/2005/08/addressing"
  wsa:IsReferenceParameter="true"
  xmlns:qman="http://amqp.apache.org/qpid/management/qman">
18.        781f4ad7-4c96-4caa-b69d-291461cdb1fc
19.      </qman:ResourceId>
20.      </wsa:ReferenceParameters>
    </wsa:From>
  </soap:Header>
  <soap:Body>
21.    <wsrf-rp:GetMultipleResourcePropertiesResponse
      xmlns:qman="http://amqp.apache.org/qpid/management/qman"
      xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2">
22.      <qman:MgmtPubInterval>32767</qman:MgmtPubInterval>
23.      <qman:Name>Initial Name</qman:Name>
24.      <qman:MsgTotalEnqueues>9223372036854775797</qman:MsgTotalEnqueues>
25.      <qman:Arguments xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<qman:entry>
<qman:key>Key3</qman:key>
<qman:value xsi:type="xsd:integer">2147483647</qman:value>
</qman:entry>
<qman:entry>
<qman:key>Key4</qman:key>
<qman:value xsi:type="xsd:float">3.4028235E38</qman:value>
</qman:entry>
<qman:entry>
<qman:key>Key1</qman:key>
<qman:value xsi:type="xsd:string">aStringValue</qman:value>
</qman:entry>
<qman:entry>
<qman:key>Key2</qman:key>
<qman:value xsi:type="xsd:long">-9223372036854775808</qman:value>
</qman:entry>
</qman:Arguments>
27.      <qman:VhostRef>2deef1b3-d2c6-49f3-a8de-51f6a75ala6b</qman:VhostRef>
28.      <qman:ExpireTime>9223372036854775807</qman:ExpireTime>
29.      <qman:Durable>true</qman:Durable>
30.      <qman:ConsumerCount>-2147483638</qman:ConsumerCount>
31.    </wsrf-rp:GetMultipleResourcePropertiesResponse>
  </soapBody>
</soap:Envelope>

```

Line(s)	Description
01 - 03	Convey the recipient of the response message. Note that this time we are talking about the service requestor; The address matches the <wsa:From> previously found in the corresponding request.
04 - 06	Indicate this is a GetMultipleResourceProperties response. This is done as usual using a wsa:Action that is part of WS-Addressing specification.
07 - 09	Convey a unique identifier associated with the current response message.

10 - 12	This element provides the identifier of the correlated (request) message.
13	The <wsa:From> element (part of WS-Addressing specs too) identifies the source endpoint, the originator of this response message.
14 - 16	This is the address of the source service endpoint. As said for lines 01-03 this time this is referred to service provider (the message originator).
17 - 20	As part of wsa:From element, this contains (specifically on line 18) additional information needed for identifying the originator of this message.
21 - 31	This is the GetMultipleResourceProperties response element which contains the requested property values as nested child.
22 - 30	Each element represents a requested properties. Note that the name of the element is the name of the property. For simple types the corresponding value is directly reported as a text content; complex types like maps (line 25 - 26) serialization is different and is declared on the WSDL of the resource.

Faults

- **ResourceUnknownFault** : There's no resource on QMan associated with the given reference information (soap address and identifier).
- **ResourceUnavailableFault** : The requested resource is unavailable. This fault should indicate a transient condition. That means a requester might resend the message.
- **InvalidResourcePropertyQNameFault** : One or more of the names (QNames) in the request message doesn't correspond to a property element of the target WS-Resource.

Quick links

[Web Services Resource](#)
[Web Services Resource Properties](#)

GetResourceMembers

GetResourceMembers

- Description
- Request
- Response
- Faults
- Quick links

Description

This interface allows a requestor to retrieve the catalogue of all managed resources.
Note that there's no a "GetResourceMembers" request on WSRF specification. That request is actually a GetResourceProperty for wsrf-sg:Entry adapter property.
For each managed resource a dedicated entry will be returned on the corresponding response.

Request

```

01. <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
02.   <soap:Header>
03.     <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
04.       http://localhost:8080/qman/services/adapter
05.     </wsa:To>
06.     <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
07.       http://docs.oasis-open.org/wsrf/rpw-2/GetResourceProperty/GetResourcePropertyRequest
08.     </wsa:Action>
09.     <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
10.       uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
11.     </wsa:MessageID>
12.     <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
13.       <wsa:Address>
14.         http://www.w3.org/2005/08/addressing/role/anonymous
15.       </wsa:Address>
16.     </wsa:From>
17.   </soap:Header>
18.   <soap:Body>
19.     <wsrf-rp:GetResourceProperty
        xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
        xmlns:wsrf-sg="http://docs.oasis-open.org/wsrf/sg-2">
20.       wsrf-sg:Entry
21.     </wsrf-rp:GetResourceProperty>
22.   </soap:Body>
23. </soap:Envelope>
```

Line(s)	Description
01	The SOAP <Envelope> is the root element in every SOAP message, and contains two child elements, <Header> and <Body>.
02	The SOAP Header will contain all metadata used for identifying the conversation participants (requestor and provider).
03 - 05	Convey the target endpoint also known (in the request phase) as service provider.
06 - 08	Indicate this is a GetResourceProperty request. In fact, the GetResourceMember is a GetResourceProperty request where a requestor ask to the adapter resource the value of wsrfs-rg:Entry property value.
09 - 11	Convey a unique identifier associated with the current message. This will be used for request / response messages correlation.
12 - 15	Provide the address of the source endpoint also known (in the request phase) as service requestor.
17 - 19	The GetResourceProperty request. Subsequent child specifies requested property name.
18	This adapter property is basically the list of all managed resources that are currently under QMan management domain.

Response

```

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
01.    <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
02.      http://www.w3.org/2005/08/addressing/role/anonymous
03.    </wsa:To>
04.    <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
05.      http://docs.oasis-open.org/wsrp/rp-w-2/GetResourceProperty/GetResourcePropertyResponse
06.    </wsa:Action>
07.    <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
08.      uuid:980617c8-e3a0-ebf1-8f5a-2b43d3d6d416
09.    </wsa:MessageID>
10.    <wsa:RelatesTo RelationshipType="wsa:Reply" xmlns:wsa="http://www.w3.org/2005/08/addressing"
>
11.      uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
12.    </wsa:RelatesTo>
13.    <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
14.      <wsa:Address>
15.        http://localhost:8080/qman/services/adapter
16.      </wsa:Address>
17.    </wsa:From>
18.  </soap:Header>
19.  <soap:Body>
20.    <wsrf-sg:Entry xmlns:wsrf-sg="http://docs.oasis-open.org/wsrf/sg-2">
21.      <wsrf-sg:ServiceGroupEntryEPR>
22.        <wsa:Address xmlns:wsa="http://www.w3.org/2005/08/addressing">
23.          http://localhost:8080/qman/services/ServiceGroupEntry
24.        </wsa:Address>
25.        <wsa:ReferenceParameters xmlns:wsa="http://www.w3.org/2005/08/addressing">
26.          <qman:ResourceId xmlns:qman="http://amqp.apache.org/qpid/management/qman">
27.            1d01b4ee-7d23-3a30-342e-62fc49984fe6
28.          </qman:ResourceId>
29.        </wsa:ReferenceParameters>
30.      </wsrf-sg:ServiceGroupEntryEPR>
31.      <wsrf-sg:MemberServiceEPR>
32.        <wsa:Address xmlns:wsa="http://www.w3.org/2005/08/addressing">
33.          http://localhost:8080/qman/services/QManWsResource
34.        </wsa:Address>
35.        <wsa:ReferenceParameters xmlns:wsa="http://www.w3.org/2005/08/addressing">
36.          <qman:ResourceId
37.            xmlns:qman="http://amqp.apache.org/qpid/management/qman">
38.            a3759467-bede-476d-8dde-169f1a652191
39.          </qman:ResourceId>
40.        </wsa:ReferenceParameters>
41.      </wsrf-sg:MemberServiceEPR>
42.      <wsrf-sg:Content/>
43.    </wsrf-sg:Entry>
44.    <wsrf-sg:Entry xmlns:wsrf-sg="http://docs.oasis-open.org/wsrf/sg-2">
45.      ...
46.    </wsrf-sg:Entry>
47.    <wsrf-sg:Entry xmlns:wsrf-sg="http://docs.oasis-open.org/wsrf/sg-2">
48.      ...
49.    </wsrf-sg:Entry>
50.  </wsrf-rp:GetResourcePropertyResponse>
51. </soapBody>
52. </soap:Envelope>

```

Line(s)	Description
01 - 03	Convey the recipient of the response message. Note that this time we are talking about the service requestor; The address matches the <wsa:From> previously found in the corresponding request
04 - 06	Indicate this is a GetResourceProperty response. This is done as usual using a wsa:Action that is part of WS-Addressing specification
07 - 09	Convey a unique identifier associated with the current response message
10 - 12	This element provides the identifier of the correlated (request) message
13	The <wsa:From> element (part of WS-Addressing specs too) identifies the source endpoint, the originator of this response message
14 - 16	This is the address of the source service endpoint. As said for lines 01-03 this time this is referred to service provider (the message originator)

17	This element represent a single resource as part of QMan management domain.
18 - 22	The "group" membership information for a specific resource are detailed using a SericeGroupEntry which is separated from the WS-Resource itself. In fact, the service group entry is a WS-Resource itself which has its own address (line 20) and identifier (line 21). That means we could have properties belonging to group membership of WS-Resource, not to WS-Resource itself.
24 - 28	The resource member that is part of QMan management group. This element contains all what is needed for identifying and addressing the WS-Resource.
25	This is the resource soap:address (as declared on its WSDL)
26 - 27	The WS-Resource identifier. This is the most important information about resource. Subsequent requests directed to resource will contain this identifier.
29	Additional (optional) management group information.
33 - 34	Those are two additional entry summaries. Note that for each resource there's a dedicated wsrf-sg:Entry.

Faults

No specific fault are thrown by this operation. A general Soap fault could be returned if, for example, the adapter is not running or is not working.

Quick links

[Web Services Service Group specification](#)
[Web Services Resource](#)
[Web Services Resource Properties](#)

GetResourceProperty

GetResourceProperty

- Description
- Request
- Response
- Faults
- Quick links

Description

This interface allows a requestor to retrieve the value of a property of a WS-Resource.

Request

```

01. <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
02.   <soap:Header>
03.     <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
04.       http://localhost:8080/qman/services/QManWsResource
05.     </wsa:To>
06.       <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
07.         http://docs.oasis-open.org/wsrf/rpw-2/GetResourceProperty/GetResourcePropertyRequest
08.       </wsa:Action>
09.       <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
10.         uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
11.       </wsa:MessageID>
12.       <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
13.         <wsa:Address>
14.           http://www.w3.org/2005/08/addressing/role/anonymous
15.         </wsa:Address>
16.       </wsa:From>
17.       <qman:ResourceId
18.         xmlns:wsa="http://www.w3.org/2005/08/addressing"
19.         wsa:IsReferenceParameter="true"
20.         xmlns:qman="http://amqp.apache.org/qpid/management/qman">
21.           781f4ad7-4c96-4caa-b69d-291461cdb1fc
22.         </qman:ResourceId>
23.       </soap:Header>
24.     <soap:Body>
25.       <wsrf-rp:GetResourceProperty
26.         xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
27.         xmlns:qman="http://amqp.apache.org/qpid/management/qman">
28.           qman:MgmtPubInterval
29.         </wsrf-rp:GetResourceProperty>
30.       </soap:Body>
31.     </soap:Envelope>

```

Line(s)	Description
01	The SOAP <Envelope> is the root element in every SOAP message, and contains two child elements, <Header> and <Body>.
02	The SOAP Header will contain all metadata used for identifying the conversation participants (requestor and provider).
03 - 05	Convey the target endpoint also known (in the request phase) as service provider.
06 - 08	Indicate this is a GetResourceProperty request.
09 - 11	Convey a unique identifier associated with the current message. This will be used for request / response messages correlation.
12 - 15	Provide the address of the source endpoint also known (in the request phase) as service requestor.
17 - 19	This indicates the target resource of this request. Specifically the line 18 contains the resource identifier.
20 - 22	The GetResourceProperty request. The name of the property is the text content of this element (line 21).
21	Indicates the name of the property. In the example above the requestor is asking for the value of the MgmtPubInterval property.

Response

```

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
01.   <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
02.     http://www.w3.org/2005/08/addressing/role/anonymous
03.   </wsa:To>
04.   <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
05.     http://docs.oasis-open.org/wsrf/rpw-2/GetResourceProperty/GetResourcePropertyResponse
06.   </wsa:Action>
07.   <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
08.     uuid:980617c8-e3a0-ebf1-8f5a-2b43d3d6d416
09.   </wsa:MessageID>
10.   <wsa:RelatesTo RelationshipType="wsa:Reply" xmlns:wsa="http://www.w3.org/2005/08/addressing"
>
11.     uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
12.   </wsa:RelatesTo>
13.   <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
14.     <wsa:Address>
15.       http://localhost:8080/qman/services/QManWsResource
16.     </wsa:Address>
17.     <wsa:ReferenceParameters>
      <qman:ResourceId
        xmlns:wsa="http://www.w3.org/2005/08/addressing"
        wsa:IsReferenceParameter="true"
        xmlns:qman="http://amqp.apache.org/qpid/management/qman">
18.       781f4ad7-4c96-4caa-b69d-291461cdb1fc
19.     </qman:ResourceId>
20.   </wsa:ReferenceParameters>
    </wsa:From>
  </soap:Header>
  <soap:Body>
21.   <wsrf-rp:GetResourcePropertyResponse xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2">
22.     <qman:MgmtPubInterval xmlns:qman="http://amqp.apache.org/qpid/management/qman">
23.       32767
24.     </qman:MgmtPubInterval>
25.   </wsrf-rp:GetResourcePropertyResponse>
  </soapBody>
</soap:Envelope>

```

Line(s)	Description
01 - 03	Convey the recipient of the response message. Note that this time we are talking about the service requestor; The address matches the <wsa:From> previously found in the corresponding request.
04 - 06	Indicate this is a GetResourceProperty response. This is done as usual using a wsa:Action that is part of WS-Addressing specification.
07 - 09	Convey a unique identifier associated with the current response message.
10 - 12	This element provides the identifier of the correlated (request) message.
13	The <wsa:From> element (part of WS-Addressing specs too) identifies the source endpoint, the originator of this response message.
14 - 16	This is the address of the source service endpoint. As said for lines 01-03 this time this is referred to service provider (the message originator).
17 - 20	As part of wsa:From element, this contains (specifically on line 18) additional information needed for identifying the originator of this message.
21 - 25	This is the GetResourceProperty response element which contains the requested property as nested child.
22 - 24	This element represents the requested property. Note that the name of the element is the name of the property.
23	Here is the value of the requested property.

Faults

- **ResourceUnknownFault** : There's no resource on QMan associated with the given reference information (soap address and identifier).
- **ResourceUnavailableFault** : The requested resource is unavailable. This fault should indicate a transient condition. That means a requester might resend the message.
- **InvalidResourcePropertyQNameFault** : The name (QName) in the request message doesn't correspond to a property element of the target WS-Resource.|

Quick links

[Web Services Resource](#)
[Web Services Resource Properties](#)

GetResourcePropertyDocument

GetResourcePropertyDocument

- Description
- Request
- Response
- Faults
- Quick links

Description

This interface allows a requestor to retrieve the values of all resource properties associated with the WS-Resource.
Extract from WS-Resource properties specification :

*"The ResourcePropertyDocument is the XML document representing a logical composition of resource property elements.
The resource properties document defines a particular view or projection of the state data implemented by the WS-Resource."*

Request

```
01. <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
02.   <soap:Header>
03.     <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
04.       http://localhost:8080/qman/services/QManWsResource
05.     </wsa:To>
06.     <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
07.
http://docs.oasis-open.org/wsrf/rpw-2/GetResourcePropertyDocument/GetResourcePropertyDocumentRequest
</wsa:Action>
09.     <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
10.       uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
11.     </wsa:MessageID>
12.     <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
13.       <wsa:Address>
14.         http://www.w3.org/2005/08/addressing/role/anonymous
15.       </wsa:Address>
16.     </wsa:From>
17.     <qman:ResourceId
      xmlns:wsa="http://www.w3.org/2005/08/addressing"
      wsa:IsReferenceParameter="true"
      xmlns:qman="http://amqp.apache.org/qpid/management/qman">
18.       781f4ad7-4c96-4caa-b69d-291461cdb1fc
19.     </qman:ResourceId>
</soap:Header>
<soap:Body>
20.   <wsrf-rp:GetResourcePropertyDocument xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"/>
</soap:Body>
</soap:Envelope>
```

Line(s)	Description
01	The SOAP <Envelope> is the root element in every SOAP message, and contains two child elements, <Header> and <Body>.
02	The SOAP Header will contain all metadata used for identifying the conversation participants (requestor and provider).
03 - 05	Convey the target endpoint also known (in the request phase) as service provider.
06 - 08	Indicate this is a GetResourcePropertyDocument request.
09 - 11	Convey a unique identifier associated with the current message. This will be used for request / response messages correlation.
12 - 15	Provide the address of the source endpoint also known (in the request phase) as service requestor.
17 - 19	This indicates the target resource of this request. Specifically the line 18 contains the resource identifier.
20	The GetResourcePropertyDocument request. Note that this is an empty element and therefore there's no additional parameter.

Response

```

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
01.    <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
02.      http://www.w3.org/2005/08/addressing/role/anonymous
03.    </wsa:To>
04.    <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
05.
http://docs.oasis-open.org/wsrf/rpw-2/GetResourcePropertyDocument/GetResourcePropertyDocumentResponse
</wsa:Action>
07.    <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
08.      uuid:980617c8-e3a0-ebf1-8f5a-2b43d3d6d416
09.    </wsa:MessageID>
10.    <wsa:RelatesTo RelationshipType="wsa:Reply" xmlns:wsa="http://www.w3.org/2005/08/addressing">
>
11.      uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
12.    </wsa:RelatesTo>
13.    <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
14.      <wsa:Address>
15.        http://localhost:8080/qman/services/adapter
16.      </wsa:Address>
17.      <wsa:ReferenceParameters>
<qman:ResourceId
  xmlns:wsa="http://www.w3.org/2005/08/addressing"
  wsa:IsReferenceParameter="true"
  xmlns:qman="http://amqp.apache.org/qpid/management/qman">
18.        781f4ad7-4c96-4caa-b69d-291461cdb1fc
19.      </qman:ResourceId>
20.      </wsa:ReferenceParameters>
    </wsa:From>
  </soap:Header>
  <soap:Body>
21.    <wsrf-rp:GetResourcePropertyDocumentResponse xmlns:wsrf-rp=
"http://docs.oasis-open.org/wsrf/rp-2">
22.      <qman:QManWsResourceProperties xmlns:qman="http://amqp.apache.org/qpid/management/qman">
23.        <qman:MgmtPubInterval>32767</qman:MgmtPubInterval>
24.        <wsrf-rp:QueryExpressionDialect>http://www.w3.org/TR/1999/REC-xpath-19991116
</wsrf-rp:QueryExpressionDialect>
25.        <qman:Name>Initial Name</qman:Name>
26.        <wsrf-rl:TerminationTime xmlns:wsrf-rl="http://docs.oasis-open.org/wsrf/rl-2"/>
27.        <qman:MsgTotalEnqueues>9223372036854775797</qman:MsgTotalEnqueues>
28.        <qman:Arguments xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<qman:entry>
<qman:key>Key3</qman:key>
<qman:value xsi:type="xsd:integer">2147483647</qman:value>
</qman:entry>
<qman:entry>
<qman:key>Key4</qman:key>
<qman:value xsi:type="xsd:float">3.4028235E38</qman:value>
</qman:entry>
<qman:entry>
<qman:key>Key1</qman:key>
<qman:value xsi:type="xsd:string">aStringValue</qman:value>
</qman:entry>
<qman:entry>
<qman:key>Key2</qman:key>
<qman:value xsi:type="xsd:long">-9223372036854775808</qman:value>
</qman:entry>
</qman:Arguments>
<qman:VhostRef>2deef1b3-d2c6-49f3-a8de-51f6a75a1a6b</qman:VhostRef>
31.        <wsrf-rl:CurrentTime xmlns:wsrf-rl="http://docs.oasis-open.org/wsrf/rl-2">1232956293823
</wsrf-rl:CurrentTime>
32.        <qman:ExpireTime>9223372036854775807</qman:ExpireTime>
33.        <qman:Durable>true</qman:Durable>
34.        <qman:ConsumerCount>-2147483638</qman:ConsumerCount>
35.      </qman:QManWsResourceProperties>
36.    </wsrf-rp:GetResourcePropertyDocumentResponse>
  </soapBody>
</soap:Envelope>

```

Line(s)	Description
---------	-------------

01 - 03	Convey the recipient of the response message. Note that this time we are talking about the service requestor; The address matches the <wsa:From> previously found in the corresponding request.
04 - 06	Indicate this is a GetResourcePropertyDocument response. This is done as usual using a wsa:Action that is part of WS-Addressing specification.
07 - 09	Convey a unique identifier associated with the current response message.
10 - 12	This element provides the identifier of the correlated (request) message.
13	The <wsa:From> element (part of WS-Addressing specs too) identifies the source endpoint, the originator of this response message.
14 - 16	This is the address of the source service endpoint. As said for lines 01-03 this time this is referred to service provider (the message originator).
17 - 20	As part of wsa:From element, this contains (specifically on line 18) additional information needed for identifying the originator of this message.
21 - 36	This is the GetResourcePropertyDocumentResponse element which contains the resource property document as nested child.
22 - 35	Resource Property Document. The example refers to a resource that has 11 properties (lines 23 - 34).

Faults

- **wsrf-rw:ResourceUnknownFault** : There's no resource on QMan associated with the given reference information (soap address and identifier).
- **wsrf-rw:ResourceUnavailableFault** : the requested resource is unavailable. This fault should indicate a transient condition. That means a requester might resend the message.

Quick links

[Web Services Resource](#)
[Web Services Resource Properties](#)

MetadataExchange

Metadata Exchange (WS-MetadataExchange)

- Description
- Request
- Response
 - WSDL Dialect
 - RMD Dialect
- Faults
- Quick links

Description

QMan WS-Resources are basically web services. Web Services use metadata to describe what other endpoints need to know in order to interact with them.

The MetadataExchange interface allows a requestor to query a specific WS-Resource for its metadata.

Request

```

01. <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
02.   <soap:Header>
03.     <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
04.       http://localhost:8080/qman/services/QManWsResource
05.     </wsa:To>
06.     <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
07.       http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadata
08.     </wsa:Action>
09.     <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
10.       uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
11.     </wsa:MessageID>
12.     <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
13.       <wsa:Address>
14.         http://www.w3.org/2005/08/addressing/role/anonymous
15.       </wsa:Address>
16.     </wsa:From>
17.     <qman-wsa:ResourceId
      xmlns:wsa="http://www.w3.org/2005/08/addressing"
      xmlns:qman-wsa="http://amqp.apache.org/qpid/management/qman/addressing"
      wsa:IsReferenceParameter="true">
18.       a3759467-bede-476d-8dde-169f1a652191
19.     </qman-wsa:ResourceId>
20.   </soap:Header>
21.   <soap:Body>
22.     <wsx:GetMetadata xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex">
23.       <wsx:Dialect>
24.         http://schemas.xmlsoap.org/wsdl/
25.       </wsx:Dialect>
26.     </wsx:GetMetadata>
27.   </soap:Body>
28. </soap:Envelope>

```

Line(s)	Description
01	The SOAP <Envelope> is the root element in every SOAP message, and contains two child elements, <Header> and <Body>
02	The SOAP Header will contain all metadata used for identifying the conversation participants (requestor and provider)
03 - 05	Convey the target endpoint also known (in the request phase) as service provider. Note that an additional information (ResourceId) needs to be supplied in order to correctly identify the target WS-Resource
06 - 08	Indicate this is a Get Metadata request. This is done using a wsa:Action that is part of WS-Addressing specification
09 - 11	Convey a unique identifier associated with the current message. This will be used for request / response messages correlation
12 - 15	Provide the address of the source endpoint also known (in the request phase) as service requestor
17 - 19	Provide the WS-Resource identifier. That allows to correctly identify the requested instance
20	The SOAP <Body> is a mandatory sub-element of the Envelope, which contains information intended for the recipient of the current message;
21	The GetMetadata request;
22	Dialect associated with the requested metadata. We could say that it identifies a specific kind of metadata. As MetadataExchange specs says :
	<i>"When this element is present, the response MUST include only Metadata Sections with the indicated dialect; if the receiver does not have any Metadata Sections of the indicated dialect, the response MUST include zero Metadata Sections.</i>
	<i>When this element is not present, the implied value is any dialect."</i>
	At the moment there are two supported dialects :
	<ul style="list-style-type: none"> • Web Service Description Language (WSDL) : dialect in this case is : http://schemas.xmlsoap.org/wsdl/ • Resource Metadata Descriptor (RMD) : dialect in this case is : http://docs.oasis-open.org/wsrif/rmd-1_

Response

MetadataExchange supports two dialects and therefore there could be two different responses depending on the requested dialect.

WSDL Dialect

The following illustrates an example response of a GetMetadata request with WSDL dialect.
For simplicity only the top level <wsdl:definitions> element has been reported. You can find a complete metadata exchange conversation

under the example directory.

```

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Header>
01.     <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
02.         http://www.w3.org/2005/08/addressing/role/anonymous
03.     </wsa:To>
04.     <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
05.         http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadataResponse
06.     </wsa:Action>
07.     <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
08.         uuid:980617c8-e3a0-ebf1-8f5a-2b43d3d6d416
09.     </wsa:MessageID>
10.     <wsa:RelatesTo RelationshipType="wsa:Reply" xmlns:wsa="http://www.w3.org/2005/08/addressing">
11.         uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
12.     </wsa:RelatesTo>
13.     <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
14.         <wsa:Address>
15.             http://localhost:8080/qman/services/QManWsResource
16.         </wsa:Address>
17.         <wsa:ReferenceParameters>
18.             <qman-wsa:ResourceId
wsa:IsReferenceParameter="true"
                xmlns:qman-wsa=
"http://amqp.apache.org/qpid/management/qman/addressing"
                xmlns:wsa="http://www.w3.org/2005/08/addressing">
19.                 a3759467-bede-476d-8dde-169f1a652191
20.             </qman-wsa:ResourceId>
                </wsa:ReferenceParameters>
            </wsa:From>
        </soap:Header>
        <soap:Body>
21.         <wsx:Metadata xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex">
22.             <wsx:MetadataSection>
23.                 <wsdl:definitions>
...
24.                 </wsdl:definitions>
            </wsx:MetadataSection>
        </wsx:Metadata>
    </soapBody>
</soap:Envelope>

```

Line(s)	Description
01 - 03	Convey the recipient of the response message. Note that this time we are talking about the service requestor; The address matches the <wsa:From> previously found in the corresponding request
04 - 06	Indicate this is a Get Metadata response. This is done as usual using a wsa:Action that is part of WS-Addressing specification
07 - 09	Convey a unique identifier associated with the current response message
10 - 12	This element provides the identifier of the correlated (request) message
13	The <wsa:From> element (part of WS-Addressing specs too) identifies the source endpoint, the originator of this response message
14 - 15	This is the address of the source service endpoint. As said for lines 01-03 this time this is referred to service provider (the message originator)
17	This element will contain all parameters needed to identify the originator identity
18 - 20	This is the (source) resource identifier which correctly identify a specific WS-Resource (instance)
21	<wsx:Metadata> is the top level container element of received metadata. It is composed by several sections (one for each requested dialect)
22	Metadata section for WSDL
23 - 24	Web Service Description Language (WSDL)

RMD Dialect

The following illustrates an example response of a GetMetadata request with RMD dialect.

For <Header> section information please refer to the previous section.

For simplicity we will report a metadata descriptor for a resource that has only one properties.

```

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
01.   <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
02.     http://www.w3.org/2005/08/addressing/role/anonymous
03.   </wsa:To>
04.   <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
05.     http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadataResponse
06.   </wsa:Action>
07.   <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
08.     uuid:980617c8-e3a0-ebf1-8f5a-2b43d3d6d416
09.   </wsa:MessageID>
10.   <wsa:RelatesTo RelationshipType="wsa:Reply" xmlns:wsa="http://www.w3.org/2005/08/addressing">
11.     uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
12.   </wsa:RelatesTo>
13.   <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
14.     <wsa:Address>
15.       http://localhost:8080/qman/services/QManWsResource
16.     </wsa:Address>
17.     <wsa:ReferenceParameters>
18.       <qman-wsa:ResourceId
           wsa:IsReferenceParameter="true"
           xmlns:qman-wsa=
           "http://amqp.apache.org/qpid/management/qman/addressing"
           xmlns:wsa="http://www.w3.org/2005/08/addressing">
19.         a3759467-bede-476d-8dde-169f1a652191
20.       </qman-wsa:ResourceId>
           </wsa:ReferenceParameters>
           </wsa:From>
         </soap:Header>
       <soap:Body>
21.       <wsx:Metadata xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex">
22.         <wsx:MetadataSection>
24.           <wsrmd:MetadataDescriptor
               interface="qman:QManWsResourcePortType"
               name="QManWsResourceMetadata"
               wsdlLocation="http://docs.oasis-open.org/wsrf/rmd-1
QManWsResource.wsdl"
               xmlns:qman="http://amqp.apache.org/qpid/management/qman"
               xmlns:wsrmd="http://docs.oasis-open.org/wsrf/rmd-1">
25.             <wsrmd:Property
26.               xmlns:qman="http://amqp.apache.org/qpid/management/qman">
27.                 modifiability="read-write"
28.                 mutability="mutable"
29.                 name="qman:operatingSystem"
29. 
30.               <wsrmd:ValidValues>
               <qman:operatingSystem>Linux</qman:operatingSystem>
               <qman:operatingSystem>Tru64</qman:operatingSystem>
               <qman:operatingSystem>HP-UX</qman:operatingSystem>
               <qman:operatingSystem>Windows XP</qman:operatingSystem>
             </wsrmd:ValidValues>
31. 
32.             <wsrmd:StaticValues>
...
...
33.             </wsrmd:StaticValues>
34. 
35.             <wsrmd:InitialValues>
...
...
</wsrmd:InitialValues>
</wsrmd:Property>
...
</wsrmd:MetadataDescriptor>
</wsx:MetadataSection>
</wsx:Metadata>
</soapBody>
</soap:Envelope>

```

Line(s)	Description
---------	-------------

24	Top level element for resource metadata descriptor. The most important things are its attributes. There you can see port type (interface), descriptor name (name) and wsdl location
25 - 29	Property metadata descriptor. This contains metadata information about a specific property. As you can see a property has several attributes : <ul style="list-style-type: none"> • mutability : The property has a constant (constant) value or it could change (mutable); • modifiability : Property access mode. Can be "read-only" or "read-write"; • name : the name of the property.
30 - 31	ValidValues are used to restrict the set of valid values that a property can assume
32 - 33	StaticValues are used to define a minimum set of values that a property can assume
34 - 35	InitialValues are used to declaratively define the set of values that a property will contain when the owner resource is initialized at first time

Faults

The only exception that could be thrown in a metadata exchange scenario is when the requestor indicates an unknown dialect.

Quick links

[WS-MetadataExchange](#)
[Web Services Addressing \(WS-Addressing\)](#)
[Web Service Description Language](#)
[Web Services Resource Metadata 1.0 \(WS-ResourceMetadataDescriptor\)](#)

OperationInvocation

Operation invocation on a WS-Resource

- [Description](#)
- [Return Type](#)
- [Request](#)
- [Response](#)
- [Faults](#)
- [Quick links](#)

Description

This interface allows a requestor to invoke an operation on a WS-Resource.

Let's say that there's a substantial difference with the other interfaces explained in this section. The name "Operation invocation" doesn't mean each WS-Resource

has an interface with this name. Remember, the WS-Resource interface definition is built at runtime, and therefore we don't know what will be the set of capabilities / operations / properties the resource will expose for management.

In this example we assume the resource has an operation called "echo" explained in detail below.

As part of that, we will provide information about

- How to invoke that operation;
- How to correctly specify / encode the input parameters;
- What is the expected return type on a basic scenario (without exceptions);
- What are possible faults that are returned as result of an exception / error scenario;

Generally speaking, an operation is mainly defined by its signature. That includes :

- Operation name : the name of the operation.
- Return Type : the return type of the operation.
- Parameters : The input parameters that will be used by operation in order to execute its task.
- Fault / Exceptions / Errors : thrown by the operation when an error occurs.

For QMan resources, consider these important points about operation signatures :

- Operations have always one and only one return type : Basically it is a value object containing an (optional) output parameters map. See section below for details.
- Operations have always the same set of faults in their signature. See section below for explanation of each fault.

Return Type

Return type is a simple "Value Object" that encapsulates the (optional) the output parameters map. Note that for example void methods won't contain that map.

The following is the schema of the mentioned return type. Obviously you can find it in each WS-Resource WSDL, too.

```

<!-- Invocation Result -->
<xsd:complexType name="result">
  <xsd:sequence>
    <xsd:element name="outputParameters" type="qman:map" />
  </xsd:sequence>
</xsd:complexType>

<!-- Output parameter map -->
<xsd:complexType name="map">
  <xsd:sequence>
    <xsd:element name="entry" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <!-- Entry key is always a string. -->
          <xsd:element name="key" type="xsd:string"/>
          <!-- While entry value could be any arbitrary object. -->
          <xsd:element name="value" type="xsd:anyType"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

```

Request

For this example we suppose the WS-Resource has the following operation :

```

01. <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
02.   <soap:Header>
03.     <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
04.       http://localhost:8080/qman/services/QManWsResource
05.     </wsa:To>
06.     <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
07.       http://amqp.apache.org/qpid/management/qman/echoWithSimpleTypes
08.     </wsa:Action>
09.     <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
10.       uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
11.     </wsa:MessageID>
12.     <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
13.       <wsa:Address>
14.         http://www.w3.org/2005/08/addressing/role/anonymous
15.       </wsa:Address>
16.     </wsa:From>
17.     <qman:ResourceId
      xmlns:wsa="http://www.w3.org/2005/08/addressing"
      wsa:IsReferenceParameter="true"
      xmlns:qman="http://amqp.apache.org/qpid/management/qman">
18.       781f4ad7-4c96-4caa-b69d-291461cdb1fc
19.     </qman:ResourceId>
    </soap:Header>
  <soap:Body xmlns:qman="http://amqp.apache.org/qpid/management/qman">
20.    <qman:echoWithSimpleTypesRequest>
21.      <qman:p1>1373</qman:p1>
22.      <qman:p2>true</qman:p2>
23.      <qman:p3>12763.44</qman:p3>
24.      <qman:p4>2727.233</qman:p4>
25.      <qman:p5>28292</qman:p5>
26.      <qman:p6>227</qman:p6>
27.      <qman:p7>expectedStringResult</qman:p7>
28.      <qman:p8>http://qpid.apache.org/</qman:p8>
29.      <qman:p9>1235061886761</qman:p9>
30.      <qman:p9 xsi:type="xsd:long">
        1235061886761
      </qman:p9>
    </qman:echoWithSimpleTypesRequest>
  </soap:Body>
</soap:Envelope>

```

Line(s)	Description
01	The SOAP <Envelope> is the root element in every SOAP message, and contains two child elements, <Header> and <Body>.

02	The SOAP Header will contain all metadata used for identifying the conversation participants (requestor and provider).
03 - 05	Convey the target endpoint also known (in the request phase) as service provider.
06 - 08	The action usually corresponds to the name of the operation. Generally speaking it is composed by the QMan namespace followed by the name of the operation.
09 - 11	Convey a unique identifier associated with the current message. This will be used for request / response messages correlation.
12 - 15	Provide the address of the source endpoint also known (in the request phase) as service requestor.
17 - 19	This indicates the target resource of this request. Specifically the line 18 contains the resource identifier.
20 - 31	The operation invocation request. Note that parameters have no type specified (except for the line 30). This is because the corresponding element is explicitly declared on resource WSDL. Line 30 (qman 😊 9 parameter) has an inlined type declaration using the xsi:type attribute because in the WSDL its declared type is xsd:anyType so we can't know in advance what it really is.

Response

```

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Header>
01.     <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
02.         http://www.w3.org/2005/08/addressing/role/anonymous
03.     </wsa:To>
04.     <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
05.         http://amqp.apache.org/qpid/management/qman/echoWithSimpleTypesResponse
06.     </wsa:Action>
07.     <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
08.         uuid:980617c8-e3a0-ebf1-8f5a-2b43d3d6d416
09.     </wsa:MessageID>
10.     <wsa:RelatesTo RelationshipType="wsa:Reply" xmlns:wsa="http://www.w3.org/2005/08/addressing">
>
11.         uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
12.     </wsa:RelatesTo>
13.     <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
14.         <wsa:Address>
15.             http://localhost:8080/qman/services/QManWsResource
16.         </wsa:Address>
17.         <wsa:ReferenceParameters>
<qman:ResourceId
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    wsa:IsReferenceParameter="true"
    xmlns:qman="http://amqp.apache.org/qpid/management/qman">
18.             781f4ad7-4c96-4caa-b69d-291461cdb1fc
19.         </qman:ResourceId>
20.     </wsa:ReferenceParameters>
    </wsa:From>
</soap:Header>
<soap:Body>
21.     <qman:echoWithSimpleTypesResponse xmlns:qman="http://amqp.apache.org/qpid/management/qman">
        <qman:echoWithSimpleTypesResponse>
22.         <outputParameters xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
23.             <qman:p1>1373</qman:p1>
24.             <qman:p2>true</qman:p2>
25.             <qman:p3>12763.44</qman:p3>
26.             <qman:p4>2727.233</qman:p4>
27.             <qman:p5>28292</qman:p5>
28.             <qman:p6>227</qman:p6>
29.             <qman:p7>expectedStringResult</qman:p7>
30.             <qman:p8>http://qpid.apache.org/</qman:p8>
31.             <qman:p9>1235061886761</qman:p9>
32.             <qman:p9 xsi:type="xsd:long">
33.                 1235061886761
34.             </qman:p9>
35.         </outputParameters>
36.         <qman:echoWithSimpleTypesResponse>
37.     </qman:echoWithSimpleTypesResponse>
    </soapBody>
</soap:Envelope>

```

Line(s)	Description
---------	-------------

01 - 03	Convey the recipient of the response message. Note that this time we are talking about the service requestor; The address matches the <wsa:From> previously found in the corresponding request.
04 - 06	Indicate this is a echoWithSimpleType response. This is done as usual using a wsa:Action that is part of WS-Addressing specification.
07 - 09	Convey a unique identifier associated with the current response message.
10 - 12	This element provides the identifier of the correlated (request) message.
13	The <wsa:From> element (part of WS-Addressing specs too) identifies the source endpoint, the originator of this response message.
14 - 16	This is the address of the source service endpoint. As said for lines 01-03 this time this is referred to service provider (the message originator).
17 - 20	As part of wsa:From element, this contains (specifically on line 18) additional information needed for identifying the originator of this message.
21 - 31	This is the GetMultipleResourceProperties response element which contains the requested property values as nested child.
22 - 37	This is the response body. Specifically the operation in the example is a simple "echo" operation, and therefore all input parameters are sent back as output section.

Faults

- ResourceUnknownFault : There's no resource on QMan associated with the given reference information (soap address and identifier).
- ResourceUnavailableFault : The requested resource is unavailable. This fault should indicate a transient condition. That means a requester might resend the message.
- InvalidResourcePropertyQNameFault : One or more of the names (QNames) in the request message doesn't correspond to a property element of the target WS-Resource.
- OperationInvocationFault : The operation invocation failed.

Quick links

[Web Services Resource](#)

PauseSubscription

Pause Subscription

- Description
- Request
- Response
- Faults
- Quick links

Description

This interface allows a requestor to temporarily suspend an existing subscription. After successful processing the pause request the subscription is in the paused state. Production of further notifications can be resumed using a ResumeSubscription request.
In order to be able to send a pause subscription request, the consumer must have a valid subscription reference like this :

```
<wsnt:SubscribeResponse xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
  <wsnt:SubscriptionReference>
    <wsa:Address xmlns:wsa="http://www.w3.org/2005/08/addressing">
      http://localhost:8080/qman/services/SubscriptionManager
    </wsa:Address>
    <wsa:ReferenceParameters xmlns:wsa="http://www.w3.org/2005/08/addressing">
      <qman-wsa:ResourceId>
        282f28e6-4396-4000-a19d-87a03978e8a0
      </qman-wsa:ResourceId>
      <wsa:ReferenceParameters>
        </wsnt:SubscriptionReference>
        <wsnt:CurrentTime>2009-02-27T13:51:56+01:00</wsnt:CurrentTime>
    </wsnt:SubscribeResponse>
```



A pause request has no effect on an already resumed subscription.

Request

```

01. <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
02.   <soap:Header>
03.     <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
04.       http://localhost:8080/qman/services/SubscriptionManager
05.     </wsa:To>
06.       <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
07.         http://docs.oasis-open.org/wsn/bw-2/SubscriptionManager/PauseSubscriptionRequest
08.       </wsa:Action>
09.       <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
10.         uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
11.       </wsa:MessageID>
12.       <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
13.         <wsa:Address>
14.           http://www.w3.org/2005/08/addressing/role/anonymous
15.         </wsa:Address>
16.       </wsa:From>
17.       <qman-wsa:ResourceId
      xmlns:wsa="http://www.w3.org/2005/08/addressing"
      wsa:IsReferenceParameter="true"
      xmlns:qman-wsa="http://amqp.apache.org/qpid/management/qman/addressing">
18.         282f28e6-4396-4000-a19d-87a03978e8a0
19.       </qman-wsa:ResourceId>
</soap:Header>
<soap:Body xmlns:qman="http://amqp.apache.org/qpid/management/qman">
20.   <wsnt:PauseSubscription xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"/>
</soap:Body>
</soap:Envelope>

```

Line(s)	Description
01	The SOAP <Envelope> is the root element in every SOAP message, and contains two child elements, <Header> and <Body>.
02	The SOAP Header will contain all metadata used for identifying the conversation participants (requestor and provider).
03 - 05	Convey the target endpoint also known (in the request phase) as service provider. In this case that's the Subscription Manager WS-Resource.
06 - 08	Indicate this is a PauseSubscription request.
09 - 11	Convey a unique identifier associated with the current message. This will be used for request / response messages correlation.
12 - 15	Provide the address of the source endpoint also known (in the request phase) as service requestor.
17 - 19	This indicates the target resource (subscription) of this request. Specifically the line 18 contains the subscription identifier previously mentioned.
20	The pause subscription body. As you can see there are no parameters for this kind of request.

Response

```

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
01.   <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
02.     http://www.w3.org/2005/08/addressing/role/anonymous
03.   </wsa:To>
04.   <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
05.     http://docs.oasis-open.org/wsn/bw-2/SubscriptionManager/PauseSubscriptionResponse
06.   </wsa:Action>
07.   <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
08.     uuid:980617c8-e3a0-ebf1-8f5a-2b43d3d6d416
09.   </wsa:MessageID>
10.   <wsa:RelatesTo RelationshipType="wsa:Reply" xmlns:wsa="http://www.w3.org/2005/08/addressing"
>
11.     uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
12.   </wsa:RelatesTo>
13.   <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
14.     <wsa:Address>
15.       http://localhost:8080/qman/services/SubscriptionManager
16.     </wsa:Address>
17.     <wsa:ReferenceParameters>
      <qman-wsa:ResourceId
        xmlns:wsa="http://www.w3.org/2005/08/addressing"
        wsa:IsReferenceParameter="true"
        xmlns:qman-wsa="http://amqp.apache.org/qpid/management/qman/addressing">
18.       282f28e6-4396-4000-a19d-87a03978e8a0
19.     </qman-wsa:ResourceId>
20.   </wsa:ReferenceParameters>
    </wsa:From>
  </soap:Header>
  <soap:Body>
21.   <wsnt:PauseSubscriptionResponse xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"/>
  </soapBody>
</soap:Envelope>

```

Line(s)	Description
01 - 03	Convey the recipient of the response message. Note that this time we are talking about the service requestor; The address matches the <wsa:From> previously found in the corresponding request.
04 - 06	Indicate this is a PauseSubscription response.
07 - 09	Convey a unique identifier associated with the current response message.
10 - 12	This element provides the identifier of the correlated (request) message.
13	The <wsa:From> element (part of WS-Addressing specs too) identifies the source endpoint, the originator of this response message.
14 - 16	This is the address of the source service endpoint. As said for lines 01-03 this time this is referred to service provider (the message originator).
17 - 20	As part of wsa:From element, this contains (specifically on line 18) additional information needed for identifying the originator of this message.
21	This is the response body indicating that the subscription has been successfully suspended.

Faults

- **ResourceUnknownFault** : There's no resource on QMan associated with the given reference information (soap address and identifier).
- **ResourceUnavailableFault** : The requested resource is unavailable. This fault should indicate a transient condition. That means a requester might resend the message.
- **PauseFailedFault** : The subscription has not been suspended.

Quick links

[Web Services Resource](#)
[Web Services Base Notification](#)

PutResourcePropertyDocument

PutResourcePropertyDocument

- Description
- Request
- Response
- Faults
- Quick links

Description

This interface allows to completely / partially replace the resource property document of a WS-Resource. So briefly, it allows requestor to change resource internal state using the resource property document. This is the main difference between this interface and the `SetResourceProperties` : it operates the change(s) directly on the resource property document.

Note for response message (extract from WS-ResourceProperties specification) :

"If, after processing the `PutResourcePropertyDocument` request, the XML Infoset of the WS-Resource's resource properties document is identical to the XML Infoset of the contents of the `PutResourcePropertyDocument` request itself, then the contents of the `PutResourcePropertyDocumentResponse` MUST be empty.

If, after processing the `PutResourcePropertyDocument` request, the XML Infoset of the WS-Resource's resource properties document is not identical to the XML Infoset of the contents of the `PutResourcePropertyDocument` request itself, then the contents of the `PutResourcePropertyDocumentResponse` MUST contain the updated resource property document. If an implementation cannot return all of the resource property values associated with the request, due to, for example, security considerations, then it MUST fault."

Briefly, that means that if the request message contains the whole state of the target resource, and that whole state is successfully applied, then the response message will be empty.

If, the request message contains a subsection of the resource property document and this partial "state" is applied to the target resource, then the response message will return the new resource property document taht reflects the current resource state.

Request

```

01. <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
02.   <soap:Header>
03.     <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
04.       http://localhost:8080/qman/services/QManWsResource
05.     </wsa:To>
06.     <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
07.
http://docs.oasis-open.org/wsrf/rpw-2/PutResourcePropertyDocument/PutResourcePropertyDocumentRequest
</wsa:Action>
09.           <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
10.             uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
11.           </wsa:MessageID>
12.           <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
13.             <wsa:Address>
14.               http://www.w3.org/2005/08/addressing/role/anonymous
15.             </wsa:Address>
16.           </wsa:From>
17.           <qman:ResourceId
      xmlns:wsa="http://www.w3.org/2005/08/addressing"
      wsa:IsReferenceParameter="true"
      xmlns:qman="http://amqp.apache.org/qpid/management/qman">
18.             781f4ad7-4c96-4caa-b69d-291461cdb1fc
19.           </qman:ResourceId>
</soap:Header>
<soap:Body xmlns:qman="http://amqp.apache.org/qpid/management/qman">
21.   <wsrf-rp:PutResourcePropertyDocument xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2">
22.     <qman:QManWsResourceProperties>
23.       <qman:MgmtPubInterval>
24.         4321
25.       </qman:MgmtPubInterval>
26.     </qman:QManWsResourceProperties>
27.     <qman:Name>
28.       New Name
29.     </qman:Name>
30.   </wsrf-rp:PutResourcePropertyDocument>
</soap:Body>
</soap:Envelope>
```

Line(s)	Description
01	The SOAP <Envelope> is the root element in every SOAP message, and contains two child elements, <Header> and <Body>.
02	The SOAP Header will contain all metadata used for identifying the conversation participants (requestor and provider).

03 - 05	Convey the target endpoint also known (in the request phase) as service provider.
06 - 08	Indicate this is a PutResourcePropertyDocument request.
09 - 11	Convey a unique identifier associated with the current message. This will be used for request / response messages correlation.
12 - 15	Provide the address of the source endpoint also known (in the request phase) as service requestor.
17 - 19	This indicates the target resource of this request. Specifically the line 18 contains the resource identifier.
21 - 30	The PutResourcePropertyDocument request. That will contain the new resource property document (total or partial).
22 - 29	This is the new resource property document. In this example it contains only two properties : MgmtPubInterval (23 - 25) and Name (27 - 29).

Response

```

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
01.   <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
02.     http://www.w3.org/2005/08/addressing/role/anonymous
03.   </wsa:To>
04.   <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
05.     http://docs.oasis-open.org/wsrf/rpw-2/SetResourceProperties/SetResourcePropertiesResponse
06.   </wsa:Action>
07.   <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
08.     uuid:980617c8-e3a0-ebf1-8f5a-2b43d3d6d416
09.   </wsa:MessageID>
10.   <wsa:RelatesTo RelationshipType="wsa:Reply" xmlns:wsa="http://www.w3.org/2005/08/addressing"
>
11.     uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
12.   </wsa:RelatesTo>
13.   <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
14.     <wsa:Address>
15.       http://localhost:8080/qman/services/QManWsResource
16.     </wsa:Address>
17.     <wsa:ReferenceParameters>
<qman:ResourceId
  xmlns:wsa="http://www.w3.org/2005/08/addressing"
  wsa:IsReferenceParameter="true"
  xmlns:qman="http://amqp.apache.org/qpid/management/qman">
18.       781f4ad7-4c96-4caa-b69d-291461cdb1fc
19.     </qman:ResourceId>
20.   </wsa:ReferenceParameters>
  </wsa:From>
</soap:Header>
<soap:Body xmlns:qman="http://amqp.apache.org/qpid/management/qman">
21.   <wsrf-rp:PutResourcePropertyDocumentResponse xmlns:wsrf-rp=
"http://docs.oasis-open.org/wsrf/rp-2">
22.     <qman:MgmtPubInterval>4321</qman:MgmtPubInterval>
<wsrf-rp:QueryExpressionDialect xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2">
  http://www.w3.org/TR/1999/REC-xpath-19991116
</wsrf-rp:QueryExpressionDialect>
23.     <qman:Name>New Name</qman:Name>
<qman:MsgTotalEnqueues>9223372036854775797</qman:MsgTotalEnqueues>
<qman:Arguments
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<qman:entry>
  <qman:key>Key3</qman:key>
  <qman:value xsi:type="xsd:integer">2147483647</qman:value>
</qman:entry>
<qman:entry>
  <qman:key>Key4</qman:key>
  <qman:value xsi:type="xsd:float">3.4028235E38</qman:value>
</qman:entry>
<qman:entry>
  <qman:key>Key1</qman:key>
  <qman:value xsi:type="xsd:string">aStringValue</qman:value>
</qman:entry>
<qman:entry>
  <qman:key>Key2</qman:key>
  <qman:value xsi:type="xsd:long">-9223372036854775808</qman:value>
</qman:entry>
</qman:Arguments>
<qman:VhostRef>57ae7a6d-6f33-48dc-9548-82078591fb9c</qman:VhostRef>
<qman:Durable>true</qman:Durable>
24.     <qman:ConsumerCount>-2147483638</qman:ConsumerCount>
    </wsrf-rp:PutResourcePropertyDocumentResponse>
  </soapBody>
</soap:Envelope>

```

Line(s)	Description
01 - 03	Convey the recipient of the response message. Note that this time we are talking about the service requestor; The address matches the <wsa:From> previously found in the corresponding request.
04 - 06	Indicate this is a PutResourcePropertyResponse message. This is done as usual using a wsa:Action that is part of WS-Addressing specification.
07 - 09	Convey a unique identifier associated with the current response message.

10 - 12	This element provides the identifier of the correlated (request) message.
13	The <wsa:From> element (part of WS-Addressing specs too) identifies the source endpoint, the originator of this response message.
14 - 16	This is the address of the source service endpoint. As said for lines 01-03 this time this is referred to service provider (the message originator).
17 - 20	As part of wsa:From element, this contains (specifically on line 18) additional information needed for identifying the originator of this message.
21 - 24	The resource property response. The nested children are all property members of resource property document. Note that the value of MgMtPublInterval (22) and Name (23) have been updated.

Faults

- **ResourceUnknownFault** : There's no resource on QMan associated with the given reference information (soap address and identifier).
- **ResourceUnavailableFault** : The requested resource is unavailable. This fault should indicate a transient condition. That means a requester might resend the message.
- **InvalidResourcePropertyQNameFault** : The name (QName) in the request message doesn't correspond to a property element of the target WS-Resource.
- **UnableToPutResourcePropertyDocumentFault** : In case of a not well-known failure while processing / applying the request.

Quick links

[Web Services Resource](#)
[Web Services Resource Properties](#)

QueryResourceProperties

QueryResourceProperties

- Description
- Request
- Response
- Faults
- Quick links

Description

This interface allows a requestor to query the resource properties document of a managed resource using a query expression. The given expression is evaluated against the resource properties document of the target resource. Note that although this request allows to declare a dialect for the given expression, only the XPath 1.0 dialect is supported at the moment.

Request

```

01. <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
02.   <soap:Header>
03.     <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
04.       http://localhost:8080/qman/services/QManWsResource
05.     </wsa:To>
06.       <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
07.
http://docs.oasis-open.org/wsrp/rpw-2/QueryResourceProperties/QueryResourcePropertiesRequest
08.   </wsa:Action>
09.     <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
10.       uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
11.     </wsa:MessageID>
12.     <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
13.       <wsa:Address>
14.         http://www.w3.org/2005/08/addressing/role/anonymous
15.       </wsa:Address>
16.     </wsa:From>
17.     <qman:ResourceId
      xmlns:wsa="http://www.w3.org/2005/08/addressing"
      wsa:IsReferenceParameter="true"
      xmlns:qman="http://amqp.apache.org/qpid/management/qman">
18.       781f4ad7-4c96-4caa-b69d-291461cdb1fc
19.     </qman:ResourceId>
   </soap:Header>
   <soap:Body>
20.     <wsrf-rp:QueryResourceProperties>
21.       <wsrf-rp:QueryExpression Dialect="http://www.w3.org/TR/1999/REC-xpath-19991116" >
22.         boolean(/*/MgtPubInterval > 100 and /*/MsgTotalEnqueues > 56272)
23.       </wsrf-rp:QueryExpression>
24.     </wsrf-rp:QueryResourceProperties>
   </soap:Body>
</soap:Envelope>

```

Line(s)	Description
01	The SOAP <Envelope> is the root element in every SOAP message, and contains two child elements, <Header> and <Body>.
02	The SOAP Header will contain all metadata used for identifying the conversation participants (requestor and provider).
03 - 05	Convey the target endpoint also known (in the request phase) as service provider.
06 - 08	Indicate this is a QueryResourceProperties request.
09 - 11	Convey a unique identifier associated with the current message. This will be used for request / response messages correlation.
12 - 15	Provide the address of the source endpoint also known (in the request phase) as service requestor.
17 - 19	This indicates the target resource of this request. Specifically the line 18 contains the resource identifier.
20 - 22	The QueryResourceProperties request.
22	The XPath expression that will be evaluated against the resource properties document of the target resource. In this example we want to know if the property "MgmtPubInterval" is greater than 100 and the property MsgTotalEnqueues is greater than 56272.

Response

```

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
01.    <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
02.      http://www.w3.org/2005/08/addressing/role/anonymous
03.    </wsa:To>
04.    <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
05.
http://docs.oasis-open.org/wsrf/rpw-2/QueryResourceProperties/QueryResourcePropertiesResponse
06.    </wsa:Action>
07.    <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
08.      uuid:980617c8-e3a0-ebf1-8f5a-2b43d3d6d416
09.    </wsa:MessageID>
10.    <wsa:RelatesTo RelationshipType="wsa:Reply" xmlns:wsa="http://www.w3.org/2005/08/addressing">
>
11.      uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
12.    </wsa:RelatesTo>
13.    <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
14.      <wsa:Address>
15.        http://localhost:8080/qman/services/QManWsResource
16.      </wsa:Address>
17.      <wsa:ReferenceParameters>
<qman:ResourceId
  xmlns:wsa="http://www.w3.org/2005/08/addressing"
  wsa:IsReferenceParameter="true"
  xmlns:qman="http://amqp.apache.org/qpid/management/qman">
18.        781f4ad7-4c96-4caa-b69d-291461cdb1fc
19.      </qman:ResourceId>
20.      </wsa:ReferenceParameters>
    </wsa:From>
  </soap:Header>
  <soap:Body>
21.    <wsrf-rp:QueryResourcePropertiesResponse xmlns:wsrf-rp=
"http://docs.oasis-open.org/wsrf/rp-2">
22.      true
23.    </wsrf-rp:QueryResourcePropertiesResponse>
  </soapBody>
</soap:Envelope>

```

Line(s)	Description
01 - 03	Convey the recipient of the response message. Note that this time we are talking about the service requestor; The address matches the <wsa:From> previously found in the corresponding request.
04 - 06	Indicate this is a QueryResourceProperties response. This is done as usual using a wsa:Action that is part of WS-Addressing specification.
07 - 09	Convey a unique identifier associated with the current response message.
10 - 12	This element provides the identifier of the correlated (request) message.
13	The <wsa:From> element (part of WS-Addressing specs too) identifies the source endpoint, the originator of this response message.
14 - 16	This is the address of the source service endpoint. As said for lines 01-03 this time this is referred to service provider (the message originator).
17 - 20	As part of wsa:From element, this contains (specifically on line 18) additional information needed for identifying the originator of this message.
21 - 23	This is the QuertyResourceProperties response element which contains the result of the evaluated query.
22	This element represents the requested property. Note that the name of the element is the name of the property.

Faults

- **ResourceUnknownFault** : There's no resource on QMan associated with the given reference information (soap address and identifier).
- **ResourceUnavailableFault** : The requested resource is unavailable. This fault should indicate a transient condition. That means a requester might resend the message.
- **UnknownQueryExpressionDialectFault** : Remember that only XPath 1.0 is supported. This fault is thrown if another dialect is declared on request message.
- **InvalidQueryExpressionFault** : The given expression is not valid according to the corresponding dialect (XPath).
- **QueryEvaluationErrorFault** : The evaluation of the given expression thrown an exception.

Quick links

ResumeSubscription

Pause Subscription

- Description
- Request
- Response
- Faults
- Quick links

Description

This interface allows a requestor to resume an previously suspended subscription.

After successful processing resume request the subscription is no longer in the paused state.

In order to be able to send a resume subscription request, the consumer must have a valid subscription reference like this :

```
<wsnt:SubscribeResponse xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
    <wsnt:SubscriptionReference>
        <wsa:Address xmlns:wsa="http://www.w3.org/2005/08/addressing">
            http://localhost:8080/qman/services/SubscriptionManager
        </wsa:Address>
        <wsa:ReferenceParameters xmlns:wsa="http://www.w3.org/2005/08/addressing">
            <qman-wsa:ResourceId>
                282f28e6-4396-4000-a19d-87a03978e8a0
            </qman-wsa:ResourceId>
            <wsa:ReferenceParameters>
        </wsnt:SubscriptionReference>
        <wsnt:CurrentTime>2009-02-27T13:51:56+01:00</wsnt:CurrentTime>
    </wsnt:SubscribeResponse>
```



A resume request has no effect on an already resumed subscription.

Request

```
01. <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
02.   <soap:Header>
03.     <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
04.       http://localhost:8080/qman/services/SubscriptionManager
05.     </wsa:To>
06.     <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
07.       http://docs.oasis-open.org/wsn/bw-2/SubscriptionManager/ResumeSubscriptionRequest
08.     </wsa:Action>
09.     <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
10.       uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
11.     </wsa:MessageID>
12.     <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
13.       <wsa:Address>
14.         http://www.w3.org/2005/08/addressing/role/anonymous
15.       </wsa:Address>
16.     </wsa:From>
17.     <qman-wsa:ResourceId
18.       xmlns:wsa="http://www.w3.org/2005/08/addressing"
19.       wsa:IsReferenceParameter="true"
20.       xmlns:qman-wsa="http://amqp.apache.org/qpid/management/qman/addressing">
21.       282f28e6-4396-4000-a19d-87a03978e8a0
22.     </qman-wsa:ResourceId>
23.   </soap:Header>
24.   <soap:Body xmlns:qman="http://amqp.apache.org/qpid/management/qman">
25.     <wsnt:ResumeSubscription xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"/>
26.   </soap:Body>
27. </soap:Envelope>
```

Line(s)	Description
---------	-------------

01	The SOAP <Envelope> is the root element in every SOAP message, and contains two child elements, <Header> and <Body>.
02	The SOAP Header will contain all metadata used for identifying the conversation participants (requestor and provider).
03 - 05	Convey the target endpoint also known (in the request phase) as service provider. In this case that's the Subscription Manager WS-Resource.
06 - 08	Indicate this is a ResumeSubscription request.
09 - 11	Convey a unique identifier associated with the current message. This will be used for request / response messages correlation.
12 - 15	Provide the address of the source endpoint also known (in the request phase) as service requestor.
17 - 19	This indicates the target resource (subscription) of this request. Specifically the line 18 contains the subscription identifier previously mentioned.
20	The resume subscription body. As you can see there are no parameters for this kind of request.

Response

```

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
    01.   <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
    02.     http://www.w3.org/2005/08/addressing/role/anonymous
    03.   </wsa:To>
    04.   <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
    05.     http://docs.oasis-open.org/wsn/bw-2/SubscriptionManager/ResumeSubscriptionResponse
    06.   </wsa:Action>
    07.   <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
    08.     uuid:980617c8-e3a0-ebf1-8f5a-2b43d3d6d416
    09.   </wsa:MessageID>
    10.   <wsa:RelatesTo RelationshipType="wsa:Reply" xmlns:wsa="http://www.w3.org/2005/08/addressing">
    >
    11.     uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
    12.   </wsa:RelatesTo>
    13.   <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
    14.     <wsa:Address>
    15.       http://localhost:8080/qman/services/SubscriptionManager
    16.     </wsa:Address>
    17.     <wsa:ReferenceParameters>
      <qman-wsa:ResourceId
        xmlns:wsa="http://www.w3.org/2005/08/addressing"
        wsa:IsReferenceParameter="true"
        xmlns:qman-wsa="http://amqp.apache.org/qpid/management/qman/addressing">
    18.       282f28e6-4396-4000-a19d-87a03978e8a0
    19.     </qman-wsa:ResourceId>
    20.   </wsa:ReferenceParameters>
    </wsa:From>
  </soap:Header>
  <soap:Body>
    21.   <wsnt:ResumeSubscriptionResponse xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2" />
  </soapBody>
</soap:Envelope>

```

Line(s)	Description
01 - 03	Convey the recipient of the response message. Note that this time we are talking about the service requestor; The address matches the <wsa:From> previously found in the corresponding request.
04 - 06	Indicate this is a ResumeSubscription response.
07 - 09	Convey a unique identifier associated with the current response message.
10 - 12	This element provides the identifier of the correlated (request) message.
13	The <wsa:From> element (part of WS-Addressing specs too) identifies the source endpoint, the originator of this response message.
14 - 16	This is the address of the source service endpoint. As said for lines 01-03 this time this is referred to service provider (the message originator).
17 - 20	As part of wsa:From element, this contains (specifically on line 18) additional information needed for identifying the originator of this message.
21	This is the response body indicating that the subscription has been successfully resumed.

Faults

- **ResourceUnknownFault** : There's no resource on QMan associated with the given reference information (soap address and identifier).
- **ResourceUnavailableFault** : The requested resource is unavailable. This fault should indicate a transient condition. That means a requester might resend the message.
- **ResumeFailedFault** : The subscription has not been resumed.

Quick links

[Web Services Resource](#)
[Web Services Base Notification](#)

SetResourceProperties

SetResourceProperties

- Description
- Request
- Response
- Faults
- Quick links

Description

This interface allows a requestor to change the state of a WS-Resource, modifying the values of multiple resource properties.

There are two types of changes that can be done on a resource property :

- Insert: wherein a new resource property element is inserted into the resource properties document; before of that the property was null and therefore wasn't part of the resource property document;
- Update: wherein existing resource property element(s) are updated; that is, the property was already part of the resource property document;



In order to be fully WSRF compliant, there should be a third type of change : Delete. It will be implemented sooner but keep in mind that at the moment is not supported.

Request

```

01. <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
02.   <soap:Header>
03.     <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
04.       http://localhost:8080/qman/services/QManWsResource
05.     </wsa:To>
06.       <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
07.         http://docs.oasis-open.org/wsrf/rpw-2/SetResourceProperties/SetResourcePropertiesRequest
08.       </wsa:Action>
09.         <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
10.           uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
11.         </wsa:MessageID>
12.         <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
13.           <wsa:Address>
14.             http://www.w3.org/2005/08/addressing/role/anonymous
15.           </wsa:Address>
16.         </wsa:From>
17.         <qman:ResourceId
18.           xmlns:wsa="http://www.w3.org/2005/08/addressing"
19.           wsa:IsReferenceParameter="true"
20.           xmlns:qman="http://amqp.apache.org/qpid/management/qman">
21.             781f4ad7-4c96-4caa-b69d-291461cdb1fc
22.           </qman:ResourceId>
23.         </soap:Header>
24.         <soap:Body xmlns:qman="http://amqp.apache.org/qpid/management/qman">
25.           <wsrf-rp:SetResourceProperties xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2">
26.             <wsrf-rp:Insert>
27.               <qman>Type>
28.                 This is a value for a string property.
29.               </qman>Type>
30.             </wsrf-rp:Insert>
31.             <wsrf-rp:Update>
32.               <qman:MgmtPubInterval>
33.                 12
34.               </qman:MgmtPubInterval>
35.             </wsrf-rp:Update>
36.           </wsrf-rp:SetResourceProperties>
37.         </soap:Body>
38.       </soap:Envelope>

```

Line(s)	Description
01	The SOAP <Envelope> is the root element in every SOAP message, and contains two child elements, <Header> and <Body>.
02	The SOAP Header will contain all metadata used for identifying the conversation participants (requestor and provider).
03 - 05	Convey the target endpoint also known (in the request phase) as service provider.
06 - 08	Indicate this is a SetResourceProperties request.
09 - 11	Convey a unique identifier associated with the current message. This will be used for request / response messages correlation.
12 - 15	Provide the address of the source endpoint also known (in the request phase) as service requestor.
17 - 19	This indicates the target resource of this request. Specifically the line 18 contains the resource identifier.
20 - 22	The SetResourceProperties request. Change types are specified using nested children.
21 - 23	This is an Insert change type. The property "Type" (specified using the nested child) is null on the target resource and therefore is not yet part of the resource property document. After that request, that property will be inserted on the property document and will have a value of "This is a value for a string property."
24 - 25	This is an Update change type. The property "MgmtPubInterval" is not null on the target resource and therefore is already part of its property document. After the request will be processed, that property will have a value of 12.

Response

```

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
01.   <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
02.     http://www.w3.org/2005/08/addressing/role/anonymous
03.   </wsa:To>
04.   <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
05.     http://docs.oasis-open.org/wsrf/rpw-2/SetResourceProperties/SetResourcePropertiesResponse
06.   </wsa:Action>
07.   <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
08.     uuid:980617c8-e3a0-ebf1-8f5a-2b43d3d6d416
09.   </wsa:MessageID>
10.   <wsa:RelatesTo RelationshipType="wsa:Reply" xmlns:wsa="http://www.w3.org/2005/08/addressing">
>
11.     uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
12.   </wsa:RelatesTo>
13.   <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
14.     <wsa:Address>
15.       http://localhost:8080/qman/services/QManWsResource
16.     </wsa:Address>
17.     <wsa:ReferenceParameters>
      <qman:ResourceId
        xmlns:wsa="http://www.w3.org/2005/08/addressing"
        wsa:IsReferenceParameter="true"
        xmlns:qman="http://amqp.apache.org/qpid/management/qman">
18.       781f4ad7-4c96-4caa-b69d-291461cdb1fc
19.     </qman:ResourceId>
20.   </wsa:ReferenceParameters>
    </wsa:From>
  </soap:Header>
  <soap:Body>
21.   <wsrf-rp:SetResourcePropertyResponse xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"/>
  </soapBody>
</soap:Envelope>

```

Line(s)	Description
01 - 03	Convey the recipient of the response message. Note that this time we are talking about the service requestor; The address matches the <wsa:From> previously found in the corresponding request.
04 - 06	Indicate this is a SetResourceProperties response. This is done as usual using a wsa:Action that is part of WS-Addressing specification.
07 - 09	Convey a unique identifier associated with the current response message.
10 - 12	This element provides the identifier of the correlated (request) message.
13	The <wsa:From> element (part of WS-Addressing specs too) identifies the source endpoint, the originator of this response message.
14 - 16	This is the address of the source service endpoint. As said for lines 01-03 this time this is referred to service provider (the message originator).
17 - 20	As part of wsa:From element, this contains (specifically on line 18) additional information needed for identifying the originator of this message.
21 - 25	This is the SetResourceProperties response element.

Faults

- **ResourceUnknownFault** : There's no resource on QMan associated with the given reference information (soap address and identifier).
- **ResourceUnavailableFault** : The requested resource is unavailable. This fault should indicate a transient condition. That means a requester might resend the message.
- **InvalidResourcePropertyQNameFault** : The name (QName) in the request message doesn't correspond to a property element of the target WS-Resource.
- **SetResourcePropertyRequestFailedFault** : Service provider was unable to satisfy the SetResourceProperties request.
- **InvalidModificationFault** : The content of the SetResourceProperties request cause the resource properties document to no longer be able to validate.
- **UnableToModifyResourcePropertyFault** : One or more properties contained in the SetResourceProperties request are read-only.

Quick links

Web Services Resource
Web Services Resource Properties

Subscribe

Subscribe

- Description
- Request
- Response
- Faults
- Quick links

Description

This interface allows a requestor to register itself as a listener of one or more QMan topics.

Request

Depending on filter and termination time we could have different combination of subscribe requests.

In general let's say we could specify the following :

- a topic filter : contains a name of a valid and existent topic.
In this case the subscriber is interested to receive only messages that are published on a given topic;
If there's no such filter the subscriber is supposed to be interested on all topics;
- a message filter : contains an expression that is evaluated against the current message.
Only if the evaluation of the given expression returns true this filter allows the message delivery.
- a producer properties filter : contains an expression that is evaluated against the resource properties document of the producer.
Only if the evaluation of the given expression returns true this filter allows the message delivery.
- a termination time : allows a requestor to specify a termination time of the Subscription being created.
If it is not present the subscription will never expire.



- Filters are not mandatory. That means if you omit lines 23 - 34 a subscription will be created for all messages / all topics without expiration date.
- Filters are processed in AND mode. That means if one of them fails, the message won't be delivered.
- Note that only XPath is supported as Dialect for filter expressions.

In the example reported below QMan (acting as a consumer) is running on localhost:8080 while the consumer is an endpoint service located on consumer.host.name:8726.

```

01. <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
02.   <soap:Header>
03.     <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
04.       http://localhost:8080/qman/services/adapter
05.     </wsa:To>
06.       <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
07.         http://docs.oasis-open.org/wsn/bw-2/NotificationProducer/SubscribeRequest
08.       </wsa:Action>
09.       <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
10.         uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
11.       </wsa:MessageID>
12.       <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
13.         <wsa:Address>
14.           http://www.w3.org/2005/08/addressing/role/anonymous
15.         </wsa:Address>
16.       </wsa:From>
17.     </soap:Header>
18.     <soap:Body xmlns:qman="http://amqp.apache.org/qpid/management/qman">
19.       <wsnt:Subscribe xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
20.         <wsnt:ConsumerReference>
21.           <wsa:Address xmlns:wsa="http://www.w3.org/2005/08/addressing">
22.             http://consumer.host.name:8726/qman/services/consumer
23.           </wsa:Address>
24.         </wsnt:ConsumerReference>
25.         <wsnt:Filter>
26.           <wsnt:TopicExpression Dialect=
27.             "http://docs.oasis-open.org/wsn/t-1/TopicExpression/Concrete">
28.               qman:EventsLifeCycleTopic
29.             </wsnt:TopicExpression>
30.             <wsnt:MessageContent Dialect="http://www.w3.org/TR/1999/REC-xpath-19991116">
31.               /NotificationMessage/Message/LifeCycleEvent/Resource/Name/text()='connection'
32.             </wsnt:MessageContent>
33.           </wsnt:Filter>
34.           <wsnt:TerminationTime>2009-02-20T13:29:24+01:00</wsnt:TerminationTime>
35.         </wsnt:Subscribe>
36.       </soap:Body>
37.     </soap:Envelope>

```

Line(s)	Description
01	The SOAP <Envelope> is the root element in every SOAP message, and contains two child elements, <Header> and <Body>.
02	The SOAP Header will contain all metadata used for identifying the conversation participants (requestor and provider).
03 - 05	Convey the target endpoint also known (in the request phase) as service provider.
06 - 08	Indicate this is a Subscribe request.
09 - 11	Convey a unique identifier associated with the current message. This will be used for request / response messages correlation.
12 - 15	Provide the address of the source endpoint also known (in the request phase) as service requestor.
17 - 23	This is the Subscribe request.
19 - 21	This element contains the address (line 20) of the consumer service endpoint. After this request has been processed it will be referred as Subscriber.
23 - 33	This is the most interesting part of the message : filters.
24 - 26	Topic filter : the consumer wants subscribe only the messages that are published on qman:EventsLifeCycleTopic (line 25) topic.
27 - 29	Message filter : the consumer wants receive notification only when the message is referred to a resource with "connection" as name (28). Basically it wants to monitor connections on broker.
30 - 32	Producer RSP filter : the consumer wants receive notification only when the producer has a property named MgmtPublInterval that is greater than 100.

Response

```

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
01.    <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
02.      http://www.w3.org/2005/08/addressing/role/anonymous
03.    </wsa:To>
04.    <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
05.      http://docs.oasis-open.org/wsn/bw-2/NotificationProducer/SubscribeResponse
06.    </wsa:Action>
07.    <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
08.      uuid:980617c8-e3a0-ebf1-8f5a-2b43d3d6d416
09.    </wsa:MessageID>
10.    <wsa:RelatesTo RelationshipType="wsa:Reply" xmlns:wsa="http://www.w3.org/2005/08/addressing"
>
11.      uuid:0cdb5112-09e0-ac39-06ba-393843f06e42
12.    </wsa:RelatesTo>
13.    <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
14.      <wsa:Address>
15.        http://localhost:8080/qman/services/adapter
16.      </wsa:Address>
17.    </wsa:From>
18.  </soap:Header>
19.  <soap:Body>
20.    <wsnt:SubscribeResponse xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
21.      <wsnt:SubscriptionReference>
22.        <wsa:Address xmlns:wsa="http://www.w3.org/2005/08/addressing">
23.          http://localhost:8080/qman/services/SubscriptionManager
24.        </wsa:Address>
25.        <wsa:ReferenceParameters xmlns:wsa="http://www.w3.org/2005/08/addressing">
26.          <qman-wsa:ResourceId xmlns:qman-wsa=
27.            "http://amqp.apache.org/qpid/management/qman/addressing">
28.              e067f34f-e7e9-4cb2-b13c-185a7e0d1d16
29.            </qman-wsa:ResourceId>
30.          </wsa:ReferenceParameters>
31.        </wsnt:SubscriptionReference>
32.        <wsnt:CurrentTime>2009-02-10T20:41:07+01:00</wsnt:CurrentTime>
33.        <wsnt:TerminationTime>2009-02-10T20:41:26+01:00</wsnt:TerminationTime>
34.      </wsnt:SubscribeResponse>
35.    </soapBody>
36.  </soap:Envelope>

```

Line(s)	Description
01 - 03	Convey the recipient of the response message. Note that this time we are talking about the service requestor; The address matches the <wsa:From> previously found in the corresponding request.
04 - 06	Indicate this is a Subscribe response. This is done as usual using a wsa:Action that is part of WS-Addressing specification.
07 - 09	Convey a unique identifier associated with the current response message.
10 - 12	This element provides the identifier of the correlated (request) message.
13	The <wsa:From> element (part of WS-Addressing specs too) identifies the source endpoint, the originator of this response message.
14 - 16	This is the address of the source service endpoint. As said for lines 01-03 this time this is referred to service provider (the message originator).
17 - 30	This is the body of the subscribe response.
18 - 27	Conveys the details of the subscription that has been created.
19 - 24	A subscription, as any other WS-Resource, has an address (20) and a resource identifier (24).
28	The creation date of the subscription.
29	The expiration / termination time of the subscription.

Faults

- **ResourceUnknownFault** : There's no resource on QMan associated with the given reference information (soap address and identifier).
- **InvalidFilterFault** : The request contains a filter that is not supported.
- **TopicExpressionDialectUnknownFault** : The request contains a topic filter with an unknown / not supported dialect.
- **TopicNotSupportedFault** : The request contains a topic that is not supported.

- **InvalidTopicExpressionFault** : The request contains a topic filter with an invalid expression.
- **InvalidProducerPropertiesExpressionFault** : The request contains a producer properties filter with an invalid expression.
- **InvalidMessageContentExpressionFault** : The request contains a message properties filter with an invalid expression.
- **SubscribeCreationFailedFault** : The notification producer failed to process the subscribe request.

Quick links

[Web Services Base Notification 1.3](#)

Qpid ACLs

ACL Formats

The Qpid project has two ACL implementations. An initial version of ACLs was added to the Java Broker for M2.1 that uses XML configuration. For M4 a new format was designed to be implemented by both C++ and Java brokers. M4 release includes the initial C++ implementation and M5 is expected to include the Java implementation.

Specifications

The specifications for each of the ACL formats are linked here:

[v1 XML ACLs \(Java Broker Only\)](#)

[v2 All brokers](#)

User Guides

To aid users in defining their ACLs we have a user guide for each of the ACL formats.

[v1 XML ACLs \(Java Broker Only\)](#)

[v2 All brokers](#)

Qpid Interoperability Documentation

Qpid Interoperability Documentation

This page documents the various interoperable features of the Qpid clients.

SASL

Standard Mechanisms

SASL Mechanisms

This table lists the various SASL mechanisms that each component supports. The version listed shows when this functionality was added to the product.

Component	ANONYMOUS	CRAM-MD5	DIGEST-MD5	EXTERNAL	GSSAPI/Kerberos	PLAIN
C++ Broker	M3[1]	M3[1,2]			M3[1,2]	M1
C++ Client	M3[1]					M1
Java Broker		M1				M1
Java Client		M1				M1
.Net Client	M2	M2	M2	M2		M2
Python Client						?
Ruby Client						?

1: Support for these will be in M3 (currently available on trunk).

2: C++ Broker uses [Cyrus Sasl](#) which supports CRAM-MD5 and GSSAPI but these have not been tested yet

Custom Mechanisms

There have been some custom mechanisms added to our implementations.

Component	AMQPLAIN	CRAM-MD5-HASHED
C++ Broker		

C++ Client		
Java Broker	M1	M2
Java Client	M1	M2
.Net Client		
Python Client	M2	
Ruby Client	M2	

AMQPLAIN

CRAM-MD5-HASHED

The Java SASL implementations require that you have the password of the user to validate the incoming request. This then means that the user's password must be stored on disk. For this to be secure either the broker must encrypt the password file or the need for the password being stored must be removed.

The CRAM-MD5-HASHED SASL plugin removes the need for the plain text password to be stored on disk. The mechanism defers all functionality to the build in CRAM-MD5 module the only change is on the client side where it generates the hash of the password and uses that value as the password. This means that the Java Broker only need store the password hash on the file system. While a one way hash is not very secure compared to other forms of encryption in environments where the having the password in plain text is unacceptable this will provide an additional layer to protect the password. In particular this offers some protection where the same password may be shared amongst many systems. It offers no real extra protection against attacks on the broker (the secret is now the hash rather than the password).

SSL

SSL How to

1. C++ broker (M4 and up)
2. Java Client
3. .Net Client

C++ broker (M4 and up)

- You need to get a certificate signed by a CA, trusted by your client.
- If you require client authentication, the clients certificate needs to be signed by a CA trusted by the broker.
- Setting up the certificates for testing.
 - For testing purposes you could use the [following guide](#) to setup your certificates.
 - In summary you need to create a root CA and import it to the brokers certificate data base.
 - Create a certificate for the broker, sign it using the root CA and then import it into the brokers certificate data base.
- Load the acl module using --load-module or if loading more than one module, copy ssl.so to the location pointed by --module-dir

```
Ex if running from source. ./qpid --load-module /libs/ssl.so
```

- Specify the password file (a plain text file with the password), certificate database and the brokers certificate name using the following options

```
Ex ./qpid ... --ssl-cert-password-file ~/pfile --ssl-cert-db ~/server_db/ --ssl-cert-name localhost.localdomain
```

- If you require client authentication you need to add --ssl-require-client-authentication as a command line argument.
- Please note that the default port for SSL connections is 5671, unless specified by --ssl-port

Here is an example of a broker instance that requires SSL client side authentication

```
./qpid ./qpid --load-module /libs/ssl.so --ssl-cert-password-file ~/pfile --ssl-cert-db ~/server_db/ --ssl-cert-name localhost.localdomain --ssl-require-client-authentication
```

Java Client (M4 and up)

- This guide is for connecting with the Qpid c++ broker.

- Setting up the certificates for testing. In summary,
 - You need to import the trusted CA in your trust store and keystore
 - Generate keys for the certificate in your key store
 - Create a certificate request using the generated keys
 - Create a certificate using the request, signed by the trusted CA.
 - Import the signed certificate into your keystore.
- Pass the following JVM arguments to your client.

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

.Net Client (M4 and up)

- If the Qpid broker requires client authentication then you need to get a certificate signed by a CA, trusted by your client.

Use the connectSSL instead of the standard connect method of the client interface.

connectSSL signature is as follows:

```
public void connectSSL(String host, int port, String virtualHost, String username, String password, String serverName, String certPath, bool rejectUntrusted)
```

Where

- host: Host name on which a Qpid broker is deployed
- port: Qpid broker port
- virtualHost: Qpid virtual host name
- username: User Name
- password: Password
- serverName: Name of the SSL server
- certPath: Path to the X509 certificate to be used when the broker requires client authentication
- rejectUntrusted: If true connection will not be established if the broker is not trusted (the server certificate must be added in your truststore)

Python & Ruby Client (M4 and up)

Simply use amqps:// in the URL string as defined above

Starting a cluster

Running a Qpidd cluster

There are several pre-requisites to running a qpidd cluster:

Install and configure openais/corosync

Qpid clustering uses a multicast protocol provided by the corosync (formerly called openais) library. Install whichever is available on your OS. E.g. in fedora10: yum install corosync.

The configuration file is /etc/ais/openais.conf on openais, /etc/corosync.conf on early corosync versions and /etc/corosync/corosync.conf on recent corosync versions. You will need to edit the default file created when you installed

Here is an example, with places marked that you will change. (Below, I will describe how to change the file.)

```

# Please read the openais.conf.5 manual page

totem {
    version: 2
    secauth: off
    threads: 0
    interface {
        ringnumber: 0
        ## You must change this address ##
        bindnetaddr: 20.0.100.0
        mcastaddr: 226.94.32.36
        mcastport: 5405
    }
}

logging {
    debug: off
    timestamp: on
    to_file: yes
    logfile: /tmp/aisexec.log
}

amf {
    mode: disabled
}

```

You must sent the bindnetaddr entry in the configuration file to the network address of your network interface. This must be a real network interface, not the loopback address 127.0.0.1

You can find your network interface by running ifconfig. This will list the address and the mask, e.g.

```
inet addr:20.0.20.32 Bcast:20.0.20.255 Mask:255.255.255.0
```

The bindnetaddr is the logical AND of the inet addr and mask values, in the example above 20.0.20.0

Open your firewall

In the above example file, I use mcastport 5405.
This implies that your firewall must allow UDP
protocol over port 5405, or that you disable the firewall

Use the proper identity.

The qpid process must be started with the correct identity in order to use the corosync/openais library.

For openais and early corosync versions the installation of openAIS/corosync on your system will create a new group called "ais". The user that starts the qpid processes of the cluster must have "ais" as its effective group id. You can create a user specifically for this purpose with ais as the primary group, or a user that has ais as a secondary group can use "newgrp" to set the primary group to ais when running qpid.

For recent corosync versions you no longer need to set your group to "ais" but you do need to create a file in /etc/corosync/uidgid.d/ to allow access for whatever user/group ID you want to use. For example create /etc/corosync/uidgid.d/qpid th the contents:

```

uidgid {
    uid: qpid
    gid: qpid
}

```

Starting a Cluster

To be a member of a cluster you must pass the --cluster-name argument to qpid. This is the only required option to join a cluster, other options can be set as for a normal qpid.

For example to start a cluster of 3 brokers on the current host

Here is an example of starting a cluster of 3 members, all on the current host but with different ports and different log files:

```
qpidd -p5672 --cluster-name=MY_CLUSTER --log-output=cluster0.log -d --no-data-dir --auth=no  
qpidd -p5673 --cluster-name=MY_CLUSTER --log-output=cluster0.log -d --no-data-dir --auth=no  
qpidd -p5674 --cluster-name=MY_CLUSTER --log-output=cluster0.log -d --no-data-dir --auth=no
```

In a deployed system, cluster members will normally be on different hosts but for development its useful to be able to create a cluster on a single host.

SELinux conflicts

Developers will often start openais/corosync as a service like this:

```
service openais start
```

But will then will start a cluster-broker without using the service script like this:

```
/usr/sbin/qpidd --cluster-name my_cluster ...
```

If SELinux is in enforcing mode this may cause qpidd to hang due because of the different SELinux contexts.

There are 3 ways to resolve this:

- run both qpidd and openais/corosync as services.
- run both qpidd and openais/corosync as user processes.
- make selinux permissive:

To check what mode selinux is running:

```
# getenforce
```

To change the mode:

```
# setenforce permissive
```

Note that in a deployed system both openais/corosync and qpidd should be started as services, in which case there is no problem with SELinux running in enforcing mode.

Troubleshooting checklist.

If you have trouble starting your cluster, make sure that:

1. You have edited the correct openais/corosync configuration file and set bindnetaddr correctly
1. Your firewall allows UDP on the openais/corosync mcastport
2. Your effective group is "ais" (openais/old corosync) or you have created an appropriate ID file (new corosync)
3. Your firewall allows TCP on the ports used by qpidd.
4. If you're starting openais as a service but running qpidd directly, ensure selinux is in permissive mode

Use of Get()

Note that SubscriptionManager::get() will cause a synchronous interaction pattern with the broker where a local queue get() will not. See example below of comparing them

Using SubscriptionManager::get():

```

#include <qpid/client/Connection.h>
#include <qpid/client/Session.h>
#include <qpid/client/Message.h>
#include <qpid/client/SubscriptionManager.h>
#include <qpid/sys/Time.h>
#include <unistd.h>
#include <cstdlib>
#include <iostream>
using namespace qpid::client;
using namespace qpid::framing;
int main(int argc, char** argv) {
    const char* host = argc>1 ? argv[1] : "127.0.0.1";
    int port = argc>2 ? atoi(argv[2]) : 5672;
    Connection connection;
    try {
        connection.open(host, port);
        Session session = connection.newSession();
        SubscriptionManager subs(session);
        Message m;
        while (!subs.get(m, "test-queue", 1*qpid::sys::TIME_SEC)) {
            std::cout << "No message available" << std::endl;
        }
        std::cout << "Got: " << m.getData() << std::endl;
        session.close();
        connection.close();
        return 0;
    } catch(const std::exception& error) {
        std::cout << error.what() << std::endl;
    }
    return 1;
}

```

Using a LocalQueue directly:

```

#include <qpid/client/Connection.h>
#include <qpid/client/Session.h>
#include <qpid/client/Message.h>
#include <qpid/client/SubscriptionManager.h>
#include <qpid/sys/Time.h>
#include <unistd.h>
#include <cstdlib>
#include <iostream>
using namespace qpid::client;
using namespace qpid::framing;
int main(int argc, char** argv) {
    const char* host = argc>1 ? argv[1] : "127.0.0.1";
    int port = argc>2 ? atoi(argv[2]) : 5672;
    Connection connection;
    try {
        connection.open(host, port);
        Session session = connection.newSession();
        SubscriptionManager subs(session);
        LocalQueue incoming;
        subs.subscribe(incoming, "test-queue");
        Message m;
        while (!incoming.get(m, 1*qpid::sys::TIME_SEC)) {
            std::cout << "No message available" << std::endl;
        }
        std::cout << "Got: " << m.getData() << std::endl;
        session.close();
        connection.close();
        return 0;
    } catch(const std::exception& error) {
        std::cout << error.what() << std::endl;
    }
    return 1;
}

```

Using Broker Federation

Introduction

Please note: Whereas broker federation was introduced in the M3 milestone release, the discussion in this document is based on the richer capabilities of federation in the M4 release.

This document presents broker federation for the administrative user. For design and developer information, please see [Federation Design Note](#).

What Is Broker Federation?

The Qpid C++ messaging broker supports broker federation, a mechanism by which large messaging networks can be built using multiple brokers. Some scenarios in which federation is useful:

- Connecting disparate locations across a wide area network. In this case full connectivity across the enterprise can be achieved while keeping local message traffic isolated to a single location.
- Departmental brokers that have a policy which controls the flow of inter-departmental message traffic.
- Scaling of capacity for expensive broker operations. High-function exchanges like the XML exchange can be replicated to scale performance.
- Co-Resident brokers Some applications benefit from having a broker co-resident with the client. This is particularly true if the client produces data that must be delivered reliably but connectivity to the consumer(s) is non-reliable. In this case, a co-resident broker provides queueing and durability not available in the client alone.
- Bridging disjoint IP networks. Message brokers can be configured to allow message connectivity between networks where there is no IP connectivity. For example, an isolated, private IP network can have messaging connectivity to brokers in other outside IP networks.

The qpid-route Utility

The qpid-route command line utility is provided with the Qpid broker. This utility is used to configure federated networks of brokers and to view the status and topology of networks.

qpid-route accesses the managed brokers remotely. It does not need to be invoked from the same host on which the broker is running. If network connectivity permits, an entire enterprise can be configured from a single location.

In the following sections, federation concepts will be introduced and illustrated using qpid-route.

Links and Routes

Federation occurs when a **link** is established between two brokers and one or more **routes** are created within that link. A **link** is a transport level connection (tcp, rdma, ssl, etc.) initiated by one broker and accepted by another. The initiating broker assumes the role of **client** with regard to the connection. The accepting broker annotates the connection as being for federation but otherwise treats it as a normal client connection.

A **route** is associated with an AMQP session established over the link connection. There may be multiple routes sharing the same link. A route controls the flow of messages across the link between brokers. Routes always consist of a session and a subscription for consuming messages. Depending on the configuration, a route may have a private queue on the source broker with a binding to an exchange on that broker.

Routes are unidirectional. A single route provides for the flow of messages in one direction across a link. If bidirectional connectivity is required (and it almost always is), then a pair of routes must be created, one for each direction of message flow.

The qpid-route utility allows the administrator to configure and manage links and routes separately. However, when a route is created and a link does not already exist, qpid-route will automatically create the link. It is typically not necessary to create a link by itself. It is, however, useful to get a list of links and their connection status from a broker:

```
$ qpid-route link list localhost:10001

Host          Port      Transport Durable  State           Last Error
=====
localhost     10002    tcp        N         Operational
localhost     10003    tcp        N         Operational
localhost     10009    tcp        N         Waiting       Connection refused
```

The example above shows a **link list** query to the broker at "localhost:10001". In the example, this broker has three links to other brokers. Two are operational and the third is waiting to connect because there is not currently a broker listening at that address.

The Life Cycle of a Link

When a link is created on a broker, that broker attempts to establish a transport-level connection to the peer broker. If it fails to connect, it retries the connection at an increasing time interval. If the connection fails due to authentication failure, it will not continue to retry as administrative intervention is needed to fix the problem.

If an operational link is disconnected, the initiating broker will attempt to re-establish the connection with the same interval back-off.

The shortest retry-interval is 2 seconds and the longest is 64 seconds. Once enough consecutive retries have occurred that the interval has

grown to 64 seconds, the interval will then stay at 64 seconds.

Durable Links and Routes

If, when a link or a route is created using qpid-route, the --durable option is used, it shall be durable. This means that its life cycle shall span restarts of the broker. If the broker is shut down, when it is restarted, the link will be restored and will begin establishing connectivity.

A non-durable route can be created for a durable link but a durable route cannot be created for a non-durable link.

```
$ qpid-route dynamic add localhost:10003 localhost:10004 fed.topic
$ qpid-route dynamic add localhost:10003 localhost:10004 fed.topic2 --durable
Failed: Can't create a durable route on a non-durable link
```

In the above example, a transient (non-durable) dynamic route was created between localhost:10003 and localhost:10004. Because there was no link in place, a new transient link was created. The second command is attempting to create a durable route over the same link and is rejected as illegal.

Dynamic Routing

Dynamic routing provides the simplest configuration for a network of brokers. When configuring dynamic routing, the administrator need only express the logical topology of the network (i.e. which pairs of brokers are connected by a unidirectional route). Queue configuration and bindings are handled automatically by the brokers in the network.

Dynamic routing uses the **Distributed Exchange** concept. From the client's point of view, all of the brokers in the network collectively offer a single logical exchange that behaves the same as a single exchange in a single broker. Each client connects to its local broker and can bind its queues to the distributed exchange and publish messages to the exchange.

When a consuming client binds a queue to the distributed exchange, information about that binding is propagated to the other brokers in the network to ensure that any messages matching the binding will be forwarded to the client's local broker. Messages published to the distributed exchange are forwarded to other brokers only if there are remote consumers to receive the messages. The dynamic binding protocol ensures that messages are routed only to brokers with eligible consumers. This includes topologies where messages must make multiple hops to reach the consumer.

When creating a dynamic routing network, The type and name of the exchange must be the same on each broker. It is **strongly** recommended that dynamic routes **NOT** be created using the standard exchanges (that is unless all messaging is intended to be federated).

A simple, two-broker network can be configured by creating an exchange on each broker then a pair of dynamic routes (one for each direction of message flow):

Create exchanges:

```
$ qpid-config -a localhost:10003 add exchange topic fed.topic
$ qpid-config -a localhost:10004 add exchange topic fed.topic
```

Create dynamic routes:

```
$ qpid-route dynamic add localhost:10003 localhost:10004 fed.topic
$ qpid-route dynamic add localhost:10004 localhost:10003 fed.topic
```

Information about existing routes can be gotten by querying each broker individually:

```
$ qpid-route route list localhost:10003
localhost:10003 localhost:10004 fed.topic <dynamic>
$ qpid-route route list localhost:10004
localhost:10004 localhost:10003 fed.topic <dynamic>
```

A nicer way to view the topology is to use **qpid-route route map**. The argument to this command is a single broker that serves as an entry point. **qpid-route** will attempt to recursively find all of the brokers involved in federation relationships with the starting broker and map all of the routes it finds.

```
$ qpid-route route map localhost:10003

Finding Linked Brokers:
localhost:10003... Ok
localhost:10004... Ok

Dynamic Routes:

Exchange fed.topic:
localhost:10004 <=> localhost:10003

Static Routes:
none found
```

More extensive and realistic examples are supplied later in this document.

Static Routing

Dynamic routing provides simple, efficient, and automatic handling of the bindings that control routing as long as the configuration keeps within a set of constraints (i.e. exchanges of the same type and name, bidirectional traffic flow, etc.). However, there are scenarios where it is useful for the administrator to have a bit more control over the details. In these cases, static routing is appropriate.

Exchange Routes

An exchange route is like a dynamic route except that the exchange binding is statically set at creation time instead of dynamically tracking changes in the network.

When an exchange route is created, a private queue (auto-delete, exclusive) is declared on the source broker. The queue is bound to the indicated exchange with the indicated key and the destination broker subscribes to the queue with a destination of the indicated exchange. Since only one exchange name is supplied, this means that exchange routes require that the source and destination exchanges have the same name.

Static exchange routes are added and deleted using **qpid-route route add** and **qpid-route route del** respectively. The following example creates a static exchange route with a binding key of "global.#" on the default topic exchange:

```
$ qpid-route route add localhost:10001 localhost:10002 amq.topic global.#
```

The route can be viewed by querying the originating broker (the destination in this case, see discussion of push and pull routes for more on this):

```
$ qpid-route route list localhost:10001
localhost:10001 localhost:10002 amq.topic global.#
```

Alternatively, the **route map** feature can be used to view the topology:

```
$ qpid-route route map localhost:10001

Finding Linked Brokers:
localhost:10001... Ok
localhost:10002... Ok

Dynamic Routes:
none found

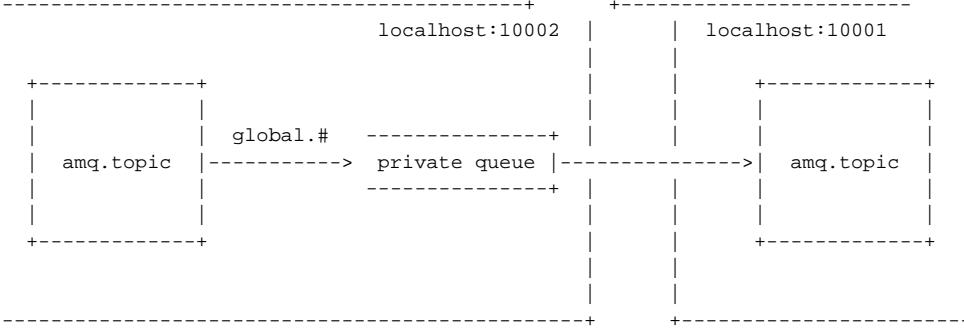
Static Routes:

localhost:10001(ex=amq.topic) <= localhost:10002(ex=amq.topic) key=global.#
```

This example causes messages delivered to the **amq.topic** exchange on broker **localhost:10002** that have a key that matches **global.#** (i.e. starts with the string "global.") to be delivered to the **amq.topic** exchange on broker **localhost:10001**. This delivery will occur regardless of whether there are any consumers on **localhost:10001** that will receive the messages.

Note that this is a uni-directional route. No messages will be forwarded in the opposite direction unless another static route is created in the other direction.

The following diagram illustrates the result, in terms of AMQP objects, of the example static exchange route. In this diagram, the exchanges, both named "amq.topic" exist prior to the creation of the route. The creation of the route causes the private queue, the binding, and the subscription of the queue to the destination to be created.



Queue Routes

A queue route causes the destination broker to create a subscription to a pre-existing, possibly shared, queue on the source broker. There's no requirement that the queue be bound to any particular exchange. Queue routes can be used to connect exchanges of different names and/or types. They can also be used to distribute or balance traffic across multiple destination brokers.

Queue routes are created and deleted using the **qpid-route queue add** and **qpид-route queue del** commands respectively. The following example creates a static queue route to a public queue called "public" that feeds the amq.fanout exchange on the destination:

Create a queue on the source broker:

```
$ qpid-config -a localhost:10002 add queue public
```

Create a queue route to the new queue

```
$ qpid-route queue add localhost:10001 localhost:10002 amq.fanout public
```

Pull vs. Push Routes

When qpid-route creates or deletes a route, it establishes a connection to one of the brokers involved in the route and configures that broker. The configured broker then takes it upon itself to contact the other broker and exchange whatever information is needed to complete the setup of the route.

The notion of **push** vs. **pull** is concerned with whether the configured broker is the source or the destination. The normal case is the pull route, where qpid-route configures the destination to pull messages from the source. A push route occurs when qpid-route configures the source to push messages to the destination.

Dynamic routes are always pull routes. Static routes are normally pull routes but may be inverted by using the `src-local` option when creating (or deleting) a route. If `src-local` is specified, qpid-route will make its connection to the source broker rather than the destination and configure the route to push rather than pull.

Push routes are useful in applications where brokers are co-resident with data sources and are configured to send data to a central broker. Rather than configure the central broker for each source, the sources can be configured to send to the destination.

qpид-route Summary and Options

```

$ qpid-route
Usage: qpid-route [OPTIONS] dynamic add <dest-broker> <src-broker> <exchange> [tag]
[exclude-list]
    qpid-route [OPTIONS] dynamic del <dest-broker> <src-broker> <exchange>

    qpid-route [OPTIONS] route add    <dest-broker> <src-broker> <exchange> <routing-key> [tag]
[exclude-list]
    qpid-route [OPTIONS] route del    <dest-broker> <src-broker> <exchange> <routing-key>
    qpid-route [OPTIONS] queue add    <dest-broker> <src-broker> <exchange> <queue>
    qpid-route [OPTIONS] queue del    <dest-broker> <src-broker> <exchange> <queue>
    qpid-route [OPTIONS] route list [<dest-broker>]
    qpid-route [OPTIONS] route flush [<dest-broker>]
    qpid-route [OPTIONS] route map   [<broker>]

    qpid-route [OPTIONS] link add   <dest-broker> <src-broker>
    qpid-route [OPTIONS] link del   <dest-broker> <src-broker>
    qpid-route [OPTIONS] link list [<dest-broker>]

Options:
--timeout seconds (10)    Maximum time to wait for broker connection
-v [ --verbose ]           Verbose output
-q [ --quiet ]              Quiet output, don't print duplicate warnings
-d [ --durable ]            Added configuration shall be durable
-e [ --del-empty-link ]     Delete link after deleting last route on the link
-s [ --src-local ]          Make connection to source broker (push route)
--ack N                     Acknowledge transfers over the bridge in batches of N
-t <transport> [ --transport <transport> ]   Specify transport to use for links, defaults to tcp

dest-broker and src-broker are in the form: [username/password@] hostname | ip-address
[:<port>]
ex: localhost, 10.1.1.7:10000, broker-host:10000, guest/guest@localhost

```

There are several transport options available for the federation link:

Transport	Description
tcp	(default) A cleartext TCP connection
ssl	A secure TLS/SSL over TCP connection
rdma	A Connection using the RDMA interface (typically for an Infiniband network)

The `tag` and `exclude-list` arguments are not needed. They have been left in place for backward compatibility and for advanced users who might have very unusual requirements. If you're not sure if you need them, you don't. Leave them alone. If you must know, please refer to "Message Loop Prevention" in the advanced topics section below. The prevention of message looping is now automatic and requires no user action.

If the link between the two sites has network latency, this can be compensated for by increasing the ack frequency with `--ack N` to achieve better batching across the link between the two sites.

Caveats, Limitations, and Things to Avoid

Redundant Paths

The current implementation of federation in the M4 broker imposes constraints on redundancy in the topology. If there are parallel paths from a producer to a consumer, multiple copies of messages may be received.

A future release of Qpid will solve this problem by allowing redundant paths with cost metrics. This will allow the deployment of networks that are tolerant of connection or broker loss.

Lack of Flow Control

M4 broker federation uses unlimited flow control on the federation sessions. Flow control back-pressure will not be applied on inter-broker subscriptions.

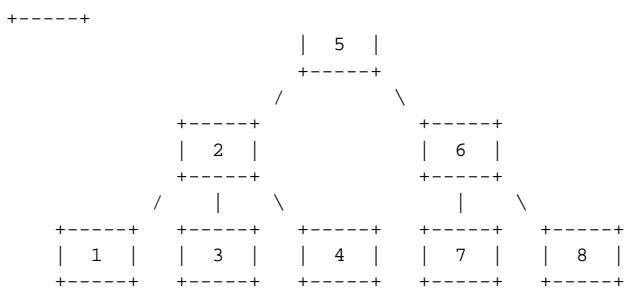
Lack of Cluster Failover Support

The client functionality embedded in the broker for inter-broker links does not currently support cluster fail-over. This will be added in a subsequent release.

Example Scenarios

Using QPID to bridge disjoint IP networks

Multi-tiered topology



This topology can be configured using the following script.

```
##  
## Define URLs for the brokers  
##  
broker1=localhost:10001  
broker2=localhost:10002  
broker3=localhost:10003  
broker4=localhost:10004  
broker5=localhost:10005  
broker6=localhost:10006  
broker7=localhost:10007  
broker8=localhost:10008  
  
##  
## Create Topic Exchanges  
##  
qpidd-config -a $broker1 add exchange topic fed.topic  
qpidd-config -a $broker2 add exchange topic fed.topic  
qpidd-config -a $broker3 add exchange topic fed.topic  
qpidd-config -a $broker4 add exchange topic fed.topic  
qpidd-config -a $broker5 add exchange topic fed.topic  
qpidd-config -a $broker6 add exchange topic fed.topic  
qpidd-config -a $broker7 add exchange topic fed.topic  
qpidd-config -a $broker8 add exchange topic fed.topic  
  
##  
## Create Topic Routes  
##  
qpidd-route dynamic add $broker1 $broker2 fed.topic  
qpidd-route dynamic add $broker2 $broker1 fed.topic  
  
qpidd-route dynamic add $broker3 $broker2 fed.topic  
qpidd-route dynamic add $broker2 $broker3 fed.topic  
  
qpidd-route dynamic add $broker4 $broker2 fed.topic  
qpidd-route dynamic add $broker2 $broker4 fed.topic  
  
qpidd-route dynamic add $broker2 $broker5 fed.topic  
qpidd-route dynamic add $broker5 $broker2 fed.topic  
  
qpidd-route dynamic add $broker5 $broker6 fed.topic  
qpidd-route dynamic add $broker6 $broker5 fed.topic  
  
qpidd-route dynamic add $broker6 $broker7 fed.topic  
qpidd-route dynamic add $broker7 $broker6 fed.topic  
  
qpidd-route dynamic add $broker6 $broker8 fed.topic  
qpidd-route dynamic add $broker8 $broker6 fed.topic
```

Load-sharing across brokers

Advanced Topics

Federation Queue Naming

Message Loop Prevention

ACL

v2 ACL file format for brokers

This new ACL implementation has been designed for implementation and interoperability on all Qpid brokers. It is currently supported in the following brokers:

Broker	Version
C++	M4 onward
Java	M5 anticipated

Contents

- v2 ACL file format for brokers
 - Specification
 - Validation
 - Example file:
- Design Documentation
 - Mapping of ACL traps to action and type
- v2 ACL User Guide
 - Writing Good/Fast ACL
 - Getting ACL to Log
 - User Id / domains running with C++ broker

Specification

Notes on file formats

- A line starting with the character '#' will be considered a comment, and are ignored.
- Since the '#' char (and others that are commonly used for comments) are commonly found in routing keys and other AMQP literals, it is simpler (for now) to hold off on allowing trailing comments (ie comments in which everything following a '#' is considered a comment). This could be reviewed later once the rest of the format is finalized.
- Empty lines ("") and lines that contain only whitespace (any combination of ' ', '\f', '\n', '\r', '\t', '\v') are ignored.
- All tokens are case sensitive. "name1" != "Name1" and "create" != "CREATE".
- Group lists may be extended to the following line by terminating the line with the '\' character. However, this may only occur after the group name or any of the names following the group name. Empty extension lines (ie just a '\' character) are not permitted.

```
# Examples of extending group lists using a trailing '\' character

group group1 name1 name2 \
    name3 name4 \
    name5

group group2 \
    group1 \
    name6

# The following are illegal:

# '\' must be after group name
group \
    group3 name7 name8

# No empty extension lines
group group4 name9 \
    \
    name10
```

- Additional whitespace (ie more than one whitespace char) between and after tokens is ignored. However group and acl definitions must start with "group" or "acl" respectively and with no preceding whitespace.
- All acl rules are limited to a single line.
- Rules are interpreted from the top of the file down until the name match is obtained; at which point processing stops.
- The keyword "all" is reserved, and matches all individuals, groups and actions. It may be used in place of a group or individual name and/or an action - eg "acl allow all all", "acl deny all all" or "acl deny user1 all".
- The last line of the file (whether present or not) will be assumed to be "acl deny all all". If present in the file, any lines below this one are ignored.

- Property names, Usernames and group names may contain only **a-z , A-Z , 0-9 , '-' , '_' & '!'**.
- Rules must be preceded by any group definitions they may use; any name not previously defined as a group will be assumed to be that of an individual.
- ACL rules must have the following tokens in order on a single line:
 - The string literal "acl";
 - The permission;
 - The name of a single group or individual or the keyword "all";
 - The name of an action or the keyword "all";
 - Optionally, a single object name or the keyword "all";
 - If the object is present, then optionally one or more property name-value pair(s) (in the form property=value).

```

user = username[/domain[@realm]]
user-list = user1 user2 user3 ...
group-name-list = group1 group2 group3 ...

group <group-name> = [user-list] [group-name-list]

permission = [allow|allow-log|deny|deny-log]
action = [consume|publish|create|access|bind|unbind|delete|purge|update]
object = [virtualhost|queue|exchange|broker|link|route|method]
property =
[name|durable|owner|routingkey|passive|autodelete|exclusive|type|alternate|queuename|schemapackage|sc
permission {<group-name>|<user-name>|"all"} {action| "all"} [object|"all"]
[property=<property-value>]
```

Validation

The new ACL file format needs to perform validation on the acl rules. The validation should be performed depending on the set value:

`strict-acl-validation=[none]`

The default setting should be 'warn'

On validation of this acl the following checks would be expected:

```
acl allow client publish routingkey=exampleQueue exchange=amq.direct
```

1. The If the user 'client' cannot be found, if the authentication mechanism cannot be queried then a 'user' value should be added to the file.
2. There is an exchange called 'amq.direct'
3. There is a queue bound to 'exampleQueue' on 'amq.direct'

Each of these checks that fail will result in a log statement being generated.

In the case of a fatal logging the full file will be validated before the broker shuts down.

Example file:

```

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

# Some rules
acl allow carl@QPID create exchange name=carl.*
acl deny rob@QPID create queue
acl allow guest@QPID bind exchange name=amq.topic routingkey=stocks.ibm.# owner=self
acl allow user-consume create queue name=tmp.*

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

# Last (default) rule
acl deny all all

```

Design Documentation

Mapping of ACL traps to action and type

The C++ broker maps the ACL traps in the follow way for AMQP 0-10:

The Java broker currently only performs ACLs on the AMQP connection not on management functions:

Object	Action	Properties	Trap C++	Trap Java
Exchange	Create	name type alternate passive durable	ExchangeHandlerImpl::declare	ExchangeDeclareHandler
Exchange	Delete	name	ExchangeHandlerImpl::delete	ExchangeDeleteHandler
Exchange	Access	name	ExchangeHandlerImpl::query	
Exchange	Bind	name routingkey queuename owner	ExchangeHandlerImpl::bind	QueueBindHandler
Exchange	Unbind	name routingkey	ExchangeHandlerImpl::unbind	ExchangeUnbindHandler
Exchange	Access	name queuename routingkey	ExchangeHandlerImpl::bound	
Exchange	Publish	name routingKey	SemanticState::route	BasicPublishMethodHandler
Queue	Access	name	QueueHandlerImpl::query	
Queue	Create	name alternate passive durable exclusive autodelete	QueueHandlerImpl::declare	QueueDeclareHandler
Queue	Purge	name	QueueHandlerImpl::purge	QueuePurgeHandler
Queue	Purge	name	Management::Queue::purge	
Queue	Delete	name	QueueHandlerImpl::delete	QueueDeleteHandler
Queue	Consume	name (possibly add in future?)	MessageHandlerImpl::subscribe	BasicConsumeMethodHandler BasicGetMethodHandler
<Object>	Update		ManagementProperty::set	
<Object>	Access		ManagementProperty::read	
Link	Create		Management::connect	
Route	Create		Management:: - createFederationRoute-	

Route	Delete		Management:: - deleteFederationRoute	
Virtualhost	Access	name	TBD	ConnectionOpenMethodHandler

Management actions that are not explicitly given a name property it will default the name property to management method name, if the action is 'W' Action will be 'Update', if 'R' Action will be 'Access'.

for example, if the mgmt method 'joinCluster' was not mapped in schema it will be mapped in ACL file as follows

Object	Action	Property
Broker	Update	name=joinCluster

v2 ACL User Guide

Writing Good/Fast ACL

The file gets read top down and rule get passed based on the first match. In the following example the first rule is a dead rule. I.e. the second rule is wider than the first rule. DON'T do this, it will force extra analysis, worst case if the parser does not kill the dead rule you might get a false deny.

```
allow peter@QPID create queue name=tmp <-- dead rule!!
allow peter@QPID create queue
deny all all
```

By default files end with

```
deny all all
```

the mode of the ACL engine can be swapped to be allow based by putting the following at the end of the file

```
allow all all
```

Note that 'allow' based file will be a LOT faster for message transfer. This is because the AMQP specification does not allow for creating subscribers on publish, so the ACL is executed on every message transfer. Also, ACL's rules using less properties on publish will in general be faster.

Getting ACL to Log

In order to get log messages from ACL actions use allow-log and deny-log for example

```
allow-log john@QPID all all
deny-log guest@QPID all all
```

User Id / domains running with C++ broker

The user-id used for ACL is taken from the connection user-id. Thus in order to use ACL the broker authentication has to be setup. i.e. (if --auth no is used in combination with ACL the broker will deny everything)

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

To load the ACL module use, load the acl module cmd line or via the config file

```
./src/qpidd --load-module src/.libs/acl.so
```

The ACL plugin provides the following option '--acl-file'. If no ACL file is supplied the broker will not enforce ACL. If an ACL file name is supplied, and the file does not exist or is invalid the broker will not start.

```

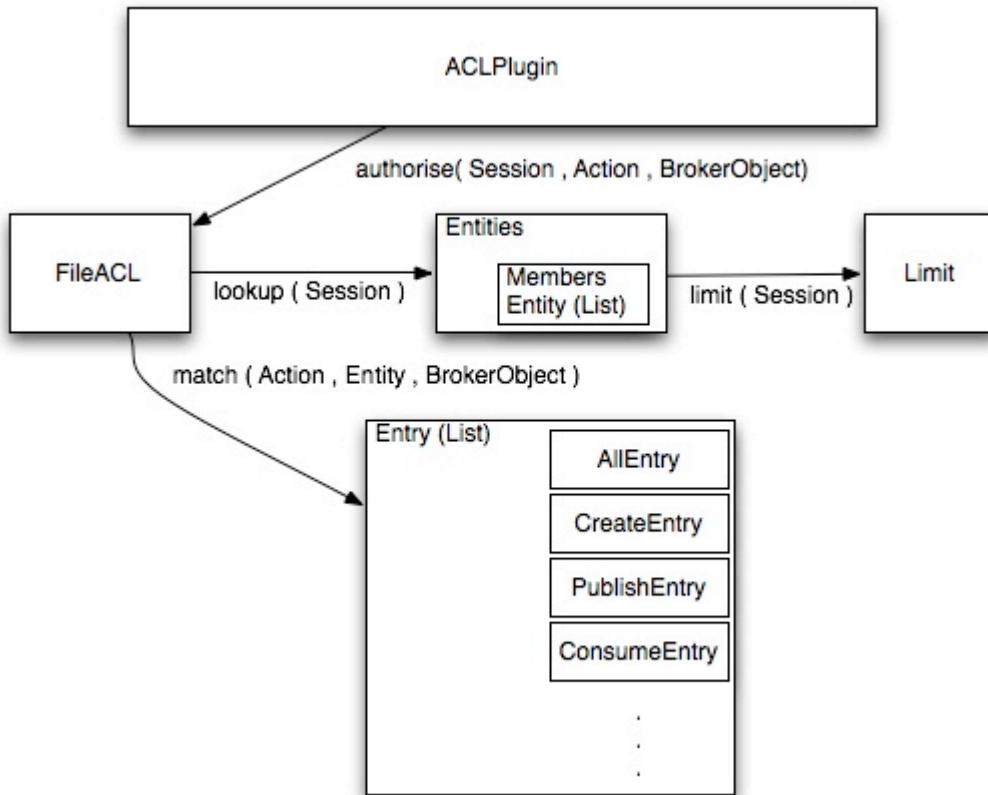
ACL Options:
--acl-file FILE      The policy file to load from, loaded from data dir

```

FileACL Design

FileACL : ACL v2 Java design documentation

Design



Interface

The current ACLPlugin interface has a single `authorise` method, however its current format ties it to the Framing layer. The interface has been abstracted to take Actions and a new BrokerObject on which the action should be performed.

```
authorise( Session, Action , BrokerObject )
```

BrokerObjects

These new objects are taken from the ACLv2 documentation are used to represent the internal broker objects. These objects can be created with the properties so the ACL can be evaluated without needing access to the functional broker components. Providing the actual broker objects is not possible as that would require items such as an AMQQueue to be created before evaluating whether the User has rights to create the queue.

ACL Entries

Each line from the ACL file is converted into an Entry and added to a list maintaining order for later evaluation. Each ACL Action type maps to an entry which in turn handles the processing. The Entries are much smaller and clearer to understand than the large case statement method that was utilised in the SimpleXML ACL Plugin. There is also scope here to provide an extension point to limit the ability of an entity.

```
[user|group] ... limit-<limit-type>=<value>
```

Examples would be to limit the number of connections a user may create or IP White/Black listing.

Current Development State

Currently the FileACL processing of the file format is complete and unit tested. Each of the entries have been created however they all do not fully take in to consideration all the potential variations of Objects and Properties that can be specified.

Testing has started by modifying the existing SimpleACLTTest to allow different configuration and ACLPlugins to be loaded and evaluated against the existing Request/Response application design.

Items to complete

1. Complete implementation and testing of all Object an Property combinations
2. Complete parsing of *user* to understand Realms and Domains.
3. Provide an ACL independent mechanism for testing ACLs that can be performed against both Java & C++ brokers to ensure consistency in implementation. This would also allow future ACL implementations to be tested for consistency with existing implementations.

Note: What do these new properties mean for the Java broker.

Integration with existing ACLPlugin

The development of this plugin has been done to require no changes to the existing broker. For clarity a rename of the Permission class to Action has been carried. The introduction of the above interface needs further discussion as any new interface will require future support.

For the moment the existing ACLPlugin interface has been implemented and maps from the Framing layer to Actions and BrokerObjects.

Future Refinement

Analysing each ACL Entry in turn for a large acl file would be expensive however, it would be possible to perform some load time optimisations.

Examples of such optimisations are:

- Each Action could have its own list so as to eliminate the lookup and method invocation on Entries that will never succeed.
- Each Entity could also have a list of entries attached to it that would allow quicker evaluation of the entries based on those that pertain to the given Entity
- Both of these optimisations can be applied to allow a much shorter list of Actions to be retrieved for a given Entity.

Qpid Management Framework

- [What Is QMF](#)
- [Getting Started with QMF](#)
- [QMF Concepts](#)
 - Console, Agent, and Broker
 - Schema
 - Class Keys and Class Versioning
- [The QMF Protocol](#)
- [How to Write a QMF Console](#)
- [How to Write a QMF Agent](#)

| [Please visit the QMFv2 Project Page for information about the future of QMF.](#)

What Is QMF

QMF (Qpid Management Framework) is a general-purpose management bus built on Qpid Messaging. It takes advantage of the scalability, security, and rich capabilities of Qpid to provide flexible and easy-to-use manageability to a large set of applications.

Getting Started with QMF

QMF is used through two primary APIs. The *console* API is used for console applications that wish to access and manipulate manageable components through QMF. The *agent* API is used for application that wish to be managed through QMF.

The fastest way to get started with QMF is to work through the "How To" tutorials for consoles and agents. For a deeper understanding of what is happening in the tutorials, it is recommended that you look at the *Qmf Concepts* section.

QMF Concepts

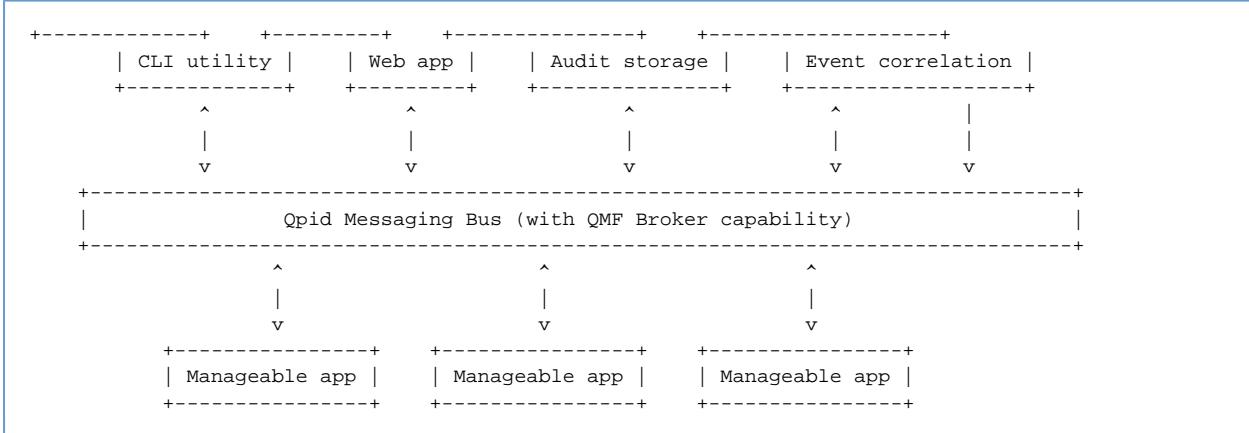
This section introduces important concepts underlying QMF.

Console, Agent, and Broker

The major architectural components of QMF are the Console, the Agent, and the Broker. Console components are the "managing" components of QMF and agent components are the "managed" parts. The broker is a central (possibly distributed, clustered and fault-tolerant) component that manages name spaces and caches schema information.

A console application may be a command-line utility, a three-tiered web-based GUI, a collection and storage device, a specialized application that monitors and reacts to events and conditions, or anything else somebody wishes to develop that uses QMF management data.

An agent application is any application that has been enhanced to allow itself to be managed via QMF.



In the above diagram, the *Manageable apps* are agents, the *CLI utility*, *Web app*, and *Audit storage* are consoles, and *Event correlation* is both a console and an agent because it can create events based on the aggregation of what it sees.

Schema

A *schema* describes the structure of management data. Each *agent* provides a schema that describes its management model including the object classes, methods, events, etc. that it provides. In the current QMF distribution, the agent's schema is codified in an XML document. In the near future, there will also be ways to programmatically create QMF schemata.

Package

Each agent that exports a schema identifies itself using a *package* name. The package provides a unique namespace for the classes in the agent's schema that prevent collisions with identically named classes in other agents' schemata.

Package names are in "reverse domain name" form with levels of hierarchy separated by periods. For example, the Qpid messaging broker uses package "org.apache.qpid.broker" and the Access Control List plugin for the broker uses package "org.apache.qpid.acl". In general, the package name should be the reverse of the internet domain name assigned to the organization that owns the agent software followed by identifiers to uniquely identify the agent.

The XML document for a package's schema uses an enclosing <schema> tag. For example:

```

<schema package="org.apache.qpid.broker">
</schema>

```

Object Classes

Object classes define types for manageable objects. The agent may create and destroy objects which are instances of object classes in the schema. An object class is defined in the XML document using the <class> tag. An object class is composed of properties, statistics, and methods.

```

<class name="Exchange">
  <property name="vhostRef" type="objId" references="Vhost" access="RC" index="y" parentRef="y"/>
  <property name="name" type="sstr" access="RC" index="y"/>
  <property name="type" type="sstr" access="RO"/>
  <property name="durable" type="bool" access="RC"/>
  <property name="arguments" type="map" access="RO" desc="Arguments supplied in exchange.declare"/>

  <statistic name="producerCount" type="hilo32" desc="Current producers on exchange"/>
  <statistic name="bindingCount" type="hilo32" desc="Current bindings"/>
  <statistic name="msgReceives" type="count64" desc="Total messages received"/>
  <statistic name="msgDrops" type="count64" desc="Total messages dropped (no matching key)" />
  <statistic name="msgRoutes" type="count64" desc="Total routed messages"/>
  <statistic name="byteReceives" type="count64" desc="Total bytes received"/>
  <statistic name="byteDrops" type="count64" desc="Total bytes dropped (no matching key)" />
  <statistic name="byteRoutes" type="count64" desc="Total routed bytes"/>
</class>

```

Properties and Statistics

<property> and <statistic> tags must be placed within <schema> and </schema> tags.

Properties, statistics, and methods are the building blocks of an object class. Properties and statistics are both object attributes, though they are treated differently. If an object attribute is defining, seldom or never changes, or is large in size, it should be defined as a *property*. If an attribute is rapidly changing or is used to instrument the object (counters, etc.), it should be defined as a *statistic*.

The XML syntax for <property> and <statistic> have the following XML-attributes:

Attribute	<property>	<statistic>	Meaning
name	Y	Y	The name of the attribute
type	Y	Y	The data type of the attribute
unit	Y	Y	Optional unit name - use the singular (i.e. MByte)
desc	Y	Y	Description to annotate the attribute
references	Y		If the type is "objId", names the referenced class
access	Y		Access rights (RC, RW, RO)
index	Y		"y" if this property is used to uniquely identify the object. There may be more than one index property in a class
parentRef	Y		"y" if this property references an object in which this object is in a child-parent relationship.
optional	Y		"y" if this property is optional (i.e. may be NULL/not-present)
min	Y		Minimum value of a numeric attribute
max	Y		Maximum value of a numeric attribute
maxLen	Y		Maximum length of a string attribute

Methods

<method> tags must be placed within <schema> and </schema> tags.

A *method* is an invokable function to be performed on instances of the object class (i.e. a Remote Procedure Call). A <method> tag has a name, an optional description, and encloses zero or more arguments. Method arguments are defined by the <arg> tag and have a name, a type, a direction, and an optional description. The argument direction can be "I", "O", or "IO" indicating input, output, and input/output respectively. An example:

```
<method name="echo" desc="Request a response to test the path to the management broker">
  <arg name="sequence" dir="IO" type="uint32"/>
  <arg name="body"      dir="IO" type="lstr"/>
</method>
```

Event Classes

Data Types

Object attributes, method arguments, and event arguments have data types. The data types are based on the rich data typing system provided by the AMQP messaging protocol. The following table describes the data types available for QMF:

QMF Type	Description
REF	QMF Object ID - Used to reference another QMF object.
U8	8-bit unsigned integer
U16	16-bit unsigned integer
U32	32-bit unsigned integer
U64	64-bit unsigned integer
S8	8-bit signed integer
S16	16-bit signed integer
S32	32-bit signed integer
S64	64-bit signed integer

BOOL	Boolean - True or False
SSTR	Short String - String of up to 255 bytes
LSTR	Long String - String of up to 65535 bytes
ABSTIME	Absolute time since the epoch in nanoseconds (64-bits)
DELTATIME	Delta time in nanoseconds (64-bits)
FLOAT	Single precision floating point number
DOUBLE	Double precision floating point number
UUID	UUID - 128 bits
FTABLE	Field-table - std::map in C++, dictionary in Python

In the XML schema definition, types go by different names and there are a number of special cases. This is because the XML schema is used in code-generation for the agent API. It provides options that control what kind of accessors are generated for attributes of different types. The following table enumerates the types available in the XML format, which QMF types they map to, and other special handling that occurs.

XML Type	QMF Type	Accessor Style	Special Characteristics
objId	REF	Direct (get, set)	
uint8,16,32,64	U8,16,32,64	Direct (get, set)	
int8,16,32,64	S8,16,32,64	Direct (get, set)	
bool	BOOL	Direct (get, set)	
sstr	SSTR	Direct (get, set)	
lstr	LSTR	Direct (get, set)	
absTime	ABSTIME	Direct (get, set)	
deltaTime	DELTATIME	Direct (get, set)	
float	FLOAT	Direct (get, set)	
double	DOUBLE	Direct (get, set)	
uuid	UUID	Direct (get, set)	
map	FTABLE	Direct (get, set)	
hilo8,16,32,64	U8,16,32,64	Counter (inc, dec)	Generates value, valueMin, valueMax
count8,16,32,64	U8,16,32,64	Counter (inc, dec)	
mma32,64	U32,64	Direct	Generates valueMin, valueMax, valueAverage, valueSamples
mmaTime	DELTATIME	Direct	Generates valueMin, valueMax, valueAverage, valueSamples



Important

When writing a schema using the XML format, types used in `<property>` or `<arg>` must be types that have *Direct* accessor style. Any type may be used in `<statistic>` tags.

Class Keys and Class Versioning

The QMF Protocol

The QMF protocol defines the message formats and communication patterns used by the different QMF components to communicate with one another.

A description of the current version of the QMF protocol can be found at [QMF Protocol](#).

A proposal for an updated protocol based on map-messages is in progress and can be found at [QMF Map Message Protocol](#).

How to Write a QMF Console

Please see the [QMF Python Console Tutorial](#) for information about using the console API with Python.

How to Write a QMF Agent

QMan

QMan - QMF/JMX Bridge

QMan is a Qpid management bridge used for exposing one (or more) Qpid broker domain model as MBean through Java Management Extensions (JMX).

Note:

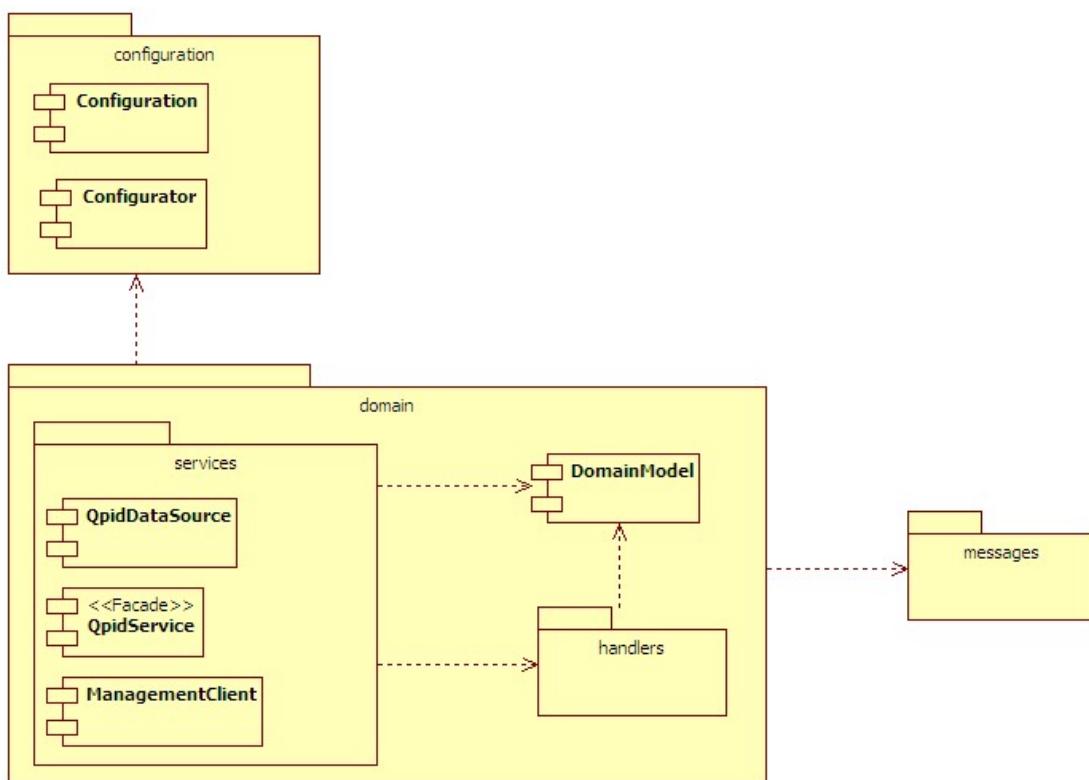
QMan has been contributed by Andrea Gazzarini. Details and discussion can be found at <https://issues.apache.org/jira/browse/QPID-1284>.

Description

QMan is a standalone application that is able to communicate, using AMQP management extensions, with one or more remote brokers.

==== To be completed! ===

Package View



Package configuration

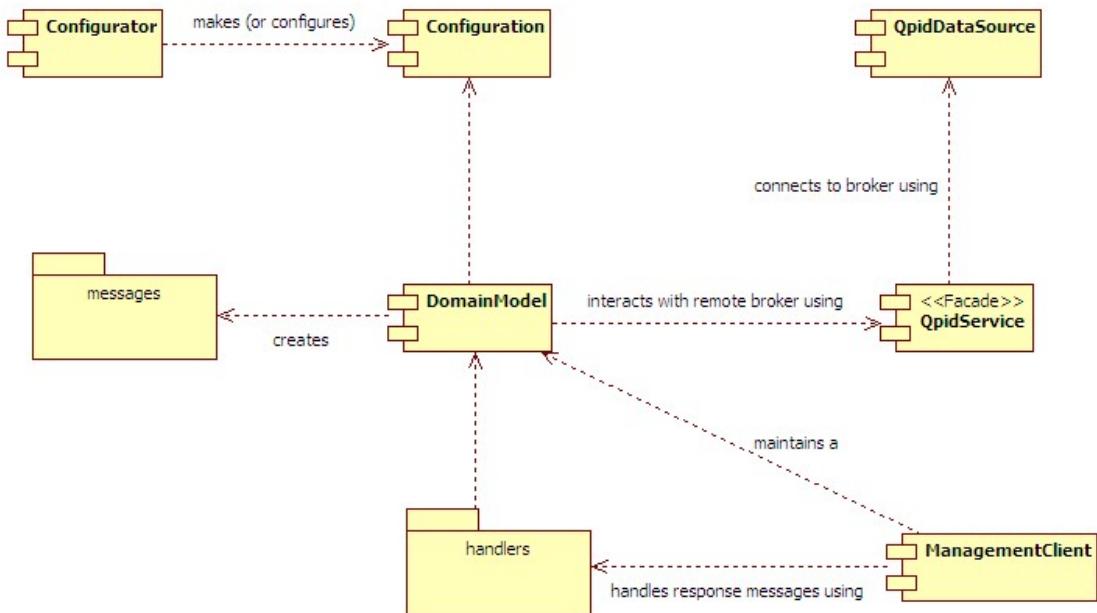
Package domain

Package services

Package handlers

Package messages

Component view



Configurator

Configuration

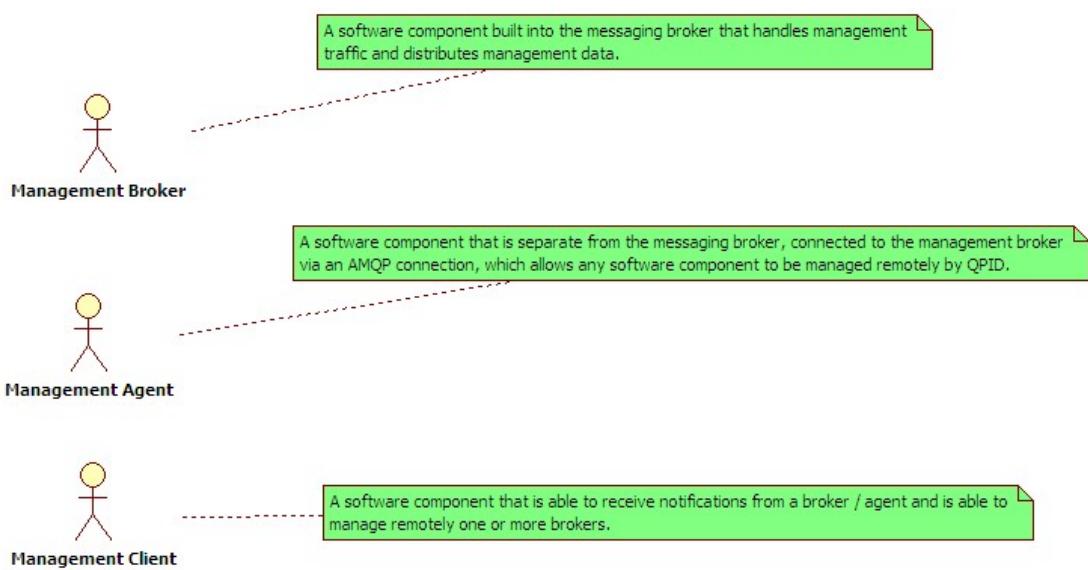
QpidDataSource

Domain Model

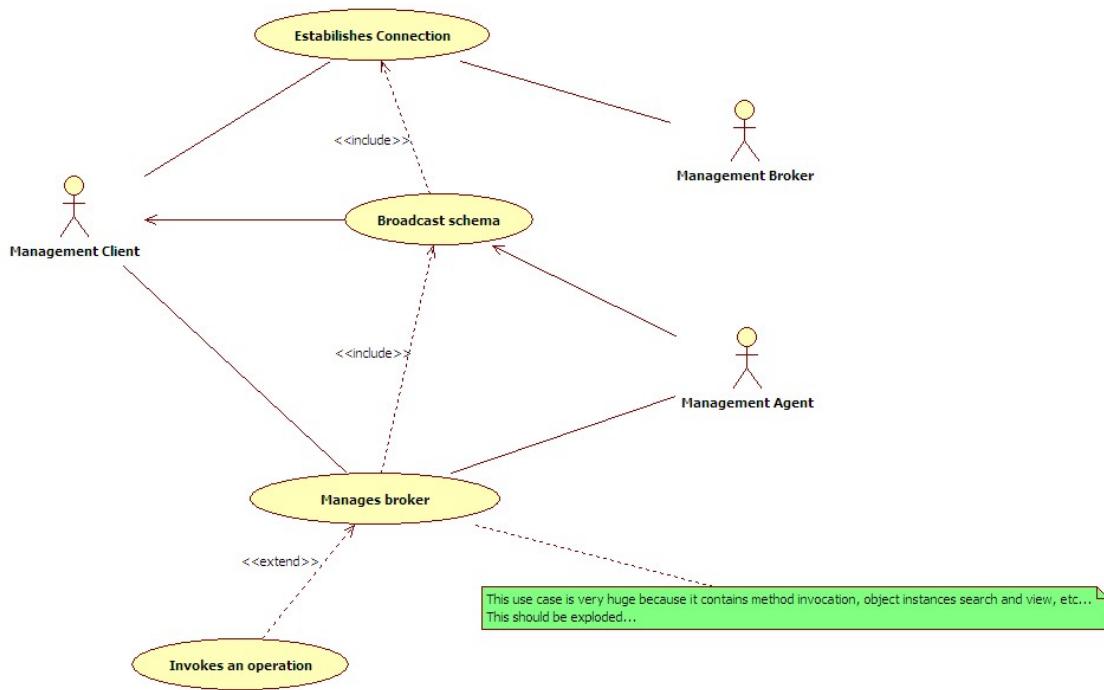
QpidService

ManagementClient

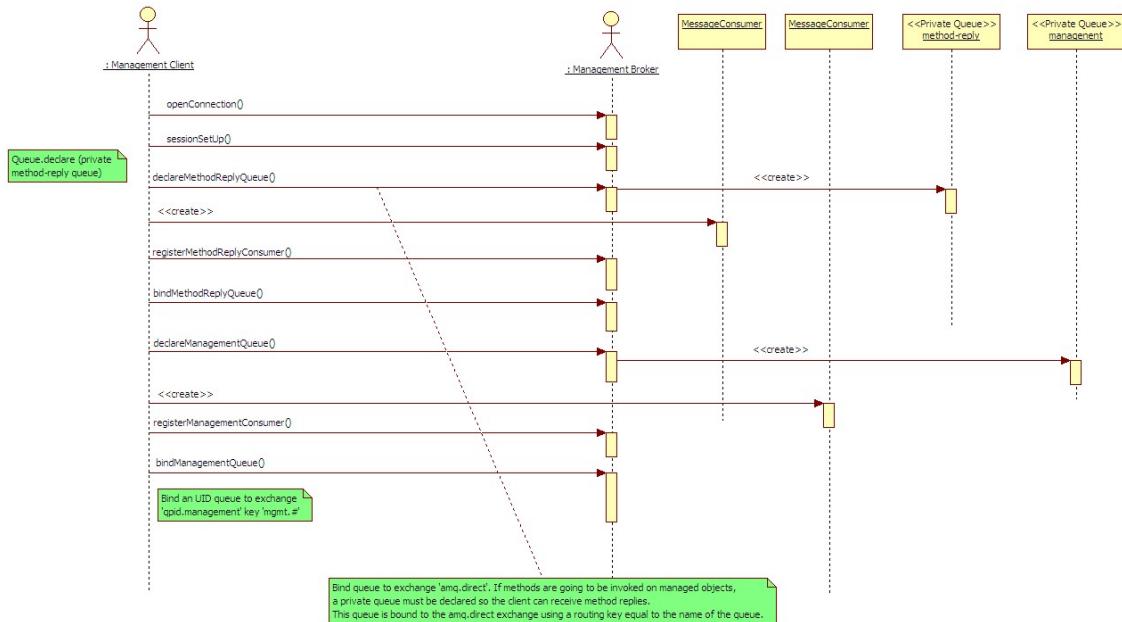
Use case view



- Management broker :
- Management agent :
- Management client :



Establishes first connection



Broadcast schema

Manages broker

Open points

- JMX interface : QMan should be exposed itself as an MBean for remote management ;
- Connector : QMan needs to be exposed as a service using a connector (WS-DM, SNMP, SOAP, etc...) ;
- Events : the latest version of AMQP management extensions includes event definitions so QMan needs to expose such events in the local domain model ;

Glossary

Definition	Description
JMX	Java Management Extensions

Screenshots

- [Viewing Queue Statistics](#)
- [Moving Message Between Queues](#)

QMF Map Message Protocol

QMFn2 Map Message Protocol

Introduction

This document describes the design of a proposed protocol for QMF based on map-messages (offered by the new C++ and Python APIs as well as the existing JMS API).

If adopted, this new protocol will change the formats of the messages used by QMF components to communicate. It will also change some of the message exchange patterns. It will **not** significantly impact the console and agent APIs and is intended to operate with applications that use the current QMF APIs.

Some highlights of the new design:

- Current QMF message bodies are in packed binary formats. While quite efficient, this style of formatting makes it difficult to make changes to the format and content for new features. The proposed format is based on encoded maps (a.k.a. dictionaries, field-tables) which are very easily extended and require less context to be useful.
- QMF currently requires the message broker to participate in the QMF protocol. The proposed protocol removes this requirement and will run properly on any AMQP message broker.
- QMF Agents currently publish periodic updates of their managed content to a globally accessible topic. This has security implications with regard to access to data. This is also inflexible in that updates to all data are sent at the same intervals. The proposed protocol removes the global publishing of data and introduces a subscription-query whereby a console may request that an agent publish certain data at a certain interval to an indicated target. Such requests can be subject to access control and may be focused on only the data that is needed for a particular application.
- The proposed protocol allows for more general use of data. For example:
 - Free-form data, that has no object-identifier nor schema, can be transferred. This is useful for complex queries (joins, reports, etc.).
 - Methods can be invoked against an agent in the absence of a managed object.

QMF Protocol

Use of Message Headers

Standard Message Properties

Message Property	Use
correlation-id	Used in request/response/indication sets to correlate responses and indications to their request.
reply-to	Used in requests to indicate the address for the response.
content-type	'amqp/map' or 'amqp/list'
user-id	Supplied in a request if authentication/authorization at the agent is appropriate.
app-id	'qmf2'

Custom Application Headers

Application Header Key	Use
method	'request', 'response', or 'indication'. This field describes the message's role in a particular message-exchange pattern.
partial	Void. If this field is present, it indicates that the message does not contain the complete request or response (i.e. another message will follow using the same opcode and correlation-id)
qmf.opcode	QMF-specific operation code (see list below). The opcode defines what content, if any, is to be found in the message body.
qmf.content	If the opcode is a data indication, this field indicates what kind of data will be found in the message body.
qmf.agent	If this message is a data indication sent by an agent, this field contains the agent's name.

QMF OpCodes

qmf.opcode field	Message Body Data Type	Sent By	Sent To	Description
_exception	QMF_DATA	Agent	reply-to	This general-purpose message can be sent by an agent in response to any request (query, subscription, method) if the request could not be completed for any reason. The QMF_DATA in the message body contains details of why the failure occurred.
_agent_locate_request	QMF_QUERY_PREDICATE	Console	QMF Topic	A console may send an agent-locate-request in order to reach all available agents. The predicate may be used to limit the set of agents that will respond to the request.
_agent_locate_response	QMF_DATA	Agent	reply-to	This is a response to an agent-locate-request. An agent will send an agent-locate-response if it received an agent-locate-request with a predicate that matches its characteristics.
_agent_heartbeat_indication	QMF_DATA	Agent	Topic	Each agent periodically sends a heartbeat message to a topic to indicate that it is alive and connected. The content of the heartbeat message is the list of the agent's characteristics.
_query_request	QMF_QUERY	Console	Agent	A console sends a query to an agent to request that the agent send data to the requester.
_query_response	List of <qmf.content>	Agent	reply-to	The response to a query sent by a console.
_subscribe_request	QMF_SUBSCRIBE	Console	Agent	A console sends a subscribe-request to an agent to receive data matched by a query. A subscription differs from a query request in that it continues to send updated information to the console when the data changes.
_subscribe_response	QMF_SUBSCRIPTION	Agent	reply-to	When an agent receives a subscribe-request, it sends a subscribe-response granting (or refusing) the subscription. Should the subscription succeed, the response will contain an identifier for the subscription assigned by the Agent. Thereafter, it will send data-indication messages on the same correlation-id with updates when they happen or periodically. The first data-indication message sent by the agent will contain all matching data, subsequent data-indications will contain only those matching data that has changed since the last update.
_subscribe_cancel_indication	QMF_SUBSCRIPTION_ID	Console	Agent	A console can request that a subscription it created be immediately cancelled. This message must have the same correlation-id as the original request, and contain the subscription identifier as assigned by the Agent.
_subscribe_refresh_indication	QMF_SUBSCRIPTION_ID	Console	Agent	A console can keep a subscription alive by periodically refreshing it by sending a subscribe-refresh-indication. This message must have the same correlation-id as the original request, and contain the subscription identifier as assigned by the Agent.
_data_indication	List of <qmf.content>	Agent	reply-to or topic	A data indication is sent by an Agent when 1) subscription data has changed and needs to be published, 2) an event has occurred and event data is being published, and 3) any other time an agent wants to send unsolicited data.
_method_request	QMF_METHOD_CALL	Console	Agent	A console may invoke a method on an object managed by an agent. It may also invoke a method directly on the agent if appropriate. This message contains the method call including the input arguments.
_method_response	QMF_METHOD_RESULT	Agent	reply-to	A method call always results in a single method result. This message carries either the output arguments from a successful method call or it holds an exception to describe a failure.

QMF Content Types

qmf.content field	Data Type	Description
_schema_package	STRING	Schema package name

_schema_id	SCHEMA_ID	Schema class identifier
_schema_class	SCHEMA_CLASS	Schema class definition
_object_id	OBJECT_ID	Managed object identifier
_data	QMF_DATA	Data, managed and/or described or free-form
_event	QMF_EVENT	Event
_query	QMF_QUERY	Query

Message Body Map Formats

SCHEMA_ID

```
SCHEMA_ID := { _package_name: STRING,
                _class_name:    STRING,
                _type:          '_data' | '_event',
                _hash:          UUID
            }
```

Field	Optional	Description
_package_name	no	Package name (namespace) for the described class
_class_name	no	Name of the described class
_type	no	Class type: data or event
_hash	yes	Hash (uuid) to distinguish different versions of a class

SCHEMA_CLASS

```
SCHEMA_CLASS := { _schema_id: SCHEMA_ID,
                  _values:   { EACH_ATTR_NAME: SCHEMA_PROPERTY | SCHEMA_METHOD },
                  _subtypes: { EACH_ATTR_NAME: qmfProperty | qmfMethod }
              }
```

Field	Optional	Description
_schema_id	no	Identifier for this schema class
_values	no	Map of schema attribute names and either their property or method descriptions. The subtype defines whether an attribute is a property or a method.
_subtypes	no	Map of subtype names ('qmfProperty' or 'qmfMethod') for each attribute

SCHEMA_PROPERTY

```
SCHEMA_PROPERTY := { _type:           QMF_TYPE,
                     _access:        'RO' | 'RC' | 'RW',
                     _unit:          STRING,
                     _min:           NUMBER,
                     _max:           NUMBER,
                     _ maxlen:       NUMBER,
                     _dir:           'I' | 'O' | 'IO',
                     _desc:          STRING,
                     _references:   SCHEMA_ID,
                     _subtype:       QMF_SUBTYPE
                 }
```

Field	Optional	Description
_type	no	The QMF data type of this property
_access	yes	The remote access rules for this property: RO => Read Only (default if not specified) RC => Read Create RW => Read Write

_unit	yes	Annotation. Units of measure for numeric values
_min	yes	Minimum numeric value
_max	yes	Maximum numeric value
_ maxlen	yes	Maximum length of a variable length value (in octets)
_dir	yes	Used only for method arguments. Direction of transfer: I => Input (caller to callee) O => Output (callee to caller) IO => Both
_desc	yes	Annotation. Description of the property
_references	yes	If the type is a reference to another managed object, this field may be used to specify the required class for that object
_subtype	yes	May be used to further specify the meaning of the value of this field. For example, a number may actually be a timestamp or a duration. A string may be a reference to another object, or a URL.

QMF_TYPE

```
QMF_TYPE := 'TYPE_VOID' |
            'TYPE_BOOL' |
            'TYPE_INT' |
            'TYPE_FLOAT' |
            'TYPE_STRING' |
            'TYPE_MAP' |
            'TYPE_LIST' |
            'TYPE_UUID'
```

QMF_SUBTYPE

```
QMF_SUBTYPE := 'reference' |
                'url' |
                'timestamp' |
                'duration'
```

SCHEMA_METHOD

```
SCHEMA_METHOD := { _desc: STRING,
                  _arguments: { EACH_ARG_NAME: SCHEMA_PROPERTY } }
```

Field	Optional	Description
_desc	yes	Annotation. Description of this method
_arguments	no	Map of argument names and SCHEMA_PROPERTY data to describe them

QMF_METHOD_CALL

```
QMF_METHOD_CALL := { _object_id: OBJECT_ID,
                     _method_name: STRING,
                     _arguments: { EACH_KEY: VALUE },
                     _subtypes: { EACH_KEY: STRING }
                   }
```

Field	Optional	Description
_object_id	yes	The identity of the managed object receiving the method call. If not supplied, this method applies generally to the agent.
_method_name	no	The name of the method
_arguments	yes	The input arguments, if any
_subtypes	yes	Subtype information for the input arguments, if any

QMF_METHOD_RESULT

```
QMF_METHOD_RESULT := { _arguments: { EACH_KEY: VALUE },
                      _subtypes: { EACH_KEY: STRING }
                    }
```

Field	Optional	Description
_arguments	yes	Output arguments from a successful method call, if any
_subtypes	yes	Subtype information for the output arguments, if any

QMF_DATA

```
QMF_DATA := { _schema_id: SCHEMA_ID,
              _object_id: OBJECT_ID,
              _values: { EACH_KEY: VALUE },
              _subtypes: { EACH_KEY: STRING }
            }
```

Field	Optional	Description
_schema_id	yes	If this data is "described", this field references the schema class that describes the data.
_object_id	yes	If this data is "managed", this field provides the identifier that can be used to address this managed object.
_values	no	The map of values keyed by their property names
_subtypes	yes	Per-property subtypes that may be used to provide more information about the meaning of a value than its QMF_TYPE

OBJECT_ID

```
OBJECT_ID := { _agent_name: STRING,
               _agent_epoch: NUMBER,
               _object_name: STRING
             }
```

Field	Optional	Description
_agent_name	yes	Name of the agent that is managing the referenced data
_agent_epoch	yes	Numeric epoch of the agent process. This number is managed by the agent and is incremented each time the agent process starts. This field is only present for <i>transient</i> object IDs that must not be the same for a given object across an agent restart. <i>Persistent</i> object IDs must not include this field.
_object_name	no	Name of the data that uniquely identifies the data within the context of the agent.

QMF_QUERY

```
QMF_QUERY := { _what: QMF_QUERY_TARGET,
                _where: QMF_QUERY_PREDICATE,
                _object_id: OBJECT_ID,
                _schema_id: SCHEMA_ID
              }
```

Field	Optional	Description
_what	no	Identifies the kind of data being queried
_where	yes	Query predicate to limit the number of results of the query
_object_id	yes	Identifier of a single object being queried
_schema_id	yes	Identifier of a single schema being queried

QMF_QUERY_TARGET

```

QMF_QUERY_TARGET := 'SCHEMA_ID' |
                    'SCHEMA' |
                    'OBJECT_ID' |
                    'OBJECT'

```

QMF_QUERY_PREDICATE

QMF_SUBSCRIBE

```

QMF_SUBSCRIBE := { _query:      QMF_QUERY,
                   _duration:   NUMBER,
                   _interval:   NUMBER
                 }

```

Field	Optional	Description
_query	no	The query that defines the set of data being subscribed to
_duration	yes	The requested time (in seconds) after which this subscription will be automatically canceled. If a subscribe_refresh_indication is received by the agent running this query, this time interval will start over.
_interval	yes	The request time (in milliseconds) between periodic updates of data in this subscription. The agent may place a minimum on this interval.

QMF_SUBSCRIPTION

```

QMF_SUBSCRIPTION := { _subscription_id:      STRING,
                      _duration:       NUMBER,
                      _interval:       NUMBER,
                    }

```

Field	Optional	Description
_subscription_id	yes	Assigned by the Agent when replying to a successful subscription request. Must be supplied by the Console when sending a subscription refresh or cancel to the Agent for this subscription.
_duration	no	The time (in seconds) after which this subscription will be automatically canceled.
_interval	no	The time (in milliseconds) between periodic updates of data in this subscription.

QMF_SUBSCRIPTION_ID

```

QMF_SUBSCRIPTION_ID := { _subscription_id:      STRING}

```

Field	Optional	Description
_subscription_id	no	Supplied by the Console when sending a subscription refresh or cancel to the Agent for this subscription.

QMF Protocol



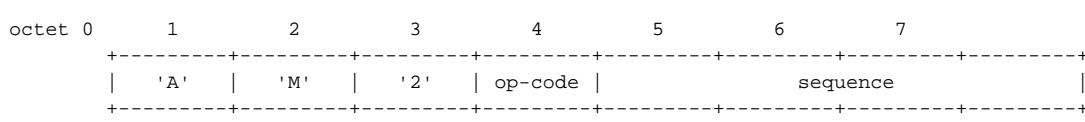
Note

This page is being updated with protocol changes introduced in M4 (and are unchanged in release 0.5)

Protocol Header

QMF messages are composed of sequences of binary-encoded data fields, in a manner consistent with the 0-10 version of the AMQP specification.

All QMF messages begin with a message header:



The first three octets contain the protocol **magic number** "AM2" which is used to identify the type and version of the message.

The **opcode** field identifies the operation represented by the message

Mapping QMF Messages to AMQP Messages

QMF messages are carried in the body segments of AMQP messages. An AMQP message body may contain 1 or more QMF messages. QMF messages do not span AMQP messages, each QMF messages must be entirely contained within a single AMQP message body.

Protocol Exchange Patterns

The following patterns are followed in the design of the protocol:

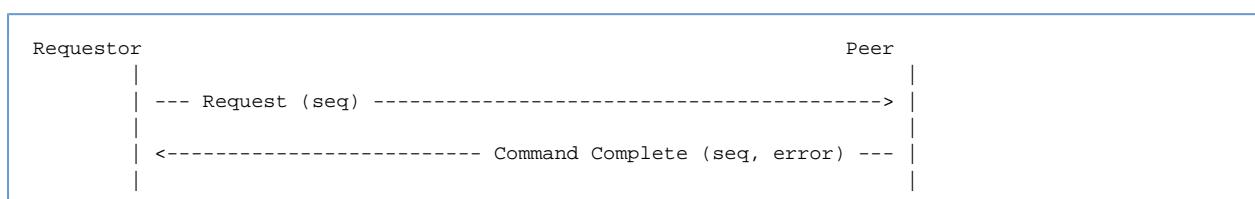
- Request-Response
- Query-Indication
- Unsolicited Indication

The Request-Response Pattern

In the request-response pattern, a requestor sends a **request** message to one of its peers. The peer then does one of two things: If the request can be successfully processed, a single **response** message is sent back to the requestor. This response contains the requested results and serves as the positive acknowledgement that the request was successfully completed.

If the request cannot be successfully completed, the peer sends a **command complete** message back to the requestor with an error code and error text describing what went wrong.

The sequence number in the **response** or **command complete** message is the same as the sequence number in the **request**.

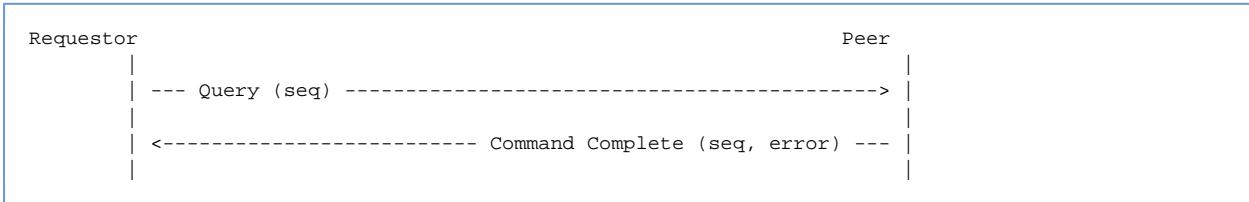
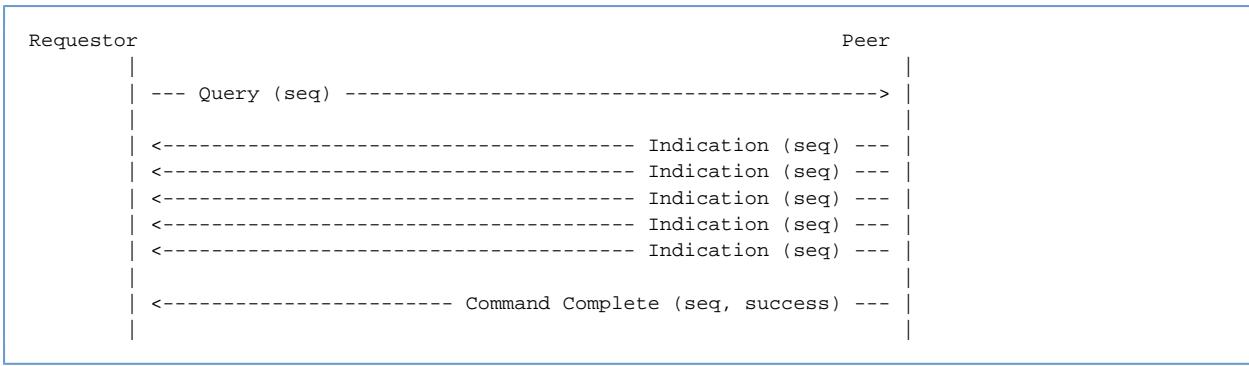


The Query-Indication Pattern

The query-indication pattern is used when there may be zero or more answers to a question. In this case, the requestor sends a **query** message to its peer. The peer processes the query, sending as many **indication** messages as needed back to the requestor (zero or more). Once the last **indication** has been sent, the peer then sends a **command complete** message with a success code indicating that the query is complete.

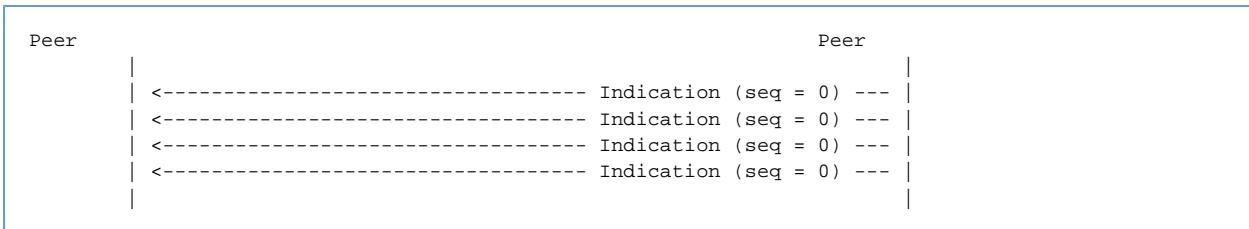
If there is an error in the **query**, the peer may reply with a **command complete** message containing an error code. In this case, no **indication** messages may be sent.

All **indication** and **command complete** messages shall have the same sequence number that appeared in the **query** message.



The Unsolicited-Indication Pattern

The unsolicited-indication pattern is used when one peer needs to send unsolicited information to another peer, or to broadcast information to multiple peers via a topic exchange. In this case, indication messages are sent with the sequence number field set to zero.



Object Identifiers

Manageable objects are tagged with a unique 128-bit object identifier. The object identifier space is owned and managed by the management broker. Objects managed by a single management broker shall have unique object identifiers. Objects managed by separate management brokers may have the same object identifier.

If a management console is designed to manage multiple management brokers, it must use the broker identifier as well as the object identifier to ensure global uniqueness.

```

first uint64:
  63 60 59          48 47          28 27          0
  +-----+-----+-----+
  | flags | sequence | broker bank | agent bank |
  +-----+-----+-----+
second uint64:
  63          0
  +-----+
  |          object |
  +-----+

```

Field	Size (bits)	Description
flags	4	Reserved, must be zero
sequence	12	Boot sequence of the agent, or zero for persistent IDs
broker bank	20	Bank number unique to the broker
agent bank	28	Bank number unique to the agent
object	64	Identifier assigned by the agent

- For persistent IDs, sequence is zero
- For non-persistent IDs, sequence is a number which increments each time the management broker is restarted.

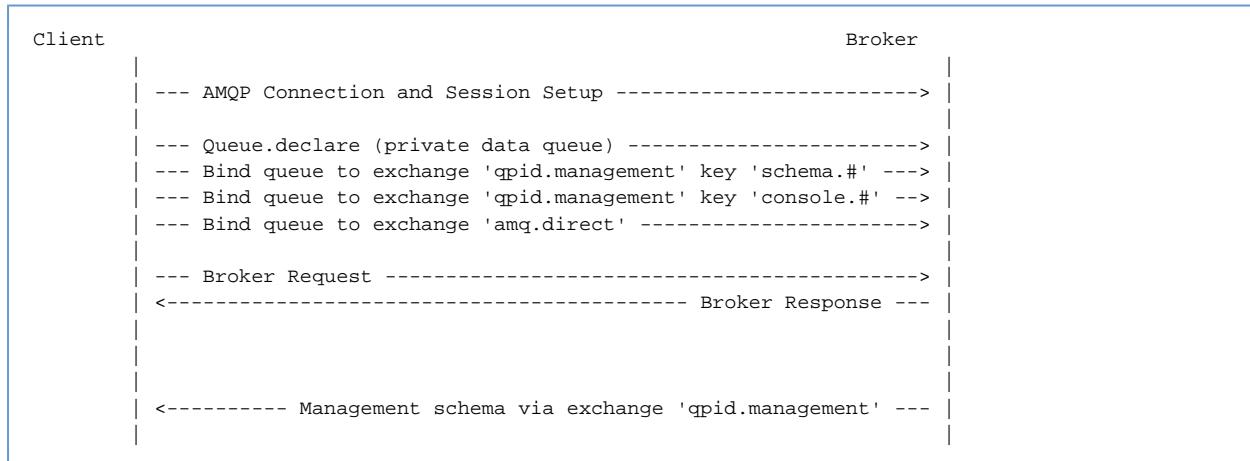
Establishing Communication Between Client and Agent

Communication is established between the management client and management agent using normal AMQP procedures. The client creates a connection to the broker and then establishes a session with its corresponding channel.

A private (exclusive/auto-delete) queue is then declared and bound to the qpid.management exchange. A binding with key "schema.#" will subscribe to all schema-related information and a second binding with key "console.#" will subscribe to all management data.

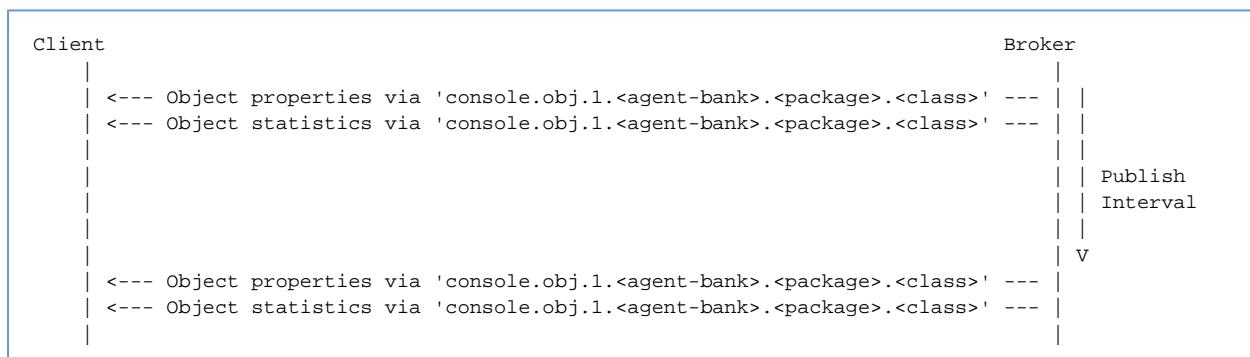
A binding must also be established to the "amq.direct" exchange using the queue's name as the binding key. This will be used as a reply-to address for requests sent to the broker and to agents.

When a client successfully binds to the qpid.management exchange, the management agent schedules a schema broadcast to be sent to the exchange. The agent will publish, via the exchange, a description of the schema for all manageable objects in its control.



Broadcast of Configuration and Instrumentation Updates

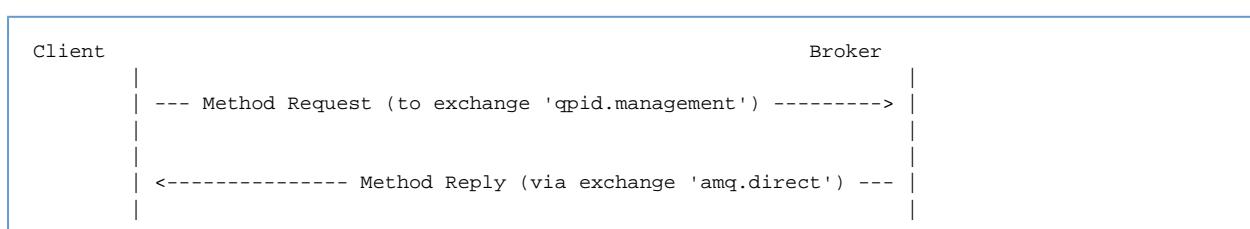
The management agent will periodically publish updates to the configuration and instrumentation of management objects under its control. Under normal circumstances, these updates are published only if they have changed since the last time they were published. Configuration updates are only published if configuration has changed and instrumentation updates are only published if instrumentation has changed. The exception to this rule is that after a management client binds to the qpid.management exchange, all configuration and instrumentation records are published as though they had changed whether or not they actually did.



Invoking a Method on a Managed Object

When the management client wishes to invoke a method on a managed object, it sends a method request message to the qpid.management exchange. The routing key contains the object class and method name (refer to Routing Key Structure below). The method request must have a header entry (reply-to) that contains the name of the method-reply queue so that the method response can be properly routed back to the requestor.

The method request contains a sequence number that is copied to the method reply. This number is opaque to the management agent and may be used by the management client to correlate the reply to the request. The asynchronous nature of requests and replies allows any number of methods to be in-flight at a time. Note that there is no guarantee that methods will be replied to in the order in which they were requested.



Details of QMF Message Types

opcode	message	handled by	description
'B'	Broker Request	broker	This message contains a broker request, sent from the management console to the broker to initiate a management session.
'b'	Broker Response	console	This message contains a broker response, sent from the broker in response to a broker request message.
'z'	Command Completion	all	This message is sent to indicate the completion of a request.
'Q'	Class Query	broker, agent	Class query messages are used by a management console to request a list of schema classes that are known by the management broker.
'q'	Class Indication	console, broker	Sent by the management broker, a class indication notifies the peer of the existence of a schema class.
'S'	Schema Request	broker, agent	Schema request messages are used to request the full schema details for a class.
's'	Schema Response	console, broker	Schema response message contain a full description of the schema for a class.
'h'	Heartbeat Indication	console	This message is published once per publish-interval. It can be used by a client to positively determine which objects did not change during the interval (since updates are not published for objects with no changes).
'c', 'i', 'g'	Content Indication	console	This message contains a content record. Content records contain the values of all properties or statistics in an object. Such records are broadcast on a periodic interval if 1) a change has been made in the value of one of the elements, or 2) if a new management client has bound a queue to the management exchange.
'e'	Event Indication	console	This message contains an event indication, sent by an agent to the topic exchange qpid.management.
'G'	Get Query	agent	Sent by a management console, a get query requests that the management broker provide content indications for all objects that match the query criteria.
'M'	Method Request	agent	This message contains a method request.
'm'	Method Response	console	This message contains a method result.
'P'	Package Query	broker, agent	This message contains a schema package query request, requesting that the broker dump the list of known packages
'p'	Package Indication	console, broker	This message contains a schema package indication, identifying a package known by the broker
'A'	Agent Attach Request	broker	This message is sent by a remote agent when it wishes to attach to a management broker
'a'	Agent Attach Response	agent	The management broker sends this response if an attaching remote agent is permitted to join
'x'	Console Added Indication	agent	This message is sent to all remote agents by the management broker when a new console binds to the management exchange

Broker Request Message

When a management console first establishes contact with the broker, it sends a Broker Request message to initiate the exchange.

```
routing_key: broker
+-----+
| 'A' | 'M' | '2' | 'B' | 0      |
+-----+
```

The Broker Request message has no payload.

Broker Response Message

When the broker receives a Broker Request message, it responds with a Broker Response message. This message contains an identifier

unique to the broker.

```
routing_key: <reply_to from request>
+-----+-----+-----+
| 'A' | 'M' | '2' | 'b' |      0      |
+-----+-----+-----+
| brokerId (uuid)           |
+-----+
```

Command Completion Message

```
routing_key: <reply_to from request>
+-----+-----+-----+
| 'A' | 'M' | '2' | 'z' |      seq      |
+-----+-----+-----+
| Completion Code (uint32)   |
+-----+
| Completion Text (str8)    |
+-----+
```

Class Query

```
routing_key: broker
+-----+-----+-----+
| 'A' | 'M' | '2' | 'Q' |      seq      |
+-----+-----+-----+
| package name (str8)       |
+-----+
```

Class Indication

```
routing_key: <reply_to from request> (if in reply to a request)
          schema.package      (if unsolicited)
+-----+-----+-----+
| 'A' | 'M' | '2' | 'q' |      seq      |
+-----+-----+-----+
| class kind (uint8) 1=Object, 2=Event |
+-----+
| package name (str8)                 |
+-----+
| class name (str8)                 |
+-----+
| schema hash (bin128)              |
+-----+
```

Schema Request

```
routing_key: broker
+-----+-----+-----+
| 'A' | 'M' | '2' | 'S' |      seq      |
+-----+-----+-----+
| packageName (str8)               |
+-----+
| className (str8)                |
+-----+
| schema-hash (bin128)            |
+-----+
```

Schema Response

```

routing_key: <reply_to from request> (if in reply to a request)
          schema.package (if unsolicited)

+-----+-----+-----+
| 'A' | 'M' | '2' | 's' |      seq      |
+-----+-----+-----+-----+
| kind (uint8) 1=Object, 2=Event   |
+-----+-----+-----+
| packageName (str8)               |
+-----+
| className (str8)                |
+-----+
| schema-hash (bin128)            |
+-----+
| propCount (uint16)              |
+-----+
| statCount (uint16)              |
+-----+
| methodCount (uint16)             |
+-----+
| propCount property records      |
+-----+
| statCount statistic records     |
+-----+
| methodCount method records       |
+-----+

```

Each **property** record is an AMQP map with the following fields. Optional fields may optionally be omitted from the map.

field name	optional	description
name	no	Name of the property
type	no	Type code for the property
refPackage	yes	Name of referenced package (for objectReference and object types)
refClass	yes	Name of referenced class (for objectReference and object types)
access	no	Access code for the property
index	no	1 = index element, 0 = not an index element
optional	no	1 = optional element (may be not present), 0 = mandatory (always present)
unit	yes	Units for numeric values (i.e. seconds, bytes, etc.)
min	yes	Minimum value for numerics
max	yes	Maximum value for numerics
maxlen	yes	Maximum length for strings
desc	yes	Description of the property

Each **statistic** record is an AMQP map with the following fields:

field name	optional	description
name	no	Name of the statistic
type	no	Type code for the statistic
unit	yes	Units for numeric values (i.e. seconds, bytes, etc.)
desc	yes	Description of the statistic

method records contain a main map that describes the method or header followed by zero or more maps describing arguments. The main map contains the following fields:

field name	optional	description
name	no	Name of the method or event
argCount	no	Number of argument records to follow
desc	yes	Description of the method or event

Argument maps contain the following fields:

field name	optional	description
name	no	Argument name
type	no	Type code for the argument
refPackage	yes	Name of referenced package (for objectReference and object types)
refClass	yes	Name of referenced class (for objectReference and object types)
dir	yes	Direction code for method arguments
unit	yes	Units for numeric values (i.e. seconds, bytes, etc.)
min	yes	Minimum value for numerics
max	yes	Maximum value for numerics
maxlen	yes	Maximum length for strings
desc	yes	Description of the argument
default	yes	Default value for the argument

type codes are numerics with the following values:

value	type	Encoding
1	uint8	uint8
2	uint16	uint16
3	uint32	uint32
4	uint64	uint64
6	str8	str8
7	str16	str16
8	absTime	uint64
9	deltaTime	uint64
10	objectReference	bin128
11	boolean	boolean
12	float	float
13	double	double
14	uuid	uuid
15	map	map
16	int8	int8
17	int16	int16
18	int32	int32
19	int64	int64
20	object	package-name(str8) + class-name(str8) + hash(bin128) + object-indication-encoding
21	list	list
22	array	array

access codes are numerics with the following values:

value	access
1	Read-Create access
2	Read-Write access
3	Read-Only access

direction codes are strings with the following values:

value	direction
"I"	Input (from client to broker)
"O"	Output (from broker to client)
"IO"	IO (bidirectional)

Heartbeat Indication

```
routing_key: console.heartbeat.1.<agent_bank>
+-----+-----+-----+-----+
| 'A' | 'M' | '2' | 'h' |      0      |
+-----+-----+-----+-----+
| timestamp of current interval (datetime)   |
+-----+-----+
```

Configuration and Instrumentation Content Messages

Content messages are published when changes are made to the values of properties or statistics or when new management clients bind a queue to the management exchange.

```
for 'g':      routing_key: <reply_to from request>
    for 'c','i': routing_key: console.obj.1.<agent_bank>.<package_name>.<class_name>
+-----+-----+-----+-----+
| 'A' | 'M' | '2' | 'g/c/i' |      seq      |
+-----+-----+-----+-----+
|           packageName (str8)          |
+-----+-----+-----+-----+
|           className (str8)          |
+-----+-----+-----+-----+
|           class hash (bin128)        |
+-----+-----+-----+-----+
| timestamp of current sample (datetime)   |
+-----+-----+-----+-----+
| time object was created (datetime)   |
+-----+-----+-----+-----+
| time object was deleted (datetime)   |
+-----+-----+-----+-----+
| objectId (bin128)                   |
+-----+-----+-----+-----+
| presence bitmasks (0 or more uint8 fields) |
+-----+-----+-----+-----+
| config/inst values (in schema order)   |
+-----+-----+
```

All timestamps are uint64 values representing nanoseconds since the epoch (January 1, 1970). The objectId is a bin128 value that uniquely identifies this object instance.

If any of the properties in the object are defined as optional, there will be 1 or more "presence bitmask" octets. There are as many octets as are needed to provide one bit per optional property. The bits are assigned to the optional properties in schema order (first octet first, lowest order bit first).

For example: If there are two optional properties in the schema called "option1" and "option2" (defined in that order), there will be one presence bitmask octet and the bits will be assigned as bit 0 controls option1 and bit 1 controls option2.

If the bit for a particular optional property is set (1), the property will be encoded normally in the "values" portion of the message. If the bit is clear (0), the property will be omitted from the list of encoded values and will be considered "NULL" or "not present".

The element values are encoded by their type into the message in the order in which they appeared in the schema message.

Event Indication Message

```

routing_key: console.event.1.<agent_bank>.<package_name>.<event_name>
+-----+-----+-----+-----+
| 'A' | 'M' | '2' | 'e' |      seq      |
+-----+-----+-----+-----+
| packageName (str8)           |
+-----+-----+
| eventName (str8)            |
+-----+-----+
| class hash (bin128)          |
+-----+-----+
| timestamp of event (datetime) |
+-----+-----+
| severity (uint8)             |
+-----+-----+
| event argument values (in schema order) |
+-----+-----+

```

Get Query Message

A Get Request may be sent by the management console to cause a management agent to immediately send content information for objects of a class.

```

routing_key: agent.1.<agent_bank>
+-----+-----+-----+-----+
| 'A' | 'M' | '2' | 'G' |      seq      |
+-----+-----+-----+-----+
| Get request field table (map) |
+-----+-----+

```

The content of a get request is a field table that specifies what objects are being requested. Most of the fields are optional and are available for use in more extensive deployments.

Field Key	Type	Description
"_class"	str8	The name of the class of objects being requested.
"_package"	str8	The name of the extension package the class belongs to. If omitted, the package defaults to "qpid" for access to objects in the connected broker.
_objectid	bin128	The object ID of the object being requested

When the management agent receives a get request, it sends content messages describing the requested objects. Once the last content message is sent, it then sends a Command Completion message with the same sequence number supplied in the request to indicate to the requestor that there are no more messages coming.

Method Request

Method request messages have the following structure. The sequence number is opaque to the management agent. It is returned unchanged in the method reply so the calling client can correctly associate the reply to the request. The objectId is the unique ID of the object on which the method is to be executed.

```

routing_key: agent.1.<agent_bank>
+-----+-----+-----+-----+
| 'A' | 'M' | '2' | 'M' |      seq      |
+-----+-----+-----+-----+
| objectId (bin128)           |
+-----+-----+
| package name (str8)          |
+-----+-----+
| class name (str8)            |
+-----+-----+
| class hash (bin128)          |
+-----+-----+
| methodName (str8)             |
+-----+-----+
| input and bidirectional argument values (in schema order) |
+-----+-----+

```

Method Response

Method reply messages have the following structure. The sequence number is identical to that supplied in the method request. The status

code (and text) indicate whether or not the method was successful and if not, what the error was. Output and bidirectional arguments are only included if the status code was 0 (STATUS_OK).

```

routing_key: <reply_to from request>
+-----+-----+-----+
| 'A' | 'M' | '2' | 'm' |           seq      |
+-----+-----+-----+
|   status code (uint32)   |
+-----+-----+
|   status text (str16)    |
+-----+-----+
|   output and bidirectional argument values (in schema order)   |
+-----+-----+

```

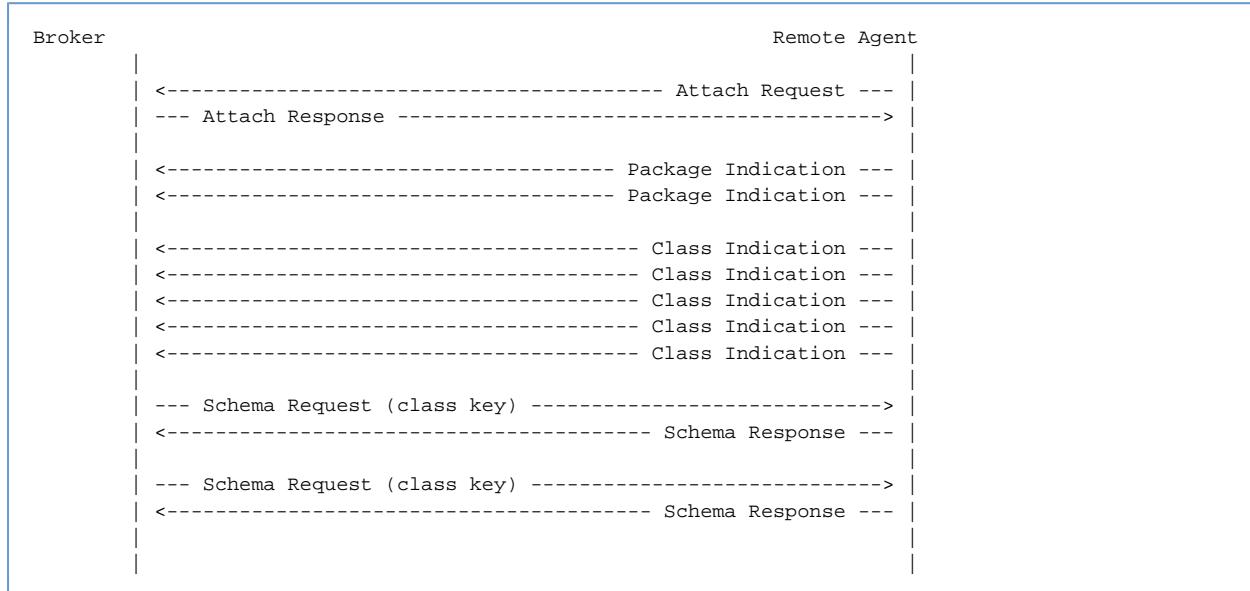
status code values are:

value	description
0	STATUS_OK - successful completion
1	STATUS_UNKNOWN_OBJECT - objectid not found in the agent
2	STATUS_UNKNOWN_METHOD - method is not known by the object type
3	STATUS_NOT_IMPLEMENTED - method is not currently implemented
4	STATUS_INVALID_PARAMETER - input argument is invalid
5	STATUS_FEATURE_NOT_IMPLEMENTED
6	STATUS_FORBIDDEN - operation is forbidden by Access Control List
7	STATUS_EXCEPTION - exception caught during method execution
8	STATUS_UNKNOWN_PACKAGE - package name not found
9	STATUS_UNKNOWN_CLASS - class name not found in package

Messages for Extended Scenario

Extended Management Protocol

Qpid supports management extensions that allow the management broker to be a central point for the management of multiple external entities with their own management schemas.



Package Query

```

routing_key: agent.1.<agent_bank>
+-----+-----+-----+-----+
| 'A' | 'M' | '2' | 'P' |      seq      |
+-----+-----+-----+-----+

```

Package Indication

```

routing_key: <reply_to from request> (if in reply to a request)
          schema.package      (if unsolicited)
+-----+-----+-----+-----+
| 'A' | 'M' | '2' | 'p' |      seq      |
+-----+-----+-----+-----+
| package name (str8)           |
+-----+-----+

```

Agent Attach Request

```

routing_key: broker
+-----+-----+-----+-----+
| 'A' | 'M' | '2' | 'A' |      seq      |
+-----+-----+-----+-----+
| label (str8)                   |
+-----+-----+
| system-id (uuid)              |
+-----+-----+
| requested broker bank (uint32) |
+-----+-----+
| requested agent bank (uint32) |
+-----+-----+

```

Agent Attach Response (success)

```

routing_key: <reply_to from request>
+-----+-----+-----+-----+
| 'A' | 'M' | '2' | 'a' |      seq      |
+-----+-----+-----+-----+
| assigned broker bank (uint32) |
+-----+-----+
| assigned agent bank (uint32) |
+-----+-----+

```

Console Added Indication

```

routing_key: agent.1.<agent_bank>
+-----+-----+-----+-----+
| 'A' | 'M' | '2' | 'x' |      seq      |
+-----+-----+-----+-----+

```

QMF Python Console Tutorial

- Prerequisite - Install Qpid Messaging
- Synchronous Console Operations
 - Creating a QMF Console Session and Attaching to a Broker
 - Accessing Managed Objects
 - Viewing Properties and Statistics of an Object
 - Invoking Methods on an Object
- Asynchronous Console Operations
 - Creating a Console Class to Receive Asynchronous Data
 - Receiving Events
 - Receiving Objects
 - Asynchronous Method Calls and Method Timeouts
- Discovering what Kinds of Objects are Available

Prerequisite - Install Qpid Messaging

QMF uses AMQP Messaging (QPid) as its means of communication. To use QMF, Qpid messaging must be installed somewhere in the

network. Qpid can be downloaded as source from Apache, is packaged with a number of Linux distributions, and can be purchased from commercial vendors that use Qpid. Please see [Download](#) for information as to where to get Qpid Messaging.

Qpid Messaging includes a message broker (qpid) which typically runs as a daemon on a system. It also includes client bindings in various programming languages. The Python-language client library includes the QMF console libraries needed for this tutorial.

Please note that Qpid Messaging has two broker implementations. One is implemented in C++ and the other in Java. At press time, QMF is supported only by the C++ broker.

If the goal is to get the tutorial examples up and running as quickly as possible, all of the Qpid components can be installed on a single system (even a laptop). For more realistic deployments, the broker can be deployed on a server and the client/QMF libraries installed on other systems.

Synchronous Console Operations

The Python console API for QMF can be used in a synchronous style, an asynchronous style, or a combination of both. Synchronous operations are conceptually simple and are well suited for user-interactive tasks. All operations are performed in the context of a Python function call. If communication over the message bus is required to complete an operation, the function call blocks and waits for the expected result (or timeout failure) before returning control to the caller.

Creating a QMF Console Session and Attaching to a Broker

For the purposes of this tutorial, code examples will be shown as they are entered in an interactive python session.

```
$ python
Python 2.5.2 (r252:60911, Sep 30 2008, 15:41:38)
[GCC 4.3.2 20080917 (Red Hat 4.3.2-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

We will begin by importing the required libraries. If the Python client is properly installed, these libraries will be found normally by the Python interpreter.

```
>>> from qmf.console import Session
```

We must now create a `Session` object to manage this QMF console session.

```
>>> sess = Session()
```

If no arguments are supplied to the creation of `Session`, it defaults to synchronous-only operation. It also defaults to user-management of connections. More on this in a moment.

We will now establish a connection to the messaging broker. If the broker daemon is running on the local host, simply use the following:

```
>>> broker = sess.addBroker()
```

If the messaging broker is on a remote host, supply the URL to the broker in the `addBroker` function call. Here's how to connect to a local broker using the URL.

```
>>> broker = sess.addBroker("amqp://localhost")
```

The call to `addBroker` is synchronous and will return only after the connection has been successfully established or has failed. If a failure occurs, `addBroker` will raise an exception that can be handled by the console script.

```
>>> try:
...     broker = sess.addBroker("amqp://localhost:1000")
... except:
...     print "Connection Failed"
...
Connection Failed
>>>
```

This operation fails because there is no Qpid Messaging broker listening on port 1000 (the default port for qpid is 5672).

If preferred, the QMF session can manage the connection for you. In this case, `addBroker` returns immediately and the session attempts to establish the connection in the background. This will be covered in detail in the section on asynchronous operations.

Accessing Managed Objects

The Python console API provides access to remotely managed objects via a *proxy* model. The API gives the client an object that serves as a proxy representing the "real" object being managed on the agent application. Operations performed on the proxy result in the same operations on the real object.

The following examples assume prior knowledge of the kinds of objects that are actually available to be managed. There is a section later in this tutorial that describes how to discover what is manageable on the QMF bus.

Proxy objects are obtained by calling the `Session.getObjects` function.

To illustrate, we'll get a list of objects representing queues in the message broker itself.

```
>>> queues = sess.getObjects(_class="queue", _package="org.apache.qpid.broker")
```

`queues` is an array of proxy objects representing real queues on the message broker. A proxy object can be printed to display a description of the object.

```
>>> for q in queues:  
...     print q  
...  
org.apache.qpid.broker:queue[0-1537-1-0-58]  
0-0-1-0-1152921504606846979:reply-localhost.localdomain.32004  
org.apache.qpid.broker:queue[0-1537-1-0-61]  
0-0-1-0-1152921504606846979:topic-localhost.localdomain.32004  
>>>
```

Viewing Properties and Statistics of an Object

Let us now focus our attention on one of the queue objects.

```
>>> queue = queues[0]
```

The attributes of an object are partitioned into `properties` and `statistics`. Though the distinction is somewhat arbitrary, `properties` tend to be fairly static and may also be large and `statistics` tend to change rapidly and are relatively small (counters, etc.).

There are two ways to view the properties of an object. An array of properties can be obtained using the `getProperties` function:

```
>>> props = queue.getProperties()  
>>> for prop in props:  
...     print prop  
...  
(vhostRef, 0-0-1-0-1152921504606846979)  
(name, u'reply-localhost.localdomain.32004')  
(durable, False)  
(autoDelete, True)  
(exclusive, True)  
(arguments, {})  
>>>
```

The `getProperties` function returns an array of tuples. Each tuple consists of the property descriptor and the property value.

A more convenient way to access properties is by using the attribute of the proxy object directly:

```
>>> queue.autoDelete  
True  
>>> queue.name  
u'reply-localhost.localdomain.32004'  
>>>
```

Statistics are accessed in the same way:

```

>>> stats = queue.getStatistics()
>>> for stat in stats:
...     print stat
...
(msgTotalEnqueues, 53)
(msgTotalDequeues, 53)
(msgTxnEnqueues, 0)
(msgTxnDequeues, 0)
(msgPersistEnqueues, 0)
(msgPersistDequeues, 0)
(msgDepth, 0)
(byteDepth, 0)
(byteTotalEnqueues, 19116)
(byteTotalDequeues, 19116)
(byteTxnEnqueues, 0)
(byteTxnDequeues, 0)
(bytePersistEnqueues, 0)
(bytePersistDequeues, 0)
(consumerCount, 1)
(consumerCountHigh, 1)
(consumerCountLow, 1)
(bindingCount, 2)
(bindingCountHigh, 2)
(bindingCountLow, 2)
(unackedMessages, 0)
(unackedMessagesHigh, 0)
(unackedMessagesLow, 0)
(messageLatencySamples, 0)
(messageLatencyMin, 0)
(messageLatencyMax, 0)
(messageLatencyAverage, 0)
>>>

```

or alternatively:

```

>>> queue.byteTotalEnqueues
19116
>>>

```

The proxy objects do not automatically track changes that occur on the real objects. For example, if the real queue enqueues more bytes, viewing the `byteTotalEnqueues` statistic will show the same number as it did the first time. To get updated data on a proxy object, use the `update` function call:

```

>>> queue.update()
>>> queue.byteTotalEnqueues
19783
>>>

```



Be Advised

The `update` method was added after the M4 release of Qpid/Qmf. It may not be available in your distribution.

Invoking Methods on an Object

Up to this point, we have used the QMF Console API to find managed objects and view their attributes, a read-only activity. The next topic to illustrate is how to invoke a method on a managed object. Methods allow consoles to control the managed agents by either triggering a one-time action or by changing the values of attributes in an object.

First, we'll cover some background information about methods. A `QMF object class` (of which a `QMF object` is an instance), may have zero or more methods. To obtain a list of methods available for an object, use the `getMethods` function.

```

>>> methodList = queue.getMethods()

```

`getMethods` returns an array of method descriptors (of type `qmf.console.SchemaMethod`). To get a summary of a method, you can simply print it. The `_repr_` function returns a string that looks like a function prototype.

```
>>> print methodList
[purge(request)]
>>>
```

For the purposes of illustration, we'll use a more interesting method available on the *broker* object which represents the connected Qpid message broker.

```
>>> br = sess.getObjects(_class="broker", _package="org.apache.qpid.broker")[0]
>>> mlist = br.getMethods()
>>> for m in mlist:
...     print m
...
echo(sequence, body)
connect(host, port, durable, authMechanism, username, password, transport)
queueMoveMessages(srcQueue, destQueue, qty)
>>>
```

We have just learned that the *broker* object has three methods: *echo*, *connect*, and *queueMoveMessages*. We'll use the *echo* method to "ping" the broker.

```
>>> result = br.echo(1, "Message Body")
>>> print result
OK (0) - {'body': u'Message Body', 'sequence': 1}
>>> print result.status
0
>>> print result.text
OK
>>> print result.outArgs
{'body': u'Message Body', 'sequence': 1}
>>>
```

In the above example, we have invoked the *echo* method on the instance of the broker designated by the proxy "br" with a sequence argument of 1 and a body argument of "Message Body". The result indicates success and contains the output arguments (in this case copies of the input arguments).

To be more precise... Calling *echo* on the proxy causes the input arguments to be marshalled and sent to the remote agent where the method is executed. Once the method execution completes, the output arguments are marshalled and sent back to the console to be stored in the method result.

You are probably wondering how you are supposed to know what types the arguments are and which arguments are input, which are output, or which are both. This will be addressed later in the "Discovering what Kinds of Objects are Available" section.

Asynchronous Console Operations

QMF is built on top of a middleware messaging layer (Qpid Messaging). Because of this, QMF can use some communication patterns that are difficult to implement using network transports like UDP, TCP, or SSL. One of these patterns is called the *Publication and Subscription* pattern (pub-sub for short). In the pub-sub pattern, data sources *publish* information without a particular destination in mind. Data sinks (destinations) *subscribe* using a set of criteria that describes what kind of data they are interested in receiving. Data published by a source may be received by zero, one, or many subscribers.

QMF uses the pub-sub pattern to distribute events, object creation and deletion, and changes to properties and statistics. A console application using the QMF Console API can receive these asynchronous and unsolicited events and updates. This is useful for applications that store and analyze events and/or statistics. It is also useful for applications that react to certain events or conditions.

Note that console applications may always use the synchronous mechanisms.

Creating a Console Class to Receive Asynchronous Data

Asynchronous API operation occurs when the console application supplies a *Console* object to the session manager. The *Console* object (which overrides the *qmf.console.Console* class) handles all asynchronously arriving data. The *Console* class has the following methods. Any number of these methods may be overridden by the console application. Any method that is not overridden defaults to a null handler which takes no action when invoked.

Method	Arguments	Invoked when...
brokerConnected	broker	a connection to a broker is established
brokerDisconnected	broker	a connection to a broker is lost
newPackage	name	a new package is seen on the QMF bus
newClass	kind, classKey	a new class (event or object) is seen on the QMF bus

newAgent	agent	a new agent appears on the QMF bus
delAgent	agent	an agent disconnects from the QMF bus
objectProps	broker, object	the properties of an object are published
objectStats	broker, object	the statistics of an object are published
event	broker, event	an event is published
heartbeat	agent, timestamp	a heartbeat is published by an agent
brokerInfo	broker	information about a connected broker is available to be queried
methodResponse	broker, seq, response	the result of an asynchronous method call is received

Supplied with the API is a class called *DebugConsole*. This is a test *Console* instance that overrides all of the methods such that arriving asynchronous data is printed to the screen. This can be used to see all of the arriving asynchronous data.

Receiving Events

We'll start the example from the beginning to illustrate the reception and handling of events. In this example, we will create a *Console* class that handles broker-connect, broker-disconnect, and event messages. We will also allow the session manager to manage the broker connection for us.

Begin by importing the necessary classes:

```
>>> from qmf.console import Session, Console
```

Now, create a subclass of *Console* that handles the three message types:

```
>>> class EventConsole(Console):
...     def brokerConnected(self, broker):
...         print "brokerConnected:", broker
...     def brokerDisconnected(self, broker):
...         print "brokerDisconnected:", broker
...     def event(self, broker, event):
...         print "event:", event
...
>>>
```

Make an instance of the new class:

```
>>> myConsole = EventConsole()
```

Create a *Session* class using the console instance. In addition, we shall request that the session manager do the connection management for us. Notice also that we are requesting that the session manager not receive objects or heartbeats. Since this example is concerned only with events, we can optimize the use of the messaging bus by telling the session manager not to subscribe for object updates or heartbeats.

```
>>> sess = Session(myConsole, manageConnections=True, rcvObjects=False, rcvHeartbeats=False)
>>> broker = sess.addBroker()
>>>
```

Once the broker is added, we will begin to receive asynchronous events (assuming there is a functioning broker available to connect to).

```
brokerConnected: Broker connected at: localhost:5672
event: Thu Jan 29 19:53:19 2009 INFO org.apache.qpid.broker:bind broker=localhost:5672 ...
```

Receiving Objects

To illustrate asynchronous handling of objects, a small console program is supplied. The entire program is shown below for convenience. We will then go through it part-by-part to explain its design.

This console program receives object updates and displays a set of statistics as they change. It focuses on broker queue objects.

```

# Import needed classes
from qmf.console import Session, Console
from time import sleep

# Declare a dictionary to map object-ids to queue names
queueMap = {}

# Customize the Console class to receive object updates.
class MyConsole(Console):

    # Handle property updates
    def objectProps(self, broker, record):

        # Verify that we have received a queue object.  Exit otherwise.
        classKey = record.getClassKey()
        if classKey.getClassName() != "queue":
            return

        # If this object has not been seen before, create a new mapping from objectID to name
        oid = record.getObjectId()
        if oid not in queueMap:
            queueMap[oid] = record.name

    # Handle statistic updates
    def objectStats(self, broker, record):

        # Ignore updates for objects that are not in the map
        oid = record.getObjectId()
        if oid not in queueMap:
            return

        # Print the queue name and some statistics
        print "%s: enqueue=%d dequeue=%d" % (queueMap[oid], record.msgTotalEnqueues,
                                              record.msgTotalDequeues)

        # if the delete-time is non-zero, this object has been deleted. Remove it from the map.
        if record.getTimestamps()[2] > 0:
            queueMap.pop(oid)

    # Create an instance of the QMF session manager. Set userBindings to True to allow
    # this program to choose which objects classes it is interested in.
    sess = Session(MyConsole(), manageConnections=True, rcvEvents=False, userBindings=True)

    # Register to receive updates for broker:queue objects.
    sess.bindClass("org.apache.gpid.broker", "queue")
    broker = sess.addBroker()

    # Suspend processing while the asynchronous operations proceed.
    try:
        while True:
            sleep(1)
    except:
        pass

    # Disconnect the broker before exiting.
    sess.delBroker(broker)

```

Before going through the code in detail, it is important to understand the differences between synchronous object access and asynchronous object access. When objects are obtained synchronously (using the `getObjects` function), the resulting proxy contains all of the object's attributes, both properties and statistics. When object data is published asynchronously, the properties and statistics are sent separately and only when the session first connects or when the content changes.

The script wishes to print the queue name with the updated statistics, but the queue name is only present with the properties. For this reason, the program needs to keep some state to correlate property updates with their corresponding statistic updates. This can be done using the `ObjectId` that uniquely identifies the object.

```

# If this object has not been seen before, create a new mapping from objectID to name
oid = record.getObjectId()
if oid not in queueMap:
    queueMap[oid] = record.name

```

The above code fragment gets the object ID from the proxy and checks to see if it is in the map (i.e. has been seen before). If it is not in the

map, a new map entry is inserted mapping the object ID to the queue's name.

```
# if the delete-time is non-zero, this object has been deleted. Remove it from the map.  
if record.getTimestamps()[2] > 0:  
    queueMap.pop(oid)
```

This code fragment detects the deletion of a managed object. After reporting the statistics, it checks the timestamps of the proxy. `getTimestamps` returns a list of timestamps in the order:

- **Current** - The timestamp of the sending of this update.
- **Create** - The time of the object's creation
- **Delete** - The time of the object's deletion (or zero if not deleted)

This code structure is useful for getting information about very-short-lived objects. It is possible that an object will be created, used, and deleted within an update interval. In this case, the property update will arrive first, followed by the statistic update. Both will indicate that the object has been deleted but a full accounting of the object's existence and final state is reported.

```
# Create an instance of the QMF session manager. Set userBindings to True to allow  
# this program to choose which objects classes it is interested in.  
sess = Session(MyConsole(), manageConnections=True, rcvEvents=False, userBindings=True)  
  
# Register to receive updates for broker:queue objects.  
sess.bindClass("org.apache.qpid.broker", "queue")
```

The above code is illustrative of the way a console application can tune its use of the QMF bus. Note that `rcvEvents` is set to False. This prevents the reception of events. Note also the use of `userBindings=True` and the call to `sess.bindClass`. If `userBindings` is set to False (its default), the session will receive object updates for all classes of object. In the case above, the application is only interested in `broker:queue` objects and reduces its bus bandwidth usage by requesting updates to only that class. `bindClass` may be called as many times as desired to add classes to the list of subscribed classes.

Asynchronous Method Calls and Method Timeouts

Method calls can also be invoked asynchronously. This is useful if a large number of calls needs to be made in a short time because the console application will not need to wait for the complete round-trip delay for each call.

Method calls are synchronous by default. They can be made asynchronous by adding the keyword-argument `_async=True` to the method call.

In a synchronous method call, the return value is the method result. When a method is called asynchronously, the return value is a sequence number that can be used to correlate the eventual result to the request. This sequence number is passed as an argument to the `methodResponse` function in the `Console` interface.

It is important to realize that the `methodResponse` function may be invoked before the asynchronous call returns. Make sure your code is written to handle this possibility.

Discovering what Kinds of Objects are Available

QMFv2 Project Page

QMFv2 Project Page

Introduction

This developer page shall be used to collect architecture, design, and project information for the development of the Qpid Management Framework. It will become the basis for a distilled, user-oriented set of documentation.

- Architecture
- Protocol
- [Design]
- APIs

Changes from Version 1

- **Broker participation in QMF is no longer mandatory.** In QMFv1, the broker provides two essential services for QMF: Agent registration and Schema caching. In QMFv2, the need for agent registration has been removed. Schema caching remains a valuable optimization but is not required in QMFv2.
- **Message body formats are based on AMQP maps.** In QMFv1, message bodies were formatted as packed binary data (using the AMQP type encodings). In QMFv2, message bodies are AMQP maps and are therefore easily extended without causing backward compatibility problems. Another benefit of the map-message format is that all messages can be fully parsed when received without the need for external schema data. In QMFv1, messages cannot be decoded unless the schema for the data is already known.
- **Federation is supported.** In QMFv1, messages cannot be transferred over federation links between brokers. Each broker

represents its own isolated QMF domain. In QMFv2, agents and consoles can connect to any broker in a federated network and interact with other agents and consoles anywhere in the network.

- **Agent and Object identification is more flexible.** Agent and object identification in QMFv1 uses fixed-length binary data fields. In QMFv2, variable-length strings are used. QMFv2 agents choose their own globally unique identifiers (either through configuration or by embedding a UUID into the ID). Managed objects are identified by the ID of the owning agent plus a unique (to the agent) string identifying the object.
- **Agents no longer publish data globally.** Global publishing of data by agents causes security and scaling issues. In QMFv2, the global publish is replaced by subscription queries where a console establishes a subscription and receives periodic updates about changes to data in its field of interest. The benefits are that the query can be authorized based on an authenticated user-id and that data is not generated if there are not consoles interested in receiving that data.
- **QMF uses native AMQP Types** The base types in QMFv2 are the same as those supported by AMQP. There is no additional type-id space as there is in QMFv1.

Backward/Forward Compatibility with QMFv1

The following compatibility matrix shows all combinations of V1 and V2 components (console, agent, and broker). Those intersections marked "OK" are supported without the need for compatibility-oriented development.

V1 Console	V2 Console		
	V1 Broker	V2 Broker	
V1 Agent	OK	note 3	V1 Broker
	note 1	note 2	V2 Broker
V2 Agent	note 3	OK	V1 Broker
	note 4	OK	V2 Broker

The following notes address how intersections in the matrix might be supported.

Note 1: V2 broker retains V1 capability

Note 2: V2 broker proxies V1 agents (including the embedded one) to V2 protocol

Note 3: V2 clients speak both protocols, choosing which to use by the version of the connected broker

Note 4: V2 broker proxies V2 agents to V1 protocol

Planning:

- The features in notes 1 and 2 **must** be implemented. It is important that users not be required to upgrade all components of their deployments at the same time.
- The feature in note 3 **may** be implemented. This is considered lower priority.
- The feature in note 4 **shall not** be implemented. Because QMF V2 is intended to scale to larger networks than are currently possible with V1, and this feature would limit the scalability of QMF V2, this is considered non-desirable.

Development Roadmap

QMFv2 APIs

QMFv2 APIs

- [QMFv2 API Proposal](#) - A working draft of the general API layout

Language Bindings

Python

Ruby

C++

Java

POJO

JMX

Windows

QMFv2 API Proposal

Proposal for "Next Generation" QMF API

Goals

- Simplify QMF API
- Use new QPID Messaging API
- Implement QMF protocol via QPID Map messages
- Improve thread safety by minimizing the callback notification mechanism.
- Move to a work-queue based event model.

Component Addressing

QMF uses AMQP messaging as the means for communications between Console and Agent components. Thus instances of Agents and Consoles must have addresses which uniquely identify them within the AMQP messaging domain.

QMF uses AMQP string types to represent Agent and Console addresses. These strings are assigned by the application, and their contents are opaque to the QMF implementation.

A QMF address is composed of two parts - an optional *domain* string, and a mandatory *name* string. The domain string is used to construct the name of the AMQP exchange to which the component's name string will be bound. If not supplied, the value of the domain defaults to "default". Both Agents and Components must belong to the same domain in order to communicate.

When a Console or Agent is instantiated, it will create a receiving endpoint (source) on the AMQP message domain. The endpoint's address will be sent as the *reply_to* field for all messages originating from that Console or Agent. The address of the endpoint is created from the value of the domain and name strings, using the following format:

```
"qmfc.<domain-string>.direct/<name-string>"
```

Data Model

Representation of Management Data

The QMF data model supports two methods for representing management data:

1. arbitrarily structured data
2. data with a formally defined structure

Arbitrarily structured data is the simplest method for representing data under QMF. It consists of a set of named data values. Each data value is represented by a primitive AMQP data type. The data is accessed using the data's name as a key. QMF represents arbitrarily structured data as a map of AMQP data types indexed by a name string.

Data that has a formally defined structure extends the data representation by associating the data with a Schema. The Schema describes the structure of all instances of data that are based on that Schema. This is akin to a record type in database design.

Both types of data representations can be managed by an agent. Managed data includes:

1. an object identifier
2. object lifecycle state

An object identifier uniquely addresses a data object within the domain of its managing agent. QMF represents the object identifier as an opaque string. An object identifier can be assigned to the object by the agent when the object is instantiated. Alternatively, a schema can define an object identifier by defining an ordered list of names of data items. In this case, the object identifier string is built by concatenating the string representations of the value of each named data item. This approach is similar to defining index fields within a database record.

For example, assume a managed object with the following map of data item values:

```
{"field1" : "foo",
 "field2" : "bar",
 "field3" : 42,
 "field4" : "baz"}
```

and assume the data item list defined by the managed object's schema is:

```
[ "field1", "field3" ]
```

The identifier for this data object would be:

```
"foo42"
```

QmfData Class

QMF defines the QmfData class to represent an atomic unit of management data. The QmfData class defines a collection of named data values. Optionally, a string tag, or "sub-type" may be associated with each data item. This tag may be used by an application to annotate the data item.

When representing formally defined data, a QmfData instance is assigned a Schema.

When representing managed data, a QmfData instance is assigned an object identifier (either explicitly, or via the Schema).

```
class QmfData:  
    <constructor>(  
        _values=map of "name"=<AMQP Type> pairs,  
        _subtypes=optional map of "name"=<AMQP String> pairs for subtype information,  
        _tag=optional, application-specific tag applied to this QmfData instance  
        _object_id=optional AMQP string that uniquely identifies this QmfData  
    instance.  
        _schema=optional <class SchemaClass> reference  
        _const=False )  
    <constructor>(  
        _map=map representation of QmfData, as generated by mapEncode() method,  
        _schema=optional <class SchemaClass> reference  
        _const=False)  
    .is_managed(): returns True if object identifier string assigned, else False.  
    .is_described(): returns True if schema is associated with this instance  
    .get_tag(): return this object's tag if present, else None  
    .get_value(name): return the value of the named data item, returns None if named property is  
    not present.  
    .has_value(name): returns True if the named data item is present in the map, else False.  
    .set_value(name, value, subType=None): set the value of the named data item. Creates a new  
    item if the named data does not exist. Raises an exception if _const is True.  
    .get_subtype(name): returns the sub type description string, else None if not present.  
    .set_subtype(name, subType): set or modify the subtype associated with name.  
    .get_object_id(): returns the object id string associated with the object instance, or None  
    if no id assigned.  
    .get_schema_class_id(): returns the identifier of the Schema that describes the structure of  
    the data, or None if no schema.  
    .map_encode(): returns a map representation of the QmfData instance, suitable for use by the  
    constructor.
```

QMF uses a map encoding to represent a QmfData class in an AMQP message. The map encoding of an instance of the QmfData Class is made up of the following elements:

Index	Optional	Type	Description
"_values"	N	map	Map containing all the "name"=<AMQP Type> pairs for this object.
"_subtype"	Y	map	Map containing all "name"=<AMQP String> subtype entries for this object.
"_tag"	Y	Any AMQP-supported type	Application-specific tag for this object.
"_object_id"	Y	string	Unique identifier for this data object.
"_schema_id"	Y	map	Map containing the SchemaClassId for this object.

QmfEvent Class

QMF supports event management functionality. An event is a notification that is sent by an Agent to alert Console(s) of a change in some aspect of the system under management. Like QMF Data, Events may be formally defined by a Schema or not. Unlike QMF Data, Events are not manageable entities - they have no lifecycle. Events simply indicate a point in time where some interesting action occurred.

An instance of an event is represented by the QmfEvent class.

An AMQP timestamp value is associated with each QmfEvent instance. It indicates the moment in time the event occurred. This timestamp is mandatory.

A *severity level* may be associated with each QmfEvent instance. The following severity levels are supported:

- "emerg" - system is unusable
- "alert" - action must be taken immediately

- "crit" - the system is in a critical condition
- "err" - there is an error condition
- "warning" - there is a warning condition
- "notice" - a normal but significant condition
- "info" - a purely informational message
- "debug" - messages generated to debug the application

The default severity is "notice".

```
class QmfEvent(QmfData):
    <constructor>(_timestamp,
                  _severity=<string>,
                  _values=map of "name"=<AMQP Type> pairs,
                  _subtypes=optional map of "name"=<AMQP String> pairs for subtype information,
                  _tag=optional, application-specific tag applied to this QmfEvent instance
                  _schema=optional <class SchemaEventClass> )
    <constructor>(_map= map encoding of a QmfEvent instance,
                  _schema=optional <class SchemaEventClass> )
    .get_timestamp(): return a timestamp indicating when the Event occurred.
    .get_severity(): return the severity associated with the Event.
    .map_encode(): return a map encoding of the Event.
```

The map encoding of an instance of the QmfEvent class extends the map of the parent class with the following class properties:

Index	Optional	Type	Description
"_timestamp"	N	AMQP Timestamp	Time the event occurred.
"_severity"	Y	string	Event severity

Schema

Schemas are used by QMF to describe the structure of management data and events. The use of Schemas is optional.

Schema Types

There are two classes (or types) of Schema - those that describe data objects, and those that describe event objects.

```
SchemaTypeData:
SchemaTypeEvent:
```

These types may be represented by the strings "_data" and "_event", respectively.

Schema Identifier

Schema are identified by a combination of their package name and class name. A hash value over the body of the schema provides a revision identifier. The class SchemaClassId represents this Schema identifier.

```
class SchemaClassId:
    <constructor>(_package=<package-name-str>,
                  _class=<class-name-str>,
                  _type=<SchemaTypeData|SchemaTypeEvent>)
    _hash=<hash-str, format="%08x-%08x-%08x-%08x">
    .get_package_name(): returns <package-name-str>
    .get_class_name(): returns <class-name-str>
    .get_hash_string(): returns <hash-str, "%08x-%08x-%08x-%08x">
    .get_type(): returns SchemaTypeObject or SchemaTypeEvent
    .map_encode(): returns a map encoding of the instance.
```

If the hash value is not supplied, then the value of the hash string will be set to None. This will be the case when a SchemaClass is being dynamically constructed, and a proper hash is not yet available.

The map encoding of a SchemaClassId:

Index	Optional	Type	Description
"_package_name"	N	string	The name of the associated package.
"_class_name"	N	string	The name of the class within the package.
"_type"	N	string	The type of schema, either "data" or "event".

"_hash_str"	Y	string	The MD5 hash of the schema, in the format "%08x-%08x-%08x-%08x"
-------------	---	--------	---

Schema For Describing The Properties of a Data Item

The SchemaProperty class describes a single data item in a QmfData object. A SchemaProperty is a list of named attributes and their values. QMF defines a set of primitive attributes. An application can extend this set of attributes with application-specific attributes.

QMF reserved attribute names all start with the underscore character ("_"). Do not create an application-specific attribute with a name starting with an underscore.

Once instantiated, the SchemaProperty is immutable.

```
class SchemaProperty:
    <constructor>( name=<name-value>,
                    type=<type-value>,
                    ... )
    .get_type(): AMQP typecode for encoding/decoding the property data
    .get_access(): "RC"=read/create, "RW"=read/write, "RO"=read only (default)
    .is_optional(): True if this property is optional
    .get_unit(): string describing units (optional)
    .get_min(): minimum value (optional)
    .get_max(): maximum value (optional)
    .get_max_len(): maximum length for string values (optional)
    .get_desc(): optional string description of this Property
    .get_direction(): "I"=input, "O"=output, or "IO"=input/output (required for method
arguments, otherwise optional)
    .get_subtype(): string indicating the formal application type for the data, example: "URL",
"Telephone number", etc.
    .is_polled(): True if changes to the data cannot be practically monitored by an Agent. Such
a data item can only
        be queried or polled - not published on change.
    .get_reference(): if type==objId, name (str) of referenced class (optional)
    .is_parent_ref(): True if this property references an object in which this object is in a
child-parent relationship.
    .get_attribute("name"): get the value of the attribute named "name". This method can be used
to retrieve
        application-specific attributes. "name" should start with the prefix "x-"
    .map_encode(): returns a map encoding of the instance.
```

The map encoding of a SchemaProperty's body:

Index	Optional	Type	Description
"_qmf_type"	N	integer	The QMF type code indicating the value's data type.
"_access"	N	string	The access allowed to this property, default "RO"
"_optional"	N	boolean	True if this property is optional, default False
"_unit"	Y	string	Description of the units used to express this property.
"_min"	Y	integer	The minimum allowed value for this property
"_max"	Y	integer	The maximum allowed value for this property
"_ maxlen"	Y	integer	For string types, this is the maximum length in bytes required to represent the longest permissible instance of this string.
"_desc"	Y	string	Human-readable description of this property.
"_dir"	Y	string	Direction for an argument when passed to a Method call: "I", "O", "IO", default value: "I"
"_subtype"	Y	string	Type information for use by the application.
"_polled"	Y	boolean	True if this property can only be queried/polled. Default False.
"_reference"	Y	string	unknown?
"_parent_ref"	Y	boolean	True if this property references an object in which this object is in a child-parent relationship. Default False
"x-<varies>"	Y	Any AMQP type	An application-defined attribute.

Schema For Describing Method Calls

The SchemaMethod class describes a method call's parameter list. The parameter list is represented by an unordered map of SchemaProperty entries indexed by parameter name.

```

class SchemaMethod:
    <constructor>([args=<map of "name":<SchemaProperty> entries>],
                  desc="description of the method")
    .get_desc(): return human-readable description of the method.
    .get_argument_count(): return the number of arguments
    .get_arguments(): returns a copy of the map of "name":<SchemaProperty>
    .get_argument("name"): return the SchemaProperty for the parameter "name"

    .add_argument("name", SchemaProperty): adds an additional argument to the parameter list.
    .map_encode(): returns a map encoding of the SchemaMethod instance

```

The map encoding of a SchemaMethod:

Index	Optional	Type	Description
"_desc"	Y	string	Description of the method.
"_arguments"	Y	map	Map of "name":<SchemaProperty> values, one for each argument to the method.

Note that the "dir" SchemaProperty attribute applies to each argument. If "dir" is not specified, it is assumed to be "I".

Schema for Data Objects and Events

The structure of QmfData objects is formally defined by the class SchemaObjectClass.

The structure of QmfEvent objects is formally defined by the class SchemaEventClass.

Both of these classes derive from the virtual base SchemaClass.

Agent applications may dynamically construct instances of these objects by adding properties and methods at run time. However, once the Schema is made public, it must be considered immutable, as the hash value must be constant once the Schema is in use.

QMF defines the following classes to represent data and event schema:

```

class SchemaClass(QmfData):
    <constructor>([classId=<class SchemaClassId>,
                  _desc=optional AMQP string containing a human-readable description for this
schema)
    .get_class_id(): return the SchemaClassId that identifies this Schema instance.
    .generate_hash(): generate a hash over the body of the schema, and return a string
representation of the hash in format "%08x-%08x-%08x-%08x"

```

The map encoding of an instance of the SchemaClass class extends the map of its parent class with elements for the following class properties.

Index	Optional	Type	Description
"_schema_id"	N	map	Map containing the SchemaClassId for this object.

```

class SchemaObjectClass(SchemaClass):
    <constructor>(< classId=<class SchemaClassId>,
                  _desc=optional AMQP string containing a human-readable description for this
schema,
                  _props=map of "name"=<SchemaProperty> instances,
                  _meths=map of "name"=<SchemaMethod> instances,
                  _id_names=(optional) ordered list of "name"s used to identify the properties
whose values are used to construct the object identifier)

    .get_id_names(): return an ordered list of names of the values that are used to construct
the key for identifying
        unique instances of this class. Returns None if no list defined.
    .get_property_count(): return the count of SchemaProperty's in this instance.
    .get_properties(): return a map of "name":<SchemaProperty> entries for each value in the
object.
    .get_property("name"): return the SchemaProperty associated with "name", else None if "name"
value does not exist.

    .get_method_count(): return the count SchemaMethod's in this instance.
    .get_methods(): return a map of "name":<SchemaMethod> entries for each method associated
with the object.
    .get_method("name"): return the SchemaMethod associated with the "name", else None if "name"
does not exist or is not a method.

    .add_property("name", SchemaProperty): add a new property.
    .add_method("name", SchemaMethod): add a new method.
    .set_id_names([name-list]): set the value of the order list of names to use when
constructing the object identifier.

```

The map encoding of an instance of the SchemaObjectClass class extends the map of its parent class with elements for the following class properties.

Index	Optional	Type	Description
"_primary_key"	Y	list	Order list of property names used to construct the primary key for objects defined by this schema

```

class SchemaEventClass(SchemaClass):
    <constructor>(< classId=<class SchemaClassId>,
                  _desc=optional AMQP string containing a human-readable description for this
event,
                  _props=map of "name":SchemaProperty instances)
    .get_property_count(): return the number of properties.
    .get_properties(): return a map of "name":<SchemaProperty> entries for each property in the
event.
    .get_property("name"): return the SchemaProperty associated with "name".

    .add_property("name", SchemaProperty): add a new property.

```

The map encoding of a SchemaEventClass instance uses the same format as the map for the SchemaClass.

Data Management

QMF allows data management via the Query, Subscription, and Method Call actions.

Queries

A Query is a mechanism for interrogating the management database. A Query represents a selector which is sent to an Agent. The Agent applies the Query against its management database, and returns those objects which meet the constraints described in the query.

A Query must specify the class of information it is selecting. This class of information is considered the *target* of the query. Any data objects selected by the query will be of the type indicated by the *target*.

A Query may also specify a *selector* which is used as a filter against the set of all *target* instances. Only those instances accepted by the filter will be returned in response to the query.

Queries are expressed using the following map syntax:

```
{
  "query": {"what":<target>,
            <selector_type>:<selector_params>}
}
```

Where:

The value of the "what" key is a map that specifies the *target* for the Query. The Agent will return a list of instances of *target* types that match the query.

<*target*> is implemented as a map with a single element in the format:

```
{ "<target name string>": <optional map of target qualifiers> }
```

QMF defines the following values for <*target name string*>:

Target	Description
"schema_package"	Query against the set of known packages. Returns a list of package name strings.
"schema_id"	Query against the set of SchemaClass objects, return a list of SchemaClassId instances for the objects that match.
"schema"	Query against the set of SchemaClass objects, and return a list of matched SchemaClass instances.
"object_id"	Query against the set of managed QmfData objects, return a list of name strings for all matching instances.
"object"	Query against the set of managed objects, return a list of matching QmfData instances.
"agent"	Query against all agents within the current QMF domain, return a list of name strings for each matching agent. Used only by the "agent-locate" message.
more tbd

The value of the <*target name string*> map entry is ignored for now, its use is TBD.

The *selector_type* is optional. If it is not present, then the Query has no filter. The Agent will return every instance of type *target* it has. If *selector_type* is present, it may be one of the following supported values:

- "id" - an exact match against the *target*'s unique identifier.
- "where" - a logical expression to apply to the *target*.
- ...more tbd...

Only one *selector_type* is allowed in the Query at a time.

"id" Selector

A Query with an *id* selector is used to find one particular instance of a given *target* type. The value of the "id" keyword is a type that suitably represents an exact identifier for the *target* type. The Agent will return the first instance that matches the identifier.

id selectors are only valid for the following subset of *targets*:

Target	Selector Description
"schema"	A SchemaClassId
"object"	String containing the object identifier
"agent"	String containing the name of the agent

The Agent will return no matches should a Query using an *id* selector specify a target name not in the above subset.

"Where" Selector

A Query with a *where* selector describes a logical expression that is used to filter the target set. This logical expression is built from a set of boolean operations that are applied against the target's data. These boolean operations may be combined using logical operations.

The Agent will apply the logical expression against every instance of the target type, and return the set of instances for which the expression evaluates as True.

The value of the "where" keyword is a list representation of a predicate expression. QMF will support the following syntax for predicate expressions:

```

PREDICATE-EXP: [LOGIC-OP LOGIC-ARG]
LOGIC-ARG: PREDICATE-EXP
LOGIC-ARG: BOOL-EXP LOGIC-ARG
LOGIC-ARG: BOOL-EXP

BOOL-EXP: [BOOL-OP BOOL-ARG]
BOOL-ARG: VARIABLE BOOL-ARG
BOOL-ARG: CONST BOOL-ARG
BOOL-ARG:

CONST: [QUOTE NAME]
CONST: [QUOTE ATOMIC]
CONST: ATOMIC

VARIABLE: [UNQUOTE NAME]
VARIABLE: NAME

ATOMIC: any non-string type
NAME: string

QUOTE: "quote"
UNQUOTE: "unquote"

LOGIC-OP: string
BOOL-OP: string

```

In the above expressions, the left and right braces indicate lists. For example, BOOL-EXP is a list containing a BOOL-OP followed by BOOL-ARG.

A BOOL-EXP is a boolean test that is applied against the target. BOOL-OP defines the boolean operation that is performed on the arguments. QMF will define a set of string literals representing the supported boolean operations. At minimum, the following boolean operators are defined:

- "eq" - equality
- "ne" - inequality
- "lt" - arithmetical/lexical less-than
- "le" - arithmetical/lexical less-than-or-equal
- "gt" - arithmetical/lexical greater-than
- "ge" - arithmetical/lexical greater-than-or-equal
- "re_match" - string regular expression match (one argument)
- "exists" - True if named value is present in target, else False (one argument)
- "true" - always true (no arguments)
- "false" - always false (no arguments)
- <more tbd>

All operators are binary unless otherwise noted.

LOGIC-OP defines the logical operation that is applied to its arguments. QMF will define a set of string literals representing supported logical operations. At minimum, the following logical operators will be defined:

- "and" - logical AND, all arguments must evaluate to True.
- "or" - logical OR, at least one argument must evaluate to True
- "not" - logical NOT, all arguments must evaluate to False
- <more tbd>

Logic operations are "short-circuiting". That is, evaluation ceases as soon as the truth value of the expression is determined. For example, the evaluation of an "and" expression stops when the first argument that evaluates as False is found.

QMF considers string arguments in boolean expressions to be names of data values in the target object. When evaluating a predicate expression, QMF will fetch the value of the named data item from each candidate target object. The value is then used in the boolean expression. In other words, QMF considers string arguments to be variables in the expression. In order to indicate that a string should be treated as a literal instead, the string must be quoted using the "quote" expression.

For example, the following boolean expression contains the data item named "employee" against the string literal "Joey Jojo":

```
[ "eq" "employee" [ "quote" "Joey Jojo" ] ]
```

In implementation, predicate expressions are nested lists. The first element of all lists is the operator keyword string. The remaining list elements are arguments to the operator.

Examples:

Assume a QmfData type defines fields named "name", "address" and "town". The following predicate expression matches any instance with a name field set to "tross", or any instance where the name field is "jross", the address field is "1313 Spudboy Lane" and the town field is "Utopia".

```
[ "or" [ "eq" "name" [ "quote" "tross" ] ]
      [ "and" [ "eq" "name" [ "quote" "jross" ] ]
          [ "eq" "address" [ "quote" "1313 Spudboy Lane" ] ]
          [ "eq" [ "quote" "Utopia" ] "town" ]
      ]
]
```

Assume a QmfData type with fields "name" and "age". A predicate to find all instances with name matching the regular expression "?ross" with an optional age field that is greater than the value 29 or less than 12 would be:

```
[ "and" [ "re_match" "name" [ "quote" "?ross" ] ]
      [ "and" [ "exists" "age" ]
          [ "or" [ "gt" "age" 27] [ "lt" "age" 12] ]
      ]
]
```

The valid set of `<name>` values in an predicate expression is determined by the `target` of the Query. QMF reserves a set of `<name>` values it recognizes. The tables below list the set of name strings allowed for each type of `target`, and what these names evaluate to from the `target` instance.

Target "schema_package" valid names	Description
<code>"_package_name"</code>	Evaluates to the schema's package name string.

Target "schema_id" and "schema" valid names	Description
<code>"_package_name"</code>	Evaluates to the schema's package name string.
<code>"_class_name"</code>	Evaluates to the schema's class name string.
<code>"_type"</code>	Evaluates to the schema's type string (" <code>_data</code> " or " <code>_event</code> ").
<code>"_hash_str"</code>	Evaluates to the schema's hash string value.
<code>"_schema_id"</code>	Evaluates to the schema's full identifier (<code>SchemaClassId</code>).
<code><property-name></code>	Name of a property defined by the schema. Evaluates to the name string, or None if the property is not defined.
<code><method-name></code>	Name of a method defined by the schema. Evaluates to the name string, or None if the method is not defined.

Target "agent_info" valid names	Description
<code>"_name"</code>	Evaluates to the agent's name string.

Target "object_id" and "object" valid names	Description
<code>"_package_name"</code>	If schema assigned, evaluates to the schema's package name string, else None.
<code>"_class_name"</code>	If schema assigned, evaluates to the schema's class name string, else None.
<code>"_hash_str"</code>	If schema assigned, evaluates to the schema's hash string value, else None.
<code>"_schema_id"</code>	If schema assigned, evaluates to the schema's full identifier (<code>SchemaClassId</code>), else None.
<code>"_object_id"</code>	Evaluates to the identifying name string.
<code>"_update_ts"</code>	Evaluates to the last update timestamp.
<code>"_create_ts"</code>	Evaluates to the creation timestamp.
<code>"_delete_ts"</code>	Evaluates to the deletion timestamp.
<code><value-name></code>	Specifies the name of a data item in the QmfData data object. Evaluates to the value of the data item.

QMF reserved `<name>` values all start with the underscore character `"_"`. Do not create value or method names starting with an underscore.

The `QmfQuery` class represents a query:

```

class QmfQuery:
    <constructor>(target=<target name>,
                  _target_params=None,
                  _predicate=(optional)<predicate map>,
                  _id=(optional)<target-identifier>
    <constructor>(map=Map representation of a QmfQuery, as generated by .map_encode())
    .get_target(): return target name
    .get_target_param(): return params.
    .get_selector(): returns QmfQuery.ID or QmfQuery.PREDICATE
    .get_id(): return identifier if selector type is QmfQuery.ID, else None
    .get_predicate(): return predicate expression if selector type is QmfQuery.PREDICATE
    .evaluate(QmfData): evaluate query against a QmfData instance. Returns True if query
matches the instance, else false.
    .map_encode(): returns a map encoding of the QmfQuery instance

```

The map encoding of a QmfQuery:

Index	Optional	Type	Description
"what"	N	map	The target map.
"id"	Y	map	The identifier, map format determined by target type.
"where"	Y	map	The predicate map.

Example Queries

With the above syntax, QMF queries can be constructed using AMQP maps and lists. For example, a query for all known schema identifiers:

```
{"query": {"what": {"schema_id":None}}}
```

Note that the absence of a "where" clause acts as a "match all" directive. This query will return a list of all known SchemaId's.

A query for all schema identifiers whose package names are equal to the string "myPackage":

```
{"query": {"where": [{"eq": "_package_name": ["quote" "myPackage"]}], "what": {"schema_id": None}}
    }
```

Query for the particular QmfData data object with the identifier "Agent007":

```
{"query": {"what": {"object":None}, "id": "Agent007"}}
```

Query for the particular SchemaObjectClass instance that has the package name "MyPackage", class name "MyClass", that has a hash string of 'b49cda2d-bbe53b97-9f6ee5d1-485ea3da':

```
{"query": {"what": {"schema":None}, "id": {"_package_name": "MyPackage", "_class_name": "MyClass", "_type": "_data", "_hash_str": "b49cda2d-bbe53b97-9f6ee5d1-485ea3da"}}
```

Query for all SchemaClass objects that match a given package and class name:

```
{"query": {"what": {"schema": None}, "where": [{"and": [{"eq": "_package_name": ["quote" "myPackage"]}, {"eq": "_class_name": ["quote" "someClass"]}]}]}
```

Query all managed objects belonging to the "myPackage" package, of the class "myClass", whose object_id matches a given regular expression. Return a list of matching object identifiers:

```
{
  "query": {
    "what": {
      "object_id": None
    },
    "where": [
      "and" [
        "eq" "_package_name" ["quote" "myPackage"],
        ["eq" ["quote" "myClass"] "_class_name"],
        ["re_match" "_primary_key" ["quote" "foo*"]]
      ]
    ]
  }
}
```

Query for all QmfData objects from a particular schema, whose "temperature" property is greater or equal to 50:

```
{
  "query": {
    "what": {
      "object": None
    },
    "where": [
      "and" [
        "eq" "_package_name" ["quote" "aPackage"],
        ["eq" ["quote" "someClass"] "_class_name"],
        ["ge" "temperature" 50]
      ]
    ]
  }
}
```

In the previous example, the agent will convert the value 50 to a type compatible with the type given by the "temperature" property's schema in order to do the comparison.

Query for all objects that match the given schema, which have a property named "state" which does not match the regular expression "\$Error*", or whose "owner" property is not set to "Cartman".

```
{
  "query": {
    "what": {
      "object": None
    },
    "where": [
      "and" [
        "eq" "_package_name" ["quote" "aPackage"],
        ["eq" ["quote" "someClass"] "_class_name"],
        ["or" [
          "not" [
            "re_match" "state" ["quote" "$Error"]
          ],
          ["ne" "owner" ["quote" "Cartman"]]
        ]]
      ]
    ]
  }
}
```

Subscriptions

A subscription allows a Console application to monitor specific management data for changes in state. A Console creates a subscription with an Agent based on a Query. The Query specifies the set of management data that is to be monitored. The Agent will periodically publish updates to the subscribing Console(s). The update contains a snapshot of the monitored data.

A subscription remains in effect for a predetermined amount of time. Once the subscription expires, no further updates are published. A console may elect to refresh a subscription prior to its expiration. Alternatively, a Console may explicitly cancel the subscription when the data no longer needs to be monitored.

Invoking Methods

QMF allows a Console application to perform a "remote procedure call" on the Agent. The procedure - or *method* - call executes on the Agent. On completion a result is passed back to the Console. Method calls can be associated with an instance of a data object, or applied to the Agent as a whole.

The structure of a method call may be described by the schema associated with the object. The schema can define a name for the method and a description of its input and output parameters. The SchemaMethod class is used for this purpose.

The value(s) returned to the Console when the method call completes are represented by the MethodResult class. The MethodResult class indicates whether the method call succeeded or not, and, on success, provides access to all data returned by the method call. Returned data is provided in a map indexed by the name of the parameter. The map contains only those parameters that are classified as "output" by the SchemaMethod.

Should a method call result in a failure, this failure is indicated by the presence of an error object in the MethodResult. This object is represented by a QmfData object. The structure of this QmfData object is application-defined, but should contain a description of the reason for the failure. There are no returned parameters when a method call fails.

A successful method invocation is indicated by the absence of the QmfData error object in the MethodResult.

```
class MethodResult:
  <constructor>( QmfData <exception> | <map of properties> )
  .succeeded(): returns True if the method call executed without error.
  .get_exception(): returns the QmfData error object if method fails, else None
  .get_arguments(): returns a map of "name"=<value> pairs of all returned arguments.
  .get_argument(<name>): returns value of argument named "name".
```

Management Events

An event is a notification that is sent by an Agent to alert Console(s) of a change in some aspect of the system under management. Agents publish events asynchronously. Consoles have the option of receiving events from a given Agent.

To publish an event, the Agent application must call the `raise_event()` method, passing an instance of a QmfEvent object. The Agent publishes the QmfEvent instance.

To receive events, the Console application must enable event reception on a per-agent basis. The Console application does this by calling the `enable_events()` method on the desired Agent instance. Published events from the Agent will then appear on the Console's work-queue. The Console application may disable events by invoking the Agent's `disable_events()` method.

Work-Queue Event Model.

The original QMF API defined a set of callback methods that a Console or Agent application needed to provide in order to process asynchronous QMF events. Each asynchronous event defined its own callback method.

The new API replaces this callback model with a work-queue approach. All asynchronous events are represented by a WorkItem object. When a QMF event occurs it is translated into a WorkItem object and placed in a FIFO queue. It is left to the application to drain this queue as needed.

This new API does require the application to provide a single callback. The callback is used to notify the application that WorkItem object(s) are pending on the work queue. This callback is invoked by QMF when one or more new WorkItem objects are added to the queue. To avoid any potential threading issues, the application is *not* allowed to call any QMF API from within the context of the callback. The purpose of the callback is to notify the application to schedule itself to drain the work queue at the next available opportunity.

For example, a console application may be designed using a `select()` loop. The application waits in the `select()` for any of a number of different descriptors to become ready. In this case, the callback could be written to simply make one of the descriptors ready, and then return. This would cause the application to exit the wait state, and start processing pending events.

The callback is represented by the Notifier virtual base class. This base class contains a single method. An application derives a custom notification handler from this class, and makes it available to the Console or Agent object.

```
class Notifier:  
    .indication(): Called when the internal work queue becomes non-empty due to the arrival of  
    one or more WorkItems.  
        This method will be called by the internal QMF management thread - it is illegal to invoke  
        any QMF APIs from  
            within this callback. The purpose of this callback is to indicate that the application  
        should schedule itself  
            to process the work items.
```

The WorkItem class represents a single notification event that is read from the work queue:

```
class WorkItem:  
    .get_type(): Identifies the type of work item.  
    .get_handle(): returns the reply handle for an asynchronous operation, if present.  
    .get_params(): Returns the payload of the work item. The type of this object is determined by  
    the type of the workitem.
```

Console and Agent-specific WorkItem types are defined.

Console Application Model

This section describes the API that is specific to Console components.

A QMF console component is represented by a Console class. This class is the topmost object of the console application's object model.

A Console is composed of the following objects:

- a connection to the AMQP bus
- a queue of inbound work items
- a collection of all known schemas
- a list of all known remote Agents
- a cache of known data object proxies

The connection to the AMQP bus is used to communicate with remote Agents. The queue is used as a source for notifications coming from remote Agents.

QmfConsoleData Class

The QmfData class is subclassed to provide a Console specific representation of management data.

The Console application represents a managed data object by the QmfConsoleData class. The Console has "read only" access to the data values in the data object via this class. The Console can also invoke the methods defined by the object via this class. The actual data stored in this object is cached from the Agent. In order to update the cached values, the Console invokes the instance's `refresh()` method.

Note that the `refresh()` and `invoke_method()` methods require communication with the remote Agent. As such, they may block. For these two methods, the Console has the option of blocking in the call until the call completes. Optionally, the Console can receive a notification

asynchronously when the operation is complete. See below for more detail regarding synchronous and asynchronous API calls.

```
class QmfConsoleData(QmfData):
    .get.timestamps(): returns a list of timestamps describing the
                      lifecycle of the object. All timestamps are
                      represented by the AMQP timestamp type.
                      [0] = time of last update from Agent,
                      [1] = creation timestamp
                      [2] = deletion timestamp, or zero if not deleted.

    .get_create_time(): returns the creation timestamp
    .get_update_time(): returns the update timestamp
    .get_delete_time(): returns the deletion timestamp, or zero if not yet deleted.
    .is_deleted(): True if deletion timestamp not zero.

    .refresh([reply-handle | timeout]): request that the Agent update the value of this object's
contents.
    .invoke_method(name, inArgs{}, [[reply-handle] | [timeout]]): invoke the named method on
this instance.
```

Asynchronous Event Model.

The Console application must support the following WorkItem types:

- AGENT_ADDED
- AGENT_DELETED
- NEW_PACKAGE
- NEW_CLASS
- OBJECT_UPDATE
- EVENT_RECEIVED
- AGENT_HEARTBEAT
- SUBSCRIBE_RESPONSE
- RESUBSCRIBE_RESPONSE
- SUBSCRIPTION_INDICATION

These WorkItem types are described in detail below:

AGENT_ADDED

When the QMF Console receives the first heartbeat from an Agent, an AGENT_ADDED WorkItem is pushed onto the work-queue. The WorkItem's get_param() call returns a map which contains a reference to the new Console Agent instance. The reference is indexed from the map using the key string "agent". There is no handle associated with this WorkItem.

Note: If a new Agent is discovered as a result of the Console *find_agent()* method, then no AGENT_ADDED WorkItem is generated for that Agent.

AGENT_DELETED

When a known Agent stops sending heartbeat messages, the Console will time out that Agent. On Agent timeout, an AGENT_DELETED WorkItem is pushed onto the work-queue. The WorkItem's get_param() call returns a map which contains a reference to the Agent instance that has been deleted. The reference is indexed from the map using the key string "agent". There is no handle associated with this WorkItem.

The Console application must release all saved references to the Agent before returning the WorkItem.

NEW_PACKAGE

TBD.

NEW_CLASS

TBD.

OBJECT_UPDATE

TBD.

EVENT_RECEIVED

TBD

SUBSCRIBE_RESPONSE

The **SUBSCRIBE_RESPONSE** WorkItem returns the result of a subscription request made by this Console. This WorkItem is generated when the Console's *create_subscription()* is called in an asynchronous manner, rather than pending for the result.

The *get_params()* method of a **SUBSCRIBE_RESPONSE** WorkItem will return an instance of the following object:

```

class SubscribeParams:
    .get_subscription_id(): If the subscription is successful, this method returns a
SubscriptionId object.
        Should the subscription fail, this method returns None, and get_error() can be used to
obtain an
            application-specific QmfData error object.
    .get_publish_interval(): returns the time interval in seconds on which the Agent will
publish updates
        for this subscription.
    .get_lifetime(): returns the time interval in seconds for the subscription. The subscription
will automatically
        expire after this interval if not renewed by the console.
    .get_error(): (optional) returns an application-specific QmfData object indicating why the
subscription
        request failed. Returns None if not supported.
    .get_console_handle(): returns the console handle as passed to the create_subscription()
call.

```

The SubscriptionId object must be used when the subscription is refreshed or cancelled - it must be passed to the Console's refresh_subscription() and cancel_subscription() methods. The value of the SubscriptionId does not change over the lifetime of the subscription.

The console handle will be provided by the Agent on each data indication event that corresponds to this subscription. It should not change for the lifetime of the subscription.

The get_handle() method returns the reply handle provided to the create_subscription() method call. This handle is merely the handle used for the asynchronous response, it is not associated with the subscription in any other way.

Once a subscription is created, the Agent that maintains the subscription will periodically issue updates for the subscribed data. This update will contain the current values of the subscribed data, and will appear as the first *SUBSCRIPTION_INDICATION* WorkItem for this subscription.

SUBSCRIPTION_INDICATION

The *SUBSCRIPTION_INDICATION* WorkItem signals the arrival of an update to subscribed data from the Agent.

The get_params() method of a *SUBSCRIPTION_INDICATION* WorkItem will return an instance of the following object:

```

class SubscribeIndication:
    .get_console_handle(): returns the console handle as passed to the create_subscription()
call.
    .get_data(): returns a list containing all updated data objects associated with the
subscription.

```

The get_handle() method returns None.

RESUBSCRIBE_RESPONSE

The *RESUBSCRIBE_RESPONSE* WorkItem is generated in response to a subscription refresh request made by this Console. This WorkItem is generated when the Console's refresh_subscription() is called in an asynchronous manner, rather than pending for the result.

The get_params() method of a *RESUBSCRIBE_RESPONSE* WorkItem will return an instance of the following object:

```

class SubscribeParams:
    .get_subscription_id(): If the re-subscription is successful, this method returns an
instance of
        the original SubscriptionId object. Should the subscription fail, this method returns
None,
        and get_error() can be used to obtain an application-specific QmfData error object.
    .get_publish_interval(): returns the time interval in seconds on which the Agent will
publish updates
        for this subscription.
    .get_lifetime(): returns the time interval in seconds for the subscription. The subscription
will automatically
        expire after this interval if not renewed by the console.
    .get_error(): (optional) returns an application-specific QmfData object indicating why the
re-subscription
        request failed. Returns None on successful resubscribe.
    .get_console_handle(): returns the console handle as passed to the create_subscription()
call, if available.
        Note: if the Agent failed the resubscribe request due to an unrecognized subscription,
this call may
            return None.

```

The `get_handle()` method returns the reply handle provided to the `refresh_subscription()` method call. This handle is merely the handle used for the asynchronous response, it is not associated with the subscription in any other way.

Local representation of a remote Agent.

The console application maintains a list of all known remote Agents. Each Agent is represented by the Agent class:

```
class Agent:  
    .get_name(): returns the identifying name string of the agent. This name is used to send  
    AMQP messages directly to this agent.  
    .is_active(): returns True if the agent is alive (heartbeats have not timed out)  
    .invoke_method(name, inArgs{}, [[reply-handle] | [timeout]]): invoke the named method  
    against the agent.  
    .enable_events(): allows reception of events from this agent.  
    .disable_events(): prevents reception of events from this agent.  
    .destroy(): releases this Agent instance. Once called, the console application should not  
    reference this instance again.  
    ?tbd?
```

The Console Object.

The Console class is the top-level object used by a console application. All QMF console functionality is made available by this object. A console application must instantiate one of these objects.

As noted below, some Console methods require contacting a remote Agent. For these methods, the caller has the option to either block for a (non-infinite) timeout waiting for a reply, or to allow the method to complete asynchronously. When the asynchronous approach is used, the caller must provide a unique handle that identifies the request. When the method eventually completes, a WorkItem will be placed on the work queue. The WorkItem will contain the handle that was provided to the corresponding method call.

All blocking calls are considered thread safe - it is possible to have a multi-threaded implementation have multiple blocking calls in flight simultaneously.

If a name is supplied, it must be unique across all Consoles attached to the AMQP bus under the given domain. If no name is supplied, a unique name will be synthesized in the format: "qmfc-<hostname>.<pid>"

```
class Console:  
    <constructor>(name=<name-str>,  
                  domain=(optional) domain string for console's AMQP address,  
                  notifier=<class Notifier>,  
                  reply_timeout=<default for all blocking calls>,  
                  agent_timeout=<default timeout for agent heartbeat>,  
                  subscription_duration=<default lifetime of a subscription>)  
  
.destroy(timeout=None): Must be called to release Console's resources.  
  
.add_connection(QPID Connection): Connect the console to the AMQP cloud.  
  
.remove_connection(conn): Remove the AMQP connection from the console. Un-does the  
add_connection() operation, and  
    releases any agents associated with the connection. All blocking methods are unblocked  
and given a failure status.  
    All outstanding asynchronous operations are cancelled without producing WorkItems.  
  
.get_address():  
    Get the AMQP address this Console is listening to (type str).  
  
.find_agent( name string, [timeout] ): Query for the presence of a specific agent in the QMF  
domain. Returns a  
    class Agent if the agent is present. If the agent is not already known to the console,  
this call will send  
    a query for the agent and block (with default timeout override) waiting for a response.  
  
.enable_agent_discovery( [Query] ): Called to enable the asynchronous Agent Discovery  
process. Once enabled, AGENT_ADDED  
    and AGENT_DELETED work items can arrive on the WorkQueue. If a query is supplied, it  
will be used to filter agent  
    notifications.  
  
.disable_agent_discovery(): Called to disable the async Agent Discovery process enabled by  
calling enable_agent_discovery().  
  
.get_workitem_count(): Returns the count of pending WorkItems that can be retrieved.
```

```

    .get_next_workitem([timeout=0]): Obtains the next pending work item, or None if none
available.

    .release_workitem(wi): Releases a WorkItem instance obtained by getNextWorkItem(). Called
when the application has finished
        processing the WorkItem.

    .get_agents(): Returns a list of available agents (class Agent)

    .get_agent( name string ): Return the class Agent for the named agent, if known.

    .get_packages( [class Agent] ): Returns a list of the names of all known packages. If an
optional Agent is provided, then
        only those packages available from that Agent are returned.

    .get_classes( [class Agent] ): Returns a list of SchemaClassIds for all available Schema.
If an optional Agent is provided,
        then the returned SchemaClassIds are limited to those Schema known to the given Agent.

    .get_schema( class SchemaClassId [, class Agent]): Return a list of all available class
SchemaClass across all known agents.
        If an optional Agent is provided, restrict the returned schema to those supported by
that Agent.

    .get_objects( _SchemaClassId= | _package=, _class= |
                  _object_identifier=,
                  [timeout=],
                  [list-of-class-Agent] ): perform a blocking query for QmfConsoleObjects.
Returns a list (possibly empty) of matching
        objects. The selector for the query may be either:
            * class SchemaClassId - all objects whose schema match the schema identified by
_SchemaClassId parameter.
            * package/class name - all objects whose schema are contained by the named package and
class.
            * the object identified by _object_identifier
        This method will block until all known agents reply, or the timeout expires. Once the
timeout expires, all
        data retrieved to date is returned. If a list of agents is supplied, then the query is
sent to only those agents.

    .create_subscription( agent, class Query, console_handle [, reply_handle] [, timeout],
                           [, publish_interval] [, lifetime] ): creates a subscription to the
agent
        using the given Query. The console_handle is an application-provided handle that will
accompany each subscription update
        send from the Agent. Subscription updates will appear as SUBSCRIPTION_INDICATION
WorkItems on the Console's work queue.
        The publish_interval is the requested time interval in seconds on which the Agent should
publish updates. The lifetime
        parameter is the requested time interval in seconds for which this subscription should
remain in effect. Both the
        requested lifetime and publish_interval may be overridden by the Agent, as indicated in
the subscription response.
        This method may be called asynchronously by providing a reply_handle argument. When
called asynchronously, the result
        of this method call is returned in a SUBSCRIBE_RESPONSE WorkItem with a handle matching
the value of reply_handle.
        Timeout can be used to override the console's default reply timeout. When called
synchronously, this method returns a class
        SubscribeParams object containing the result of the subscription request.

    .refresh_subscription( SubscriptionId [, lifetime] [,reply_handle] [, timeout] ): renews a
subscription identified by
        SubscriptionId. The Console may request a new subscription duration by providing a
requested lifetime. This
        method may be called asynchronously by providing a reply_handle argument. When called
asynchronously, the result
        of this method call is returned in a SUBSCRIBE_RESPONSE WorkItem. Timeout can be used
to override the console's
        default reply timeout. When called synchronously, this method returns a class
SubscribeParams object containing
        the result of the subscription request.

```

```
.cancel_subscription( SubscriptionId ): terminates the given subscription.
```

Example Console Application

The following pseudo-code performs a blocking query for a particular agent.

```
logging.info( "Starting Connection" )
conn = Connection("localhost")
conn.connect()

logging.info( "Starting Console" )
myConsole = Console()
myConsole.add_connection( conn )

logging.info( "Finding Agent" )
myAgent = myConsole.find_agent( "com.aCompany.Examples.anAgent", _timeout=5 )

if myAgent:
    logging.info( "Agent Found: %s" % myAgent )
else:
    logging.info( "No Agent Found!" )

logging.info( "Removing connection" )
myConsole.remove_connection( conn )

logging.info( "Destroying console:" )
myConsole.destroy( _timeout=10 )
```

The following pseudo-code performs a non-blocking query for all agents. It completes when at least one agent is found.

```
class MyNotifier(Notifier):
    def __init__(self, context):
        self._myContext = context
        self.WorkAvailable = False

    def indication(self):
        print("Indication received! context=%d" % self._myContext)
        self.WorkAvailable = True

noteMe = MyNotifier( 668 )

logging.info( "Starting Connection" )
conn = Connection("localhost")
conn.connect()

myConsole = Console(notifier=noteMe)
myConsole.add_connection( conn )

myConsole.enable_agent_discovery()
logging.info( "Waiting..." )

while not noteMe.WorkAvailable:
    print("No work yet...sleeping!")
    time.sleep(1)

print("Work available = %d items!" % myConsole.getWorkItemCount())
wi = myConsole.get_next_workitem(timeout=0)
while wi:
    print("work item %d:%s" % (wi.getType(), str(wi.getParams())))
    wi = myConsole.get_next_workitem(timeout=0)

logging.info( "Removing connection" )
myConsole.remove_connection( conn )

logging.info( "Destroying console:" )
myConsole.destroy( 10 )
```

Agent Application Model

This section describes the API that is specific to Agent components.

A QMF agent component is represented by a instance of the Agent class. This class is the topmost object of the agent application's object model. Associated with a particular agent are:

- the set of objects managed by that agent
- the set of schema that describes the structured objects owned by the agent
- a collection of consoles that are interfacing with the agent

The Agent class communicates with the application using the same work-queue model as the console. The agent maintains a work-queue of pending requests. Each pending request is associated with a handle. When the application is done servicing the work request, it passes the response to the agent along with the handle associated with the originating request.

QmfAgentData Class

The Agent manages the data it represents by the QmfAgentData class - a derivative of the QmfData class. The Agent is responsible for managing the values of the properties within the object, as well as servicing the object's method calls. Unlike the Console, the Agent has full control of the state of the object.

```
class QmfAgentData(QmfData):
    .destroy(): mark the object as deleted by setting the deletion timestamp to the current
    time.
    .set_value(name, value): update the value of the property.
    .inc_value(name, delta): add the delta to the property
    .dec_value(name, delta): subtract the delta from the property
    ?tbd?
```

An agent can support one of two different models for managing its database of QmfAgentData objects: internal or external store.

Internal Object Store

An agent that implements internal object store gives full responsibility for managing its data objects to the QMF infrastructure. In this model, the application passes a reference for each managed object to the QMF agent. The agent manages the set of objects internally, directly accessing the contents of the object in order to service console requests.

With this model, the application's complexity is reduced. The application needs to instantiate the object and register it with the agent. The application also maintains a reference to the object, as the application is responsible for updating the object's properties as necessary.

However, the application must still service method calls. The agent notifies the application when a method call has been requested by a console. The application services the method call, passing the result of the method back to the agent. The agent then relays the response to the originating console.

The application may decide to delete an object instance. The application does this by invoking the `destroy()` method on the object. This notifies the agent, which will mark the object as deleted in its database. Once the application invokes the `destroy()` method on an object, it must no longer access the object. The agent will clean up the object at a later point in time.

Internal object store is the default model for agent object management.

Data Consistency

The internal object store requires sharing of the managed data between the agent and the application. The application is responsible for keeping the data up to date, while the agent is responsible for providing the data to client consoles. It is likely that these components may be implemented in separate execution contexts. This raises the possibility that a data item could be in the process of being written to by the application at the same moment the agent attempts to read it. This could result in invalid data being read.

To prevent this from occurring, the QmfAgentObject class provides accessors for all data in the object. These accessors provide atomic access to the underlying data. Therefore, both the agent and the application code **must** use these accessors to manipulate a shared object's data.

External Object Store

An alternate agent implementation allows the application to take full responsibility for managing the objects. With this model, all instances of managed objects exist external to the agent. When a console requests an action against an object, that action is transferred from the agent to the application. The application then must process the request, and send the result to the agent. The agent then sends a reply to the requesting console.

The model gives full control of the managed objects to the application, but usually requires more application development work.

Agent Class

The base class for the agent object is the Agent class. This base class represents a single agent implementing internal store.

```

class Agent:
    <constructor>( name=<name-string>,
                    domain=(optional) domain string for agent's AMQP address,
                    notifier=class Notifier,
                    heartbeat_interval=30,
                    max_msg_size=0)
    .get_name(): return the name string of the agent.
    .set_connection( QPID Connection ): connect the agent to the AMQP cloud.
    .register_object_class( class SchemaObjectClass ): Register a schema for an object class
with the agent. The agent must
        have a registered schema for an object class before it can handle objects of that
class.
    .register_event_class( class SchemaEventClass ): Register a schema for an event class with
the agent. The agent must
        have a registered schema for an event class before it can handle events of that class.
    .raise_event( class QmfEvent ): Cause the agent to raise the given event.
    .add_object( class QmfAgentData ): passes a reference to an instance of a managed QMF object
to the agent. The object's
        name must uniquely identify this object among all objects known to this agent.
    .get_workitem_count(): Returns the count of pending WorkItems that can be retrieved.
    .get_next_workitem([timeout=0]): Obtains the next pending work item, or None if none
available.
    .release_workitem(wi): Releases a WorkItem instance obtained by get_next_workitem(). Called
when the application has finished
        processing the WorkItem.
    .method_response(name="method name",
                    handle=<handle from WorkItem>,
                    out_args={output argument map}
                    error=<QmfData> ): Indicate to the agent that the application has completed
processing a method
        request. See the description of the METHOD_CALL WorkItem.

```

AgentExternal Class

The AgentExternal class must be used by those applications that implement the external store model. The AgentExternal class extends the Agent class by adding interfaces that notify the application when it needs to service a request for management operations from the agent.

```

class AgentExternal(Agent):
    <constructor>(name=<name-string>,
                  domain=(optional) domain string for agent's AMQP address,
                  notifier= class Notifier,
                  heartbeat_interval=30,
                  max_msg_size=65535)
    .alloc_object_id( name="object name"): indicate to QMF that the named object is available to
    be managed. Once this method returns,
        the agent will service requests from consoles referencing this data.
    .free_object_id( name="object name" ): indicate to QMF that the named object is no longer
    available to be managed.
    .query_response( handle=<handle from WorkItem>,
                     class QmfAgentObject): send a managed object in reply to a received query.
Note that ownership of the object
    instance is returned to the caller on return from this call.
    .query_complete( handle=<handle from WorkItem>,
                     result=<status code> ): Indicate to the agent that the application has
    completed processing a query request.
        Zero or more calls to the queryResponse() method should be invoked before calling
    query_complete(). If the query should
        fail - for example, due to authentication error - the result should be set to a non-zero
    error code ?TBD?.

    .subscription_response( handle=<handle from WorkItem>,
                           console_handle=<handle provided by Console for this subscription>,
                           subscription_handle=<agent-provided context>,
                           lifetime=<seconds>, publish_interval=<seconds>,
                           error=<QmfData>): Indicate the result of a SUBSCRIBE_REQUEST
WorkItem.
        If the subscription request is successful, the Agent application must provide a unique
    subscription_handle. If replying
            to a sucessful subscription refresh, the original subscription_handle must be supplied.
    The lifetime parameter should be
        set to the duration of the subscription in seconds. The publish_interval should be set
    to the time interval in seconds
        between successive publications on this subscription. If the subscription or refresh
    fails, the subscription_handle
        should be set to None and error may be set to an application-specific QmfData instance
    that describes the error. Should
        a refresh request fail, the console_handle may be set to None if unknown.

    .subscription_indicate(console_handle, [list of subscribed data]): Send a list of updated
subscribed data to the Console.

    .subscription_cancel(handle=<handle from WorkItem>, console_handle): Acknowledge a
Subscription Cancel WorkItem.

```

Asynchronous Event Model.

The Agent uses the same notification driven work-queue model as the Console. In the Agent case, the following set of WorkItem types are supported:

- METHOD_CALL
- QUERY
- SUBSCRIBE_REQUEST
- RESUBSCRIBE_REQUEST
- UNSUBSCRIBE_REQUEST

Note Well: In the case of an internal store agent implementation, only the METHOD_CALL work item is generated. An external store agent must support all work item types.

METHOD_CALL

The *METHOD_CALL* WorkItem describes a method call that must be serviced by the application on behalf of this agent.

The *get_params()* method of a *METHOD_CALL* WorkItem will return an instance of the following object:

```

class MethodCallParams:
    .get_name(): returns a string containing the name of the method call.
    .get_object_id(): returns the identifier for the object on which this
        method needs to be invoked. Returns None iff there is no associated
        object (a method call against the agent itself).
    .get_args(): returns a map of input arguments for the method. Arguments
        are in "name"=<value> pairs. Returns None if no arguments are supplied.
    .get_user_id(): returns authenticated user id of caller if present, else None.

```

On completion of the method call, the application must provide the result of the call to the Agent. This is done by invoking the Agent's `method_response()` method. The `method_response()` method must be passed the handle from the `METHOD_CALL` WorkItem.

On successful completion of a method call, any output arguments from the method call must be passed in the `out_args` map parameter, in `name=<value>` pairs. The `error` parameter must be set to `None`.

If the method call fails the application must indicate the failure by passing a `QmfData` instance via the `error` parameter. The structure of this `QmfData` is application-specific, and meant to provide a description of the failure to the console.

QUERY

```

QUERY parameters: ( class Query,
                     user_id=<authenticated id of the user> )

```

The `QUERY` WorkItem describes a query that the application must service. The application should call the `query_response()` method for each object that satisfies the query. When complete, the application must call the `query_complete()` method. If a failure occurs, the application should indicate the error to the agent by calling the `query_complete()` method with a description of the error.

SUBSCRIBE_REQUEST

The `SUBSCRIBE_REQUEST` WorkItem provides a query that the agent application must periodically publish until the subscription is cancelled or expires. On receipt of this WorkItem, the application should call the `Agent::subscription_response()` method to acknowledge the request. On each publish interval, the application should call `Agent::subscription_indicate()`, passing a list of the objects that satisfy the query. The subscription remains in effect until an `UNSUBSCRIBE_REQUEST` WorkItem for the subscription is received, or the subscription expires.

The `get_params()` method call of the `SUBSCRIBE_REQUEST` WorkItem returns an instance of the following object:

```

class SubscriptionParams:
    .get_console_handle(): returns the handle that the console uses to identify this
        subscription.
        This handle must be passed along with every published update from the Agent.
    .get_query(): returns the QmfQuery object associated with the subscription.
    .get_publish_interval(): returns the requested time interval in seconds for updates.

    Returns
        zero if the Agent's default interval should be used.
    .get_lifetime(): returns the requested lifetime for the subscription. Zero if the Agent's
        default subscription lifetime should be used.
    .get_user_id(): returns authenticated user id of Console if present, else None.

```

The Agent application must call the `AgentExternal::subscription_response()` method in response to this WorkItem.

The `get_handle()` WorkItem method returns the reply handle which should be passed to the Agent's `subscription_response()` method.

RESUBSCRIBE_REQUEST

The `RESUBSCRIBE_REQUEST` is sent by a Console to renew an existing subscription. The Console may request a new duration for the subscription, otherwise the previous lifetime interval is repeated.

The `get_params()` method call of the `RESUBSCRIBE_REQUEST` WorkItem returns an instance of the following object:

```

class ResubscribeParams:
    .get_subscription_id(): returns the subscription identifier provided by the Agent.
    .get_lifetime(): returns the requested lifetime for the subscription. Zero if the previous
        interval should be used.
    .get_user_id(): returns the authenticated user id of the Console if present, else None.

```

The Agent application must call the `AgentExternal::subscription_reponse` method in response to this WorkItem.

The `get_handle()` WorkItem method returns the reply handle which should be passed to the Agent's `subscription_reponse()` method

UNSUBSCRIBE_REQUEST

The `UNSUBSCRIBE_REQUEST` is sent by a Console to terminate an existing subscription.

The get_params() method call returns the subscription identifier assigned by the Agent when the subscription is created.

The Agent application should terminate the given subscription if it exists, and cancel sending any further updates against it.

QMFv2 Architecture

QMFv2 Architecture

Introduction

QMF (Qpid Management Framework) is a set of APIs, in multiple programming languages, that provide an abstraction for object-oriented and service-oriented remote management. It is layered on top of AMQP messaging and implemented using the Apache Qpid messaging APIs.

Architectural Components and Operations

There are two central components in the architecture of QMF: The **Agent** and the **Console**. An agent is a software component that is managed via QMF and a console is a component that manages agents. The distinction will become clearer as we discuss operations.

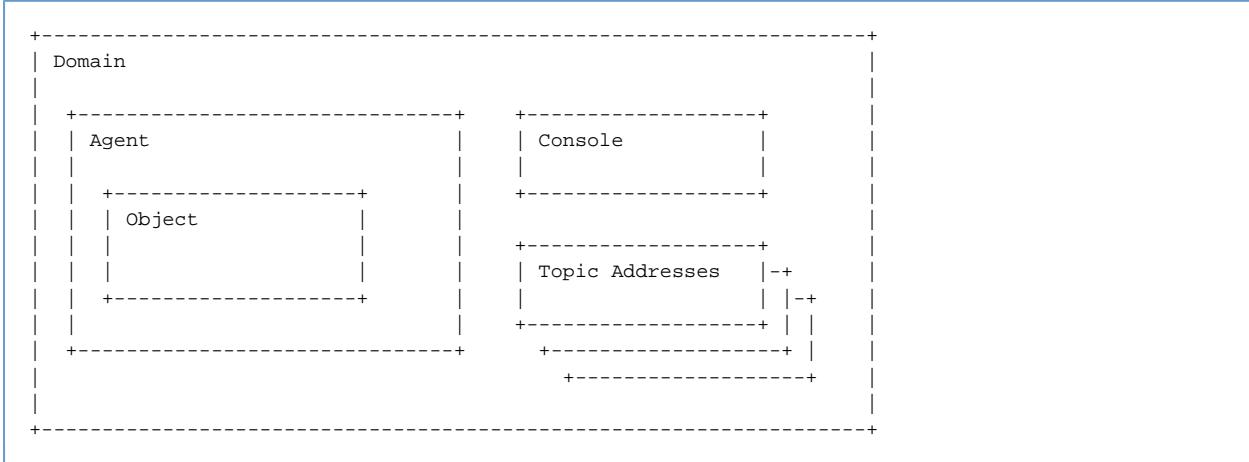
A better way to describe **console** and **agent** is to call them *roles*. For example, a software program may perform the *role* of **QMF Agent**, and therefore become manageable via QMF. Another program may act in the *role* of **QMF Console** and manage some number of agents. It is also possible for a software program to act in both roles simultaneously and can do so using a single connection into the AMQP messaging infrastructure.



In QMF, there may be any number of agents and any number of consoles. Of course, if there are no agents, it won't be a very interesting network. Agents and consoles may appear and disappear dynamically.

Addressing and Scoping

QMF, being distributed in nature, needs a way to address the various entities that it represents. Further, the addressing of entities within QMF is organized in a scope hierarchy as follows:



QMF Domain

Note that this is a new concept being introduced in QMFv2. It refers to a top-level scope for the addressing of all QMF entities. Multiple domains may be established so that independent, disjoint QMF deployments can share the same AMQP messaging network.

Console Address

When a console joins the network (i.e. an application that acts in the role of "Qmf Console"), it uses a unique and temporary identifier for addressing. This address is most likely different for each execution of the application. It is used only for reply messages sent directly to the

console from an agent in the network.

Agent Address

The agent address, like the console address, must be unique within the domain. Unlike the console address, the agent address **should** be persistent, remaining the same even after the agent application is shut down and restarted.

Object Address

Topic Addresses

Operations

QMF operations are defined in terms of the console and agent roles. This section will introduce the operations at a fairly high level. For full details on how the operations operate, refer to the API specification for the programming language you are interested in.

Agent Locate

After a console first connects itself to an AMQP messaging broker, the first thing it must do is identify one or more agents with which to interact. This is done in one of two ways, depending on the purpose of the console application.

The console may attempt to locate a single specific agent by name (i.e. The HP printer on the second floor). In this case, the console communicates directly with the agent to see if it is present in the network (using a direct exchange and the agent's unique name).

Alternatively, the console may wish to interact with a set of agents, possibly all of them. In this case, the console establishes a set of criteria in the form of a query (i.e. "all printers where vendor == 'HP'") and builds a list of available agents. In this case, the console sends a multicast query (using a topic exchange) and collects responses from matching agents. Furthermore, if a new agent comes on line that meets the criteria, the console will learn about the new agent.

Query

A **query** is initiated by a console and answered by one or more agents. A query requests data from an agent and supplies selection criteria for the data. Once each agent sends the requested data to the console, the query operation is complete.

Subscription Query

A **subscription query** is similar to a normal query except that after the requested data is sent by an agent, the query remains open and subsequent changes to the requested data are sent to the address supplied by the console.

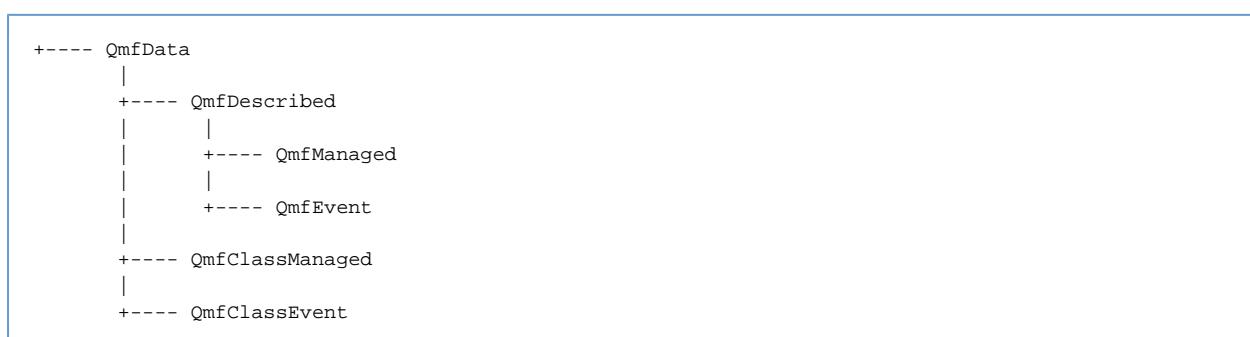
A subscription query is kept open by the agent and is closed in one of two situations. Either the console explicitly closes the subscription query or it times out after a time interval. The console may periodically refresh an existing subscription query to keep it from timing out. This allows for subscription queries to be cleaned up in case the requesting console goes away without shutting down the subscription.

Schema Query

Method Invocation

Event Distribution

The QMF Data Model



Data Type	Extends	Contains
QmfData		Values - An arbitrarily structured map of keys and values
QmfDescribed	QmfData	SchemaClassKey - Uniquely identifies the schema that describes the content of Values
QmfManaged	QmfDescribed	AgentId - Uniquely identifies the agent that manages this object. ObjectId - Uniquely identifies the object in the scope of the agent. Lifecycle - Identifies the creation, deletion, and last-changed timestamps for this object.
QmfEvent	QmfDescribed	Severity - Identifies the severity of this instance of the event. Timestamp - Identifies the time that the event was raised.

QmfClassManaged	QmfData	
QmfClassEvent	QmfData	

Other names that have been assigned to these concepts:

Data Class	Other Names Used in the Past
QmfManaged	Object, QmfObject
QmfDescribed	Unmanaged Object
QmfClassManaged	SchemaObjectClass, Object Class
QmfClassEvent	SchemaEventClass, Event Class

Broker job queue limits

Broker memory usage

It is possible for the broker to receive frames at a rate faster than it can process. When this occurs a large number of Jobs and Events are produced. This can further slow down the system by increasing memory usage, causing the GC to run frequently and generally compound the issue. This is undesirable.

High level solution

Ultimately, the broker needs to decide to cease creating new jobs until those that already exist have been processed. The broker will stop reading frames from the network layer. The servers network buffer will fill, and for OS will cease to read the socket as TCP flow control kicks in. The corresponding client side buffer to fill, and then writes to it will block.

When memory usage falls as the events are processed, the broker will start to process frames again, and normal operation will resume. However, if the broker does not recover sufficiently quickly it is possible that the socket will time out and the connection will be closed.

Required changes

1. To completely implement the above solution, a number of changes are required.
2. The broker needs to be able to determine currently used / available memory. This can be obtained via JMX.
3. The threads which process Jobs and Events need to be signalled to pause, and to resume.
4. Protection in the MINA layer of both the client and the broker needs to be enabled by default.
5. When a memory threshold is reached, the broker should fire an event which signals the job processing threads to pause. In future this event should be listened for by other mechanisms designed to mitigate the issue - such as flow to disk.

JMX Console Use Cases

1. User needs to know which queues have messages
2. User needs to know which queues are taking up memory
3. User needs to remove problematic message
4. User needs to disconnect problematic client

Current Architecture

- Current implementation
- Issues
- Current implementation
 - Broker
 - Client connection creation
 - Client processing

Current implementation

Inside Qpid, data is read from a socket and placed in a buffer. A separate thread then takes this buffer and attempts to parse it as an AMQP command. This AMQP command is then put on a second buffer. Finally a third thread reads the command and processes it.

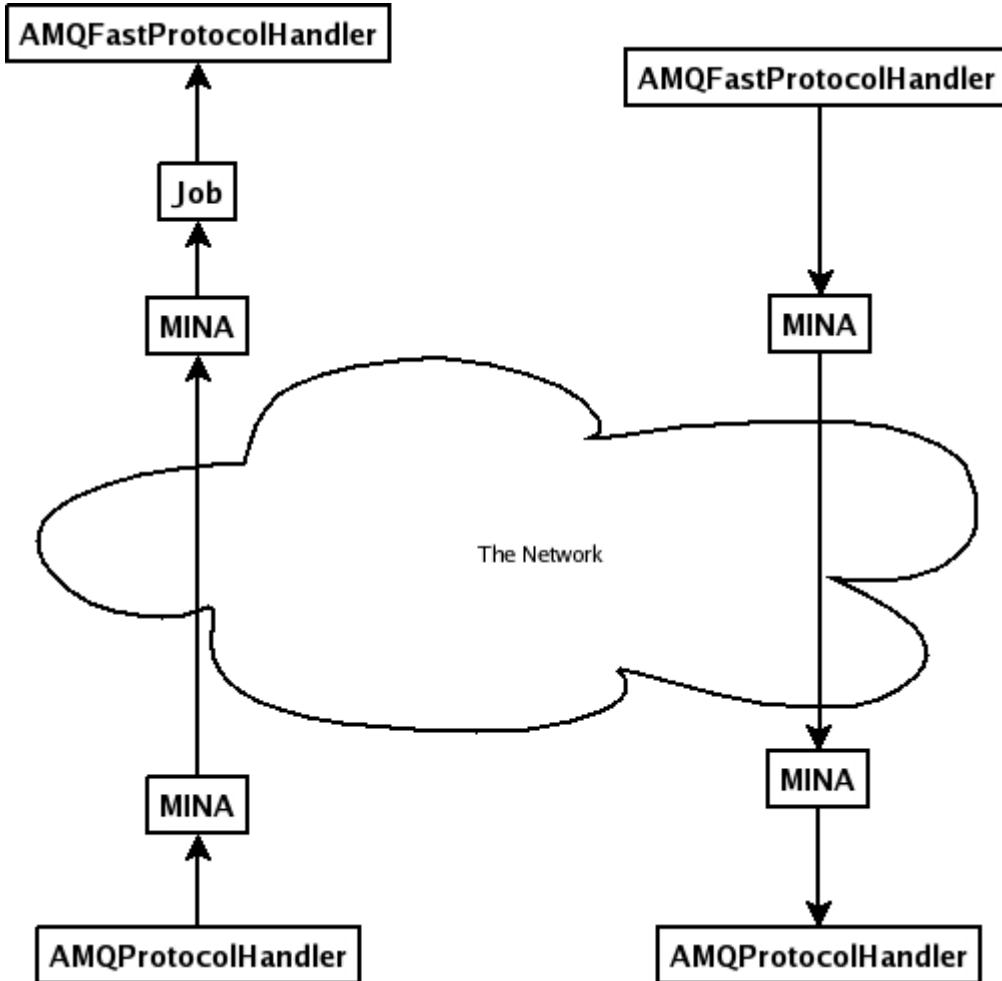
Currently the two buffers between these three threads are unbounded. This means that data is read from the network as fast as possible with no regard as to whether the broker has the capacity to process it.

Queues are themselves a kind of buffer between client applications.

From a queue the message can be assigned to be sent to a client. At this point a delivery command is placed in another buffer awaiting sending on the network. When received by the client a similar process to receiving on the broker occurs.

The whole process looks something like this

Client App sends message -> (MINA Buffer)
 -> MINA Thread takes message and sends to TCP -> (TCP Buffer)
 -> TCP places bytes on wire ->
 ~~~~~ Network ~~~~~  
 -> TCP reads from wire -> (TCP Buffer)  
 -> MINA Reads from TCP -> (MINA Buffer)  
 -> Bytes parsed and converted into AMQP Command -> (Job Queue Buffer)  
 -> AMQP Command processed, message placed on Queue -> (Queue - which is a buffer)  
 -> Message taken from queue and delivery command created -> (MINA Buffer)  
 -> MINA Thread takes message and sends to TCP -> (TCP Buffer)  
 -> TCP places bytes on wire ->  
 ~~~~~ Network ~~~~~  
 -> TCP reads from wire -> (TCP Buffer)
 -> MINA Reads from TCP -> (MINA Buffer)
 -> Bytes parsed and converted into AMQP Command -> (Job Queue Buffer)
 -> AMQP Command processed, message placed on Delivery Queue -> (Delivery Queue Buffer)
 -> Message received by client application code



Or, pictorially:

Of all the buffers above, only the TCP buffers are bounded (the Delivery Queue Buffer in the client is potentially bounded by prefetch, although prefetch is not set on bytes but on messages which may be of arbitrary size), every other buffer is a potential source of out of memory exceptions.

From the above we can see that there are many potential sources of OutOfMemoryExceptions. We need to consider where we may get unbounded growth, what scenarios will cause that, and what other ways we have to mitigate those risks.

In general we get growth of the IO (MINA) buffers when sender and receiver are operating at mismatched rates (i.e. the Client and Broker). We will get unbounded growth of the queue if the sending client is producing at a faster rate than the receiving client can process.

Issues

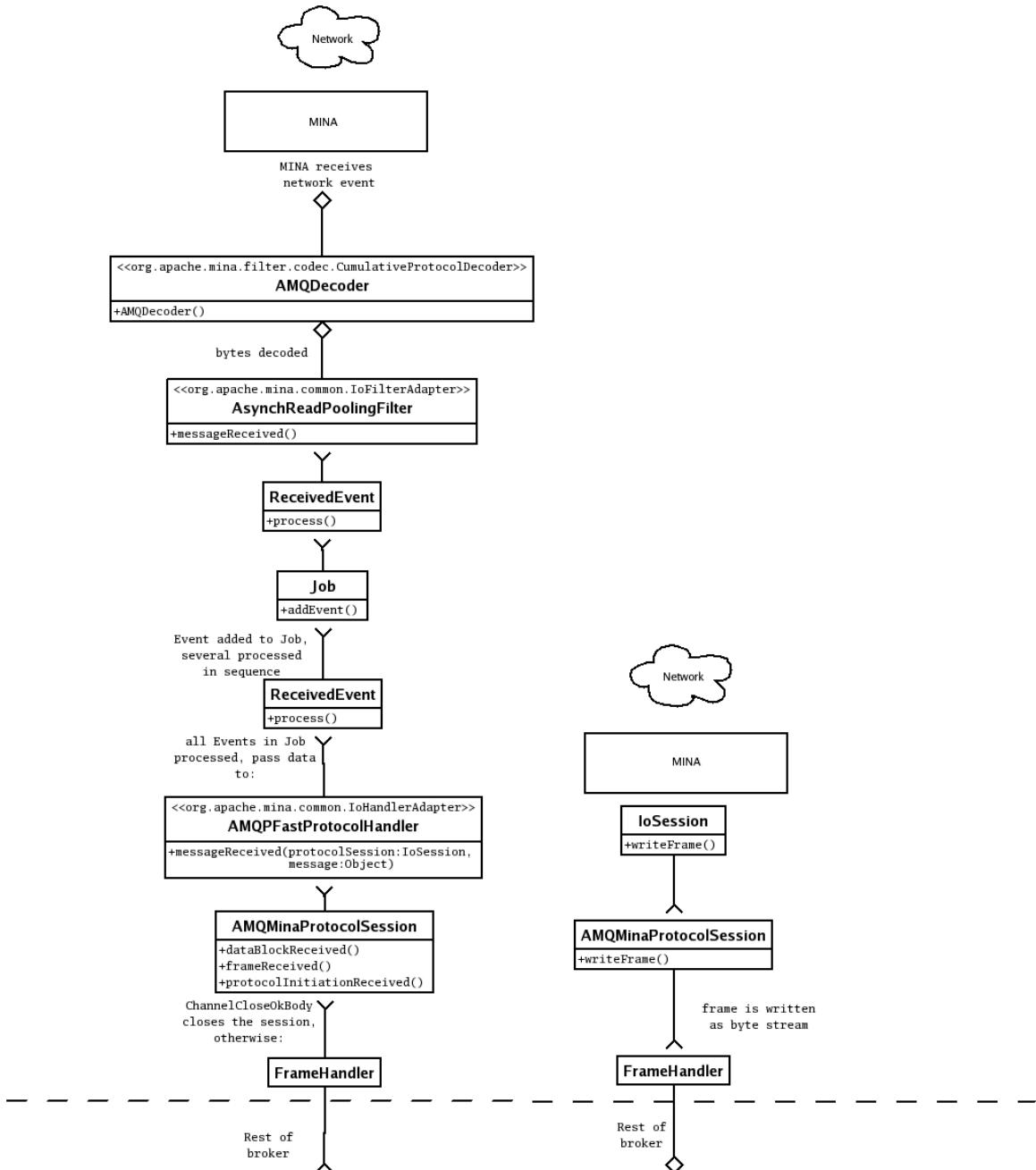
1. The current MINA networking uses unbounded buffers.
2. We replace over a dozen MINA classes, none of which have any unit test coverage. We failed to get our patches upstream and haven't attempted since then.
3. Existing unit test coverage is minimal (approx 30%)
4. Improving unit test coverage is difficult due to poor encapsulation
5. Poor encapsulation has lead to tight coupling of MINA to server

6. The current behaviour of send() leaves the potential for message loss when not using transactions and violates JMS spec. Persistent messages which are held in either the client or servers buffers before being written to disk can be lost.
7. MINA's internal state is currently a black box, leaving no way to determine how much memory is being used by an individual client connection.
8. The way that we use MINA is suboptimal for our purposes but is difficult to change due to the tight coupling
9. Supporting alternative transport layers is impossible due to tight coupling of MINA (OSI layer 4) with the AMQP handlers (OSI layer 7).

Current implementation

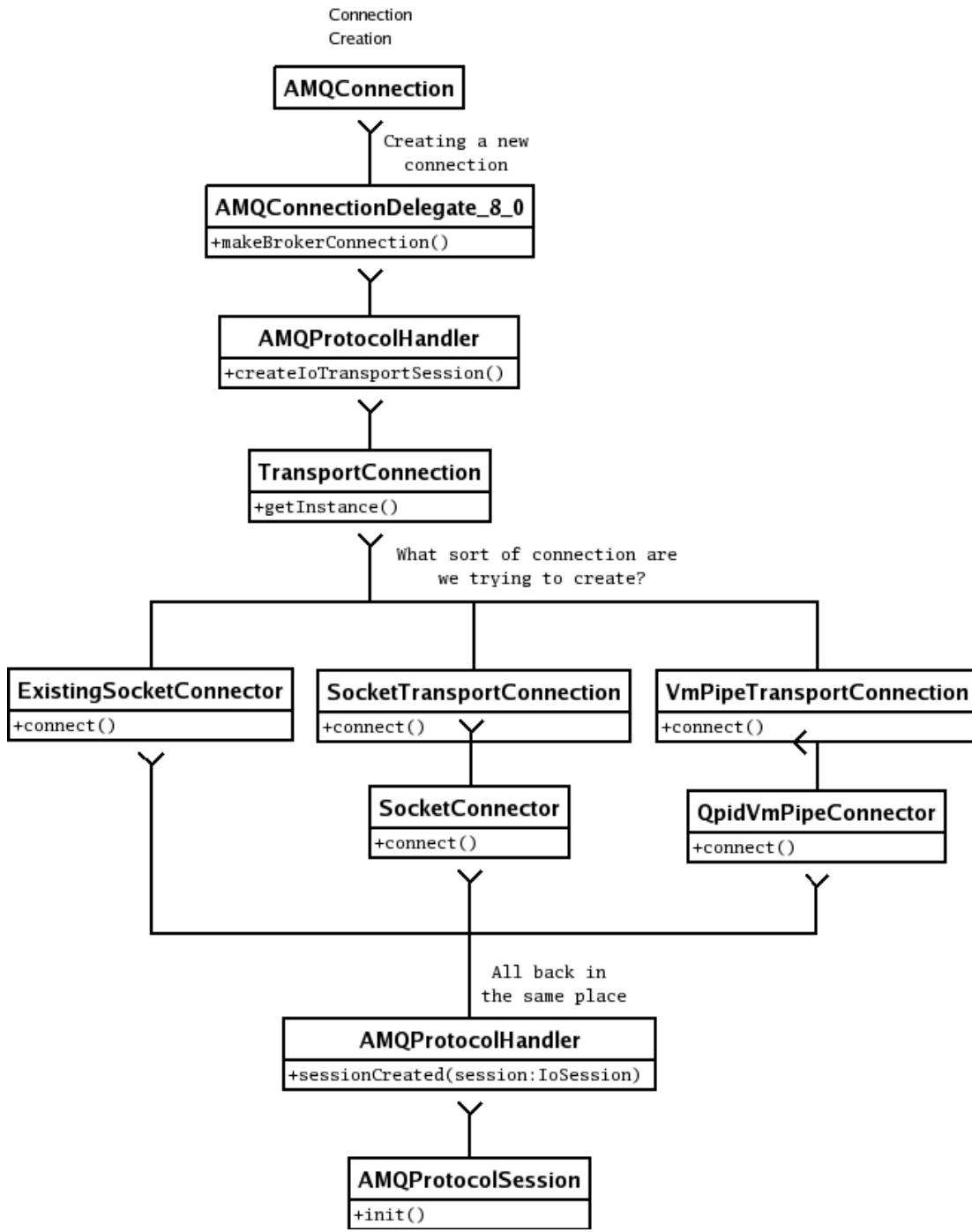
Broker

Currently the broker decodes the incoming network data, adds the frames to a Job queue which are then processed as Events by AMQPFastProtocolHandler which passes the majority of the work to AMQMinaProtocolSession. Often this results in a FrameHandler being called. On the outbound route Frames are written to AMQMinaProtocolSession which calls IoSession.writeFrame which passes the data to MINA for writing to the wire.



Client connection creation

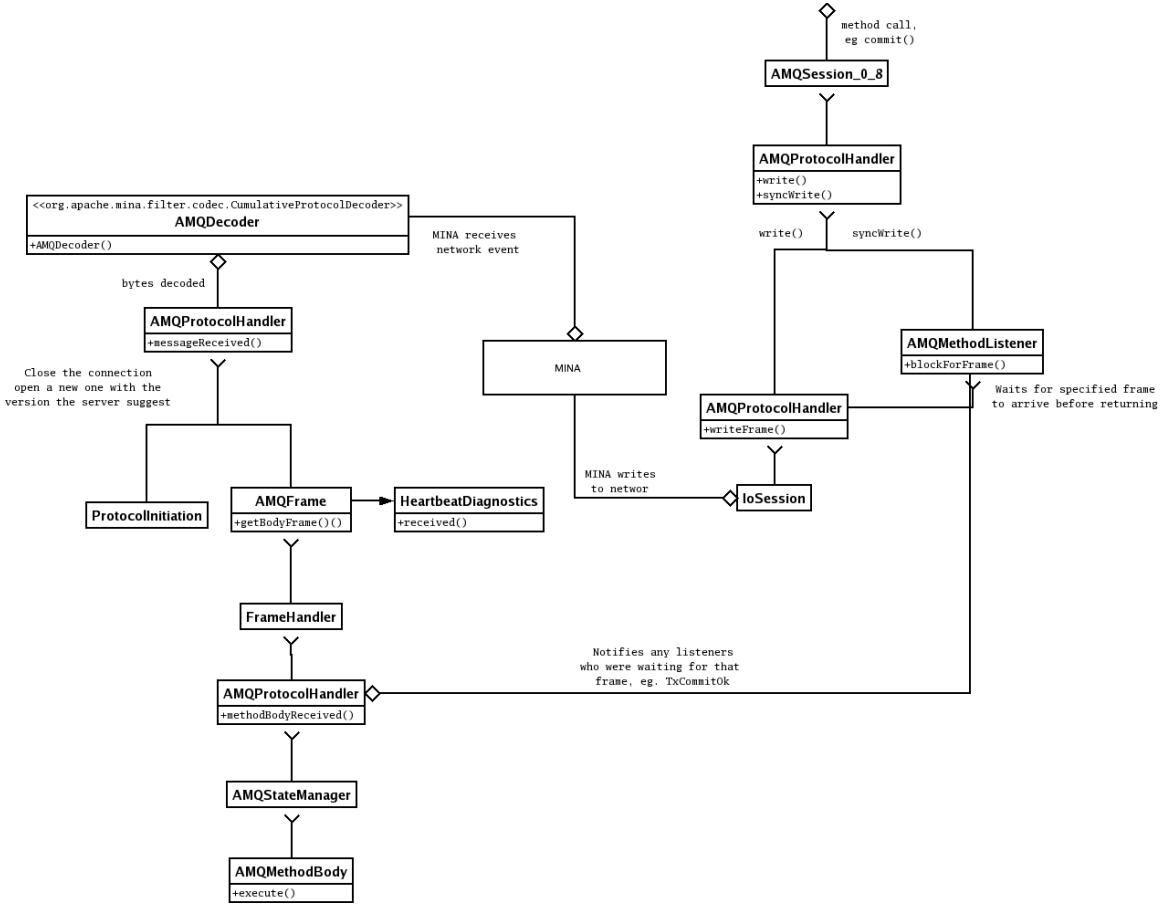
When the client creates a connection it creates an AMQConnectionDelegate for the protocol version it requires and passes the new protocol handler to TransportConnection which creates a socket of the requested type (new TCP socket, existing TCP socket or InVM). It then attaches the socket to the protocol handler which init()s a new ProtocolSession which begins version negotiation with the broker.



Client processing

Once a socket has been opened the client processes data similarly to the broker, decoding frames using AMQDecoder and passing them to AMQProtocolHandler which, normally, calls a frame handler to perform the actual work. If this frame is one which has a listener waiting for it, those listeners are notified.

Outgoing data is generated in AMQSession or its delegate and written to AMQProtocolHandler, optionally with a return frame to wait for. This is passed to Mina directly.



If the frame is a BasicDeliver containing message payload, it adds an UnprocessedMessage to the session which then waits for the ContentHeaderBody and ContentBody payloads to arrive. Once all the expected bodies have been received, the complete message is given to the AMQSession for that channel.

The AMQSession instance adds the message to its internal delivery queue and any locks waiting on the queue are notified. The Dispatcher thread takes the message and delivers it to one of the consumers.

The BasicMessageConsumer converts the UnprocessedMessage to an AbstractJMSMessage and then either delivers it to MessageListener if one has been set or stores it on a queue which is popped when the application calls the consumers receive() method.

MessageProducer.send() behaviour

In network terms, the semantics of the current (0.5) implementation of BasicMessageProducer_0_8 is as follows. The message is split into an AMQP ContentHeader frame and one or more ContentBody frames. These frames are then combined back into an CompositeAMQDataBlock and passed to a MinaProtocolSession.write(), which places it onto the client side outgoing MINA buffers for writing.

At this point, send() returns. It has not actually sent the message data to the broker, nor has the broker accepted that data.

Multiple Java Brokers - Use Cases

Purpose

This page is intended to outline the known use cases for running multiple Java Brokers, addressing logged issues and limitations of the current implementation (as of V0.5). It is **not** about clustering proper.

Use Cases

High Volume Transient Broker

Description

This use case relates to applications with a high residual message load i.e. where message data on the broker remains in memory for some time or consumption lags production such that a backlog is constantly present in the broker queues.

This paradigm is reasonably common, partly because publication threads are generally handling only the simple publish call where we often see consumption threads handling writes to RDBMS or other time expensive processing. Thus a rate gap opens up, and creates a data tailback.

In this scenario, particularly for deployments on a 32bit VM, the broker can exhaust a 3GB heap or start to perform poorly as it approaches max heap.

Result

Broker side OoM or performance degradation requiring bounce. Messages in flight not processed, client applications experience connection loss.

Possible Solution A - Load Balancing Module

For our end users, we could potentially reduce the hassle factor in running 2 brokers by providing a solution comprised of load balancing module which would reside alongside the broker i.e. on server side. This module would intercept published messages and share them between multiple brokers (scaling according to app parameters). Consumers would require multiple connections, but publication would be unaffected and the burden of load balancing could be shifted from the user application to Qpid.

Possible Solution B - No Message Order, 2 Brokers

In this scenario, it would be possible to use 2 brokers and message order would not matter. Publishing clients would use 2 connections and publish alternately to each broker, providing a simplistic load balancing solution. Consuming clients would then consume from the 2 brokers, using the same topic name etc. The consumer could choose to consume in parallel, thus potentially speeding up processing time or by taking messages singly from the two sources alternately.

Possible Solution C - With Message Order, Paired Flow

Again, using 2 brokers but this time working with the assumption that the application data flows across the broker can be separated by source/destination. An easy solution for this is to simply divide the required traffic by source or destination and put an amount on each broker.

This may necessitate multiple consuming connections (to each of the brokers) on the client side where there are multiple sources feeding the same client. Alternatively, for some applications, the clients can be segmented in pairs of publisher-consumer by flow.

Possible Solution D - Redirect to Passive Broker

An alternative approach might be to monitor heap use on the primary broker and kick off a second broker once the first is under heavy load. Client connections (publishing and consuming) would required to be redirected to the secondary broker until the first broker recovers. This is a kind of active-passive pair approach, indeed the secondary broker could be up all the time and simply redirected to as required. Rob mentioned that AMQP 1.0 has the concept of redirect, so it may be something we could look at to inform the solution.

There are some questions around a redirect solution:

- might require an ability to manually override on an incoming connection so that if broker 1 is maxed out due to a down consumer, on restart that consumer can drain broker 1.
- could the console be used to redirect connections - allowing operate control, or possibly using a JMX script or similar (with required MBean method support added) to allow us a cheap solution not broker oriented ?

Considerations

1. Where message order is important, only a solution which separates flows in pairs could be used
2. Failover ?
3. Management of the brokers might need some scripting such that they can be brought up & down as a pair, to at least black box the operations cost
4. ? Priority Queues ...

Java Client Test Coverage

Goal

This page aims to collect information about the current test coverage for the JMS client.

Note this doesn't cover unit tests as there are proper tools to analyse that. This is specifically focused on creating an inventory of our system tests, integration tests etc.

Hopefully this can help us identify gaps, duplication etc..

Structure.

I haven't really thought about a structure yet, hoping to figure it out as we collect more content. For now I will just list the tests under specific functional areas.

System Tests/Functionality

Connection

| Test | Class | package | notes |
|-----------|---------------|----------------------------------|-----------------|
| prefetch | AMQConnection | org.apache.qpid.test.unit.client | testPrefetch() |
| heartbeat | AMQConnection | org.apache.qpid.test.unit.client | testHeartbeat() |

Performance Tests

Interop Tests

ACL Design

- Current ACL Design
- Andrew Kennedy's Proposal For ACL v3
- [Rajith Attapattu's Proposal For ACL v3]

andrew acl proposal

ACL Implementation

See also [Method Considered Harmful](#) and [Method Considered Harmful Redux](#) for discussion on the *METHOD* object type and its implications.

Use Cases

1. Allow access to broker functions to be controlled by an ACL, with the checks being carried out independantly of the mechanism used to access the broker. This would mean that a single `CREATE_QUEUE` permission would apply whether the queue was created when a user logged in and used it, or if that user connected to the broker via JMX or QMF and used the management operations to create the queue.
2. Permissions must be definable at a virtualhost level, with fallback to global permissions. This allows access to be granted for operations only on a certain host, while global operations such as broker administration can be defined at the global level. It also allows default behaviour to be specified globally and then overridden on a per-host basis.
3. The ACL mechanism controls access to operations on particular objects for all users, if at least one user has a rule controlling access to that operation on that type of object. This means that all users requiring access to a particular operation must be configured. The default behaviour will be to deny access.
4. It should be possible for the addition of one access control rule to trigger the addition of other rules, to simplify creation of rulesets.
5. The behaviour of the access control mechanism should be configurable.

Plugin interaction

The plugins can return four different values - *ALLOWED*, *DENIED*, *ABSTAIN* and *DEFER*. Since we can have two plugins of the same type looking at a particular access request, one for the virtual host and the other for global, the results interact as follows:

| Host | Global | Result |
|---------|-------------|---------------|
| ALLOWED | <i>any</i> | ALLOWED |
| DENIED | <i>any</i> | DENIED |
| ABSTAIN | ALLOWED | ALLOWED |
| ABSTAIN | DENIED | DENIED |
| ABSTAIN | ABSTAIN | ABSTAIN |
| ABSTAIN | DEFER | <i>global</i> |
| ABSTAIN | <i>none</i> | ABSTAIN |
| DEFER | ALLOWED | ALLOWED |
| DEFER | DENIED | DENIED |
| DEFER | ABSTAIN | <i>host</i> |
| DEFER | DEFER | <i>global</i> |
| DEFER | <i>none</i> | <i>host</i> |

The *host* and *global* entries in the Result column indicate that the default answer for that plugin should be returned.

ACL File

The access control file consists of a series of rules, describing the permissions granted to users or groups for operations on object types, with specific properties. these are all restricted to certain values, as illustrated by the following lists of tokens:

Permission

ALLOW, ALLOW_LOG, DENY, DENY_LOG

Operation

ALL, CONSUME, PUBLISH, CREATE, ACCESS, CONNECT, BIND, UNBIND, DELETE, PURGE, UPDATE, (ADMIN ?)

ObjectType

ALL, VIRTUALHOST, QUEUE, TOPIC, EXCHANGE, BROKER, LINK, ROUTE, METHOD, (USER, LOG, CONFIG ?)

ObjectProperty

ROUTING_KEY, NAME, QUEUE_NAME, OWNER, TYPE, ALTERNATE, INTERNAL, NO_WAIT, NO_LOCAL, NO_ACK, PASSIVE, DURABLE, EXCLUSIVE, TEMPORARY, AUTO_DELETE

The ObjectProperties are keys that are listed as `key = value` pairs after an Operation/ObjectType combination. They **must** be in this format; a lone string is not accepted here. This is to make the ACL entries less ambiguous.

Allowed Combinations

The object types and operations are related, with only certain combinations allowed. The table below lists allowed combinations with `y`. The rows contain ObjectTypes and the columns Operations.

| | CONSUME | PUBLISH | CREATE | ACCESS | BIND | UNBIND | DELETE | PURGE | UPDATE | EXECUTE |
|-------------|---------|---------|--------|--------|------|--------|--------|-------|--------|---------|
| VIRTUALHOST | | | | y | | | | | | |
| QUEUE | y | | y | | | | y | y | | |
| TOPIC | y | | y | | | | y | | | |
| EXCHANGE | | y | y | | y | y | y | | | |
| BROKER | | | | y | | | | | | |
| LINK | | | | | | | | | | |
| ROUTE | | | | | | | | | | |
| METHOD | | | | y | | | | y | y | |
| OBJECT | | | | y | | | | | | |

See [Method Considered Harmful Redux](#) for more information on how `METHOD` and `OBJECT` are intended to work.

ACL Configuration

These are true/false properties that can be specified to configure the ACL mechanism further, and would be added to the start of an ACL file.

- **transitive** If true, the creation of ACLs, so that if, e.g. `CREATE QUEUE` is permissioned, appropriate `ACCESS VIRTUALHOST` and `BIND EXCHANGE` permissions would also be added.
- **defaultdeny** Sets the default result to `DENIED` if true.
- **defaultallow** Sets the default result to `ALLOWED` if true.
- **expand** Expands synthetic objects, such as `TOPIC` if true.
- **controlled** If there are no access controls on a particular object and operation, the result should be to abstain, whereas if the controlled configuration property is true, the request should be denied.

Syntax

1. Whitespace is considered to be any ASCII byte with a value below 0x20, and is ignored when it occurs between tokens.
2. Continuations using the '\ character (ASCII 0x5c) are allowed anywhere on a line, and can consist of a blank line with a continuation character as the last non-whitespace token
3. Comments are line-style comments, and any text after an un-quoted '#' (ASCII 0x23) are ignored, including continuations. The '#' character may appear in a quoted string.
4. Quoted strings consist of any ASCII inside matching pairs of "" or """ (ASCII 0x27 and 0x22) characters, including any otherwise special characters.
5. Tokens are **NOT** case sensitive, but quoted strings **ARE**.
6. The '=' (ASCII 0x3d) character is special, and is used to indicate property value assignment.
7. Wildcards are specified using the '*' (ASCII 0x2a) character in a property value string, which may be quoted.

The declarations are as follows, using some kind of grammar, with + and * having the usual regular expression meanings, parenthesis denote grouping and brackets denote optional elements.

```
CONFIG ( <config-property> '=' <TRUE | FALSE> ) +
GROUP <group-name> ( <username | group-name> ) +
[ <number> ] ACL <permission> <username | group-name | ALL> <operation> [ <object-type> (
<property-name> '=' <property-value> ) * ]
```

This allows a rather looser and more readable style for ACL files, while still retaining the ability to read the stricter files accepted by the C++

broker. Bear in mind that the group declarations are to be deprecated, in favour of an external directory service, using a plugin mechanism.

The initial <number> is used to allow rulesets to be created which allow individual rules to be enabled and disabled using an admin interface, and an ACL file using numbered lines would be restricted to having increasing numbers per rule, although gaps would be allowed to enable rules to be inserted later, again using an admin interface. This administrative interface would also allow saving of a modified ruleset and re-loading.

Examples

```
Allow "adk@iterator.co.uk" Create Queue \
    Owner="adk@iterator.co.uk" Routingkey = "chocolate biscuits" \
    QueueName="kitten.*"
```

```
# allow adk to create queues
Allow "adk@iterator.co.uk" Create Queue \
    Owner = "adk@iterator.co.uk" \
    Routingkey = "chocolate biscuits" \
    QueueName=kitten
```

```
# allow adk access to this virtual host
110 ALLOW "adk@iterator" ACCESS VIRTUALHOST

# allow creating temporary queues and queues with names matching adk.*
210 ALLOW-LOG \
    "adk@iterator" BIND EXCHANGE \
    routingKey="adk.*" \
    name="amq.direct" # allow adk.* queue bind to amq.direct
220 \
    ALLOW-LOG "adk@iterator" BIND EXCHANGE \
    routingKey="tmp.*" name="amq.direct"
230 ALLOW "adk@iterator" CREATE QUEUE name="adk.*" owner="adk@iterator"
240 ALLOW "adk@iterator" CREATE QUEUE temporary="true" owner="adk@iterator"

# allow publish and consume of messages on the queues
310 ALLOW "adk@iterator" CONSUME QUEUE name="adk.*"
315 ALLOW "adk@iterator" PUBLISH QUEUE routingkey="adk.export#extra" # foo
320 ALLOW "adk@iterator" PUBLISH QUEUE name="adk.*"

# default deny
910 DENY ANY ALL ALL
```

Method Considered Harmful

A lot of the object types and operations used in the ACL file are shared between the Java and C++ brokers and are non-contentious, since they represent actual objects that exist in AMQP - broker, queue, exchange and so forth. What appears to be at issue is how to permission extra functionality in the broker, such as administration of user accounts or logging levels. The C++ broker's 'METHOD' object is one mechanism, and results in ACL lines that specify a single method or set of methods that can be executed, and does not convey whether these are reading, writing or have other side effects on the broker. An example is shown below:

```
ACL ALLOW adk UPDATE METHOD name=getLoggingLevel
ACL ALLOW adk UPDATE METHOD name=setLoggingLevel
ACL ALLOW adk UPDATE METHOD name=reloadLoggingConfig
```

This seems to be at the wrong level of abstraction. Looking at this in a general fashion, there are three things we wish to do to objects: get a property, set a property and execute an operation. These can be mapped to READ, WRITE, EXECUTE or GET, SET, INVOKE, ACCESS, UPDATE, ADMIN, and so on as operations. The next step would be to decide what the object type is that is being manipulated. I would be happy for this to be one of the existing AMQP objects, including BROKER, since this follows the existing pattern of permissions. Another point to note is that existing mechanisms such as JMX already have the conceptual split into these three types of action.

If we abandon the METHOD object in favour of existing object types, we still need to be able to permission such items as users and logging, and I propose these are made part of the broker object, with the possibility of adding other, vendor-specific extensions too. This would result in ACL lines as shown below, which would grant permission to view attributes of the logging subsystem, update those attributes and execute other administrative actions. Finally, if there is a management schema change and the names of methods used change, or new methods and attributes are added, the ACL file does not have to be changed, since the permissions relate to subsystems or extensions.

```
ACL ALLOW adk ACCESS BROKER extension=logging
ACL ALLOW adk UPDATE BROKER extension=logging
ACL ALLOW adk ADMIN BROKER extension=logging
```

or

```
ACL ALLOW adk ADMIN BROKER subsystem=acl
```

If we want to create an ACL file format that is usable across AMQP brokers, then the use of 'extension=<name>' or 'subsystem=<name>' with a set of pre-defined names, say 'logging', 'users', 'configuration', and a naming convention to prevent clashes, such as 'x-<vendor>-*' for vendor specific implementations or just 'x-*' for experimental extensions/subsystems seems appropriate.

Method Considered Harmful Redux

Method Madness

"Though this be madness, yet there is method in't."

Polonius, Hamlet Act 2, scene 2

Introduction

The main point of contention in the ACL debate seems to centre around the mechanism used to permission non AMQP entities, the current *METHOD* method is felt to be unsuitable. This document proposes an update to this syntax and describes exactly how it should be interpreted across brokers.

Access Control

The things that are being controlled or permissioned by entries in the access control list are objects that form part of the Qpid broker. These entities could reasonably be said to be *children* of the broker, although I don't feel that a tree-type structure is either helpful or necessary here, since there is no parallel in the Qpid or AMQP internals. A flat *object type* space has therefore been assumed, continuing current behavior. These types of object have until now simply represented the major types of object that exist and are manipulable inside a broker. The only addition is that of the broker itself, since there are some operations and actions that can only realistically be said to be performed globally. This is the rationale behind such proposed ACL entries as:

```
ACL ALLOW robot ACCESS LOG
ACL DENY robot UPDATE CONFIG
ACL DENY kitten UPDATE USERS
ACL ALLOW kitten ADMIN LOG
```

The *LOG*, *CONFIG* and *USERS* object types here represent subsystems or components or simply collections of management methods that perform a similar set of tasks. They are **not** actual broker objects, although (see later) they may be QMF classes, with their own management schema and package.

A different approach to access control for these management methods relies on the *BROKER* object type being used for permissioning, giving rise to ACL entries as follows:

```
ACL DENY robot ACCESS BROKER
ACL DENY kitten UPDATE BROKER subsystem=logging
ACL ALLOW kitten ACCESS BROKER method=get*
ACL ALLOW kitten ACCESS BROKER method=invoke*
ACL ALLOW kitten MANAGE BROKER subsystem=users
```

This *BROKER* object type represents the entire runtime entity, and is in fact represented in the QMF management schema, with properties, statistics and methods available. This is not meant to indicate a preference for QMF as a final reference point, it should be noted, rather this is illustrative of the sorts of entities an access control object type could map to.

```
<class name="Broker">
  <property name="name" />
  <property name="systemRef" />
  <property name="port" />
</class>
```

Existing Syntax

The previous ACL entries would all be permissioned using the *METHOD* object type in the current C++ broker, assuming a logical extension of the existing syntax. The problem with this syntax is that it is very closely coupled to the management framework, QMF. Also, the granularity of the controls falls awkwardly between extremes, and requires too much specificity to enumerate all methods dealing with a particular area of interest when controlling that type of access, and not distinguishing between **getName** methods on various different managed objects. This makes it impossible to correctly permission access to multiple objects with similarly named methods. Also, since JMX provides access to properties using a method call, a permission for that method would need to be created to allow *READ* access to a property, which blurs the distinction between methods and properties.

Mechanism versus Meaning

Since the current C++ implementation is based exclusively on QMF, only features supported and used by QMF are available. It is preferable to have a mechanism-agnostic access control specification, since QMF and JMX will not be the only management entry-points for ever, with

SNMP and other industry standards available as well as future JEE development. Also, it should be possible to permission access in a manner that does not depend on the version of the QMF schema or API, depending only on the existence or not of particular manageable objects within the broker. This means that when a new method or attribute is added at an API change, or a method name is changed, existing ACLs will have the same meaning as before. This semantic preservation is the aspect of the ACLs that is most important.

The existing object types all relate to the Qpid broker objects, and the best way to move forward is to maintain that relationship, and ensure that all operations have the correct meaning and are controlled correctly in the broker, no matter how they are accessed. This means that an ACCESS QUEUE ACL entry would entail granting permission to view the properties of a queue via the QMF console, via a JMX console utility or the JMX API and by interrogating the queue over JMS or through the C++ AMQP client.

Operational Constraints

The ACCESS operation is assumed here to map to some kind of read-only access. Typically in a software management system the following three types of operation are available:

- **Read** - Access the contents of an attribute, statistic or property.
- **Write** - Set the contents of an attribute or property.
- **Execute** - Call a method or take an action or operation.

It is proposed that the existing operations are maintained, along with the mappings to object types they are allowed to manipulate (as described in a previous text) and the three types described above are mapped as follows:

- ACCESS - Read
- UPDATE - Write
- EXECUTE - Execute

ACCESS continues to describe simple, read-only property or attribute access, mapping nicely a JMX intent of INFO or a **get** type of operation. UPDATE would be used for read-write access to properties, when the operation carried out is a simple change of value with no side effects. The new EXECUTE operation replaces the contentious ADMIN or MANAGE described previously, and more accurately describes the execution or invocation of an administrative action or operation with a particular effect on the broker.

Brokerage

The BROKER object type is to be used to control access to any new set of features. For example, if it is desired to add an ACL entry that will allow the *robots* group to read and write properties on the **Acl** QMF managed object, and additionally to execute all methods that are present, this could be done as follows:

```
ACL ALLOW robots ACCESS BROKER package="org.apache.qpid.acl"
ACL ALLOW robots UPDATE BROKER package="org.apache.qpid.acl"
ACL ALLOW robots EXECUTE BROKER package="org.apache.qpid.acl"
```

If a logging subsystem was added, with the QMF management schema package defined as **org.apache.qpid.log** and methods such as **setLoggingLevel**, **getAvailableLoggingLevels**, **reloadLogFile**, **rollLogFile** are defined, along with properties like **currentLevel** and **lastLogTime** then it could be permissioned this way:

```
ACL ALLOW robots ACCESS BROKER package="org.apache.qpid.log"
ACL ALLOW robots UPDATE BROKER package="org.apache.qpid.log"
ACL ALLOW kitten EXECUTE BROKER package="org.apache.qpid.log" method="rollLogFile"
ACL ALLOW robots EXECUTE BROKER package="org.apache.qpid.log" method="reloadLogFile"
ACL DENY robots EXECUTE BROKER package="org.apache.qpid.log"
```

In this example, the *robots* group can only execute **reloadLogFile** while *kitten* (a member) can also execute **rollLogFile**, and the group has read/write access to all properties and statistics. It should be obvious that there is scope for adding arbitrary new packages and then permissioning them. Also, if the contents of the packages are well defined and they are suitably finely grained then it will mostly suffice to permission at the package level for all operations and properties. This gives freedom to update APIs and add new methods without making ACL files obsolete or causing security issues, since the **meaning** of the ACL entries should be unchanged.

Care will need to be taken with, for example, JMX **invoke** method, which offers a level of indirection that could enable bypassing access checks. This is currently handled at a common JMX entry point, and should suffice at present.

Syntactic Sugar

In an attempt to divorce the ACL syntax from the mechanism further, it could also be possible to remove references to the package and use a different naming scheme, which would have a mapping to QMF, JMX managed objects and any future management information repository. This could work as follows, with **users** mapping to the JMX **UserManagement** MBean and a QMF **org.apache.qpid.users** package with a **Users** class. The change to the ACL syntax is trivial. Additionally, changing all properties to simply *name* would standardise the syntax further, with only a small loss of readability.

```
ACL ALLOW kitten EXECUTE BROKER component="users" name="*"
ACL ALLOW kitten UPDATE BROKER component="users" name="fileName"
```

It is possible that another object could be chosen instead of **BROKER**, such as **METHOD** (however this gives the issue of changing the meaning of existing files) but this would not change the discussion presented above. The only issue might be that it is cumbersome to add a

permission granting access to all management methods, properties and statistics, both read and write, at the same time. Even though *UPDATE* will usually include *ACCESS* permissions, two lines are needed (for *EXECUTE* separately) unless it is satisfactory for *EXECUTE* to include *UPDATE* (and hence *ACCESS*) rights.

```
ACL DENY robot EXECUTE METHOD component="config" name="reload*"
ACL ALLOW kitten ACCESS METHOD component="config" name="*"
```

It could be pointed out that *ACCESS METHOD* ought more correctly to read *ACCESS PROPERTY* and similarly for *UPDATE*, however it is felt that the number of object types should be kept to a minimum, which is the purpose of this proposal.

Conclusion

The ACL changes described above are fairly simple, but should provide a sensible and easily extensible syntax that will allow both the Java and C++ brokers to provide fine grained access control for custom components that are specific to each implementation without compromising cross-platform file compatibility. The actual access and management mechanisms for the brokers do not impact the ACL syntax, which remains agnostic, and also maintains its meaning through API upgrades and extensions without compromising platform security.

There are several options for ACL file syntax describing access to methods controlling a (for example) logging mechanism, which are summarised below:

- Extra object type created for each set of management methods, with unique per-object set of properties. This is the least extensible mechanism.

```
ACL ALLOW kitten EXECUTE LOG
```

- Specify all method names to be permissioned as in the current C++ broker implementation, using *METHOD* as the object type. This does not allow property access to be permissioned or prevent name clashes in different managed object classes.

```
ACL ALLOW kitten UPDATE METHOD name="methodNameOne"
ACL ALLOW kitten UPDATE METHOD name="methodNameTwo"
```

- Use *BROKER* or *METHOD* object type and specify the component or subsystem by an arbitrary name, with the option to specify down to individual methods by adding a wildcarded name pattern. Using *METHOD* in this way is close to the current C++ syntax. Alternatively, the QMF package name could be used to identify the component. Different operations are used to describe access to properties or attributes.

```
ACL ALLOW kitten EXECUTE BROKER subsystem=logging
ACL ALLOW kitten EXECUTE BROKER package="org.apache.qpid.log"
```

```
ACL ALLOW kitten EXECUTE METHOD component="log" name="reload*"
ACL ALLOW kitten UPDATE METHOD component="log"
ACL ALLOW robot ACCESS METHOD component="log"
ACL ALLOW robot EXECUTE METHOD component="acl" name="reload*"
ACL DENY robot EXECUTE METHOD component="config" name="reload*"
ACL ALLOW robot EXECUTE METHOD component="config"
```

The last described syntax is close to a preferred option, and incorporates features from recent dev.qpid.apache.org discussion, although any updates or suggestions are welcome.

(**Note** this has the interesting/useful feature/bug of falling back to C++ broker syntax if the *component* property is omitted. This would be legal in the Java broker, just not recommended.)

Post Scriptum

The following points should clarify some of the proposed features, however the syntax as described in the Conclusion is intended to represent the preferred usage.

In the C++ broker there exists a feature whereby plugins, uniquely identified by a schema package and a class name, can have ACLs applied to them. This will also become available in the Java broker, and would be permissioned using the *OBJECT* object type. This allows objects that are external to the broker to be controlled. For the Java broker it is intended that the main class for a plugin would check with the security manager using the Java package and class names as properties, as below.

```
ACL ALLOW kittens ACCESS OBJECT package="com.example.plugin" class="Example"
```

When management functions are being permissioned, a symbolic name for a logical grouping of related methods, properties, attributes and operations is used to identify what is being controlled. This is identified using the *component* property in the examples above. These groupings will map onto JMX managed objects or MBeans, QMF management schemas, or some other form of management object. It is

intended that a particular broker implementation handles these mappings internally and ignores mappings that do not exist, such as logging management on the C++ broker currently. It is also possible to offer finer grained control by specifying the `name` property for the ACL entry, thus restricting the scope to a single method or property. It *may* also be possible to specify other properties that have meaning for a particular broker implementation, thus maintaining backward compatibility. The list of possible property names should be fixed as part of the definition of the ACL file format.

Andrew D Kennedy, andrew.international@gmail.com, 2010-05-20

AMQP Distributed Transaction Classes (C++)

Overview

This page describes the classes involved in handling distributed transactions in the C++ broker. The store plugins are not described; only the classes in the broker proper are described here. The store plugin modules should have separate documentation.

Much of the code is common for local/one-phase transactions; however, this document is primarily concerned with distributed/two-phase transactions.

The basic approach taken in the C++ broker is that, once a transaction is begun, enqueue/dequeue commands received from a client are recorded in a list of operations associated with the transaction but not actually carried out until the transaction is committed (one phase) or prepared (two-phase).

Classes

These classes are all in the `qpid::broker` namespace.

Record-Keeping Classes

- `DtxManager`. Maps xid to a `DtxWorkRecord`.
- `DtxWorkRecord`. Refers to work to be done under the transaction using a vector of `DtxBuffer` objects.

| How could this have multiple DtxBuffer objects associated? When dtx-start is called with the join flag set, the subsequent work is added to a set of operations previously performed (perhaps on a different session). [Gordon]

- `DtxBuffer`. Per-xid list of operations requested under the transaction. The operations are derived from `TxOp`. *[As above, each buffer contains the operations between one pair of start/end calls. Where end has the suspend flag set and a subsequent start has the join flag set, a new buffer will be populated and associated with the xid - Gordon]*
- `TxPublish`. Publish (enqueue) operation record. Has a message and list of Queues the message will be available on; message routing is done when the publish command is received although the actual delivery to the queue(s) happens when the transaction is prepared.
- `TxAccept`. Accept (dequeue) operation record. The point at which a dequeue is recognized depends on acceptance policy; it may be on delivery/ack or when message is explicitly accepted.

Operation Classes

- `SessionAdapter`. Handles operations on a session; has a `SemanticState`.
- `SemanticState`. Implements the handling of transactional commands start, publish, etc.
- `RecoveryManagerImpl`. Involved in recovering prepared distributed transactions. This is a target that the store module calls to rebuild the record-keeping objects above. The primary methods in transaction recovery are:
 - `recoverTransaction`. Associate an xid with a `TPCTransactionContext`. Returns a `RecoverableTransaction`.
 - `RecoverableTransaction::enqueue`, `dequeue`. Rebuilds the records of what message enqueue/dequeue operations have been prepared and may be committed or rolled back. *[The current recovery of prepared transactions loses the original position of consumed messages in the queue, thus if the transaction is then rolled back, the messages may be requeued in the wrong place - Gordon].*

API Error Conditions

Types of errors that are possible:

- connection failure (after connection has been established)
- connection cannot be established
- missed heartbeats
- authentication errors
- authorisation errors
- queue not found
- exchange not found
- other address validation errors (e.g. exchange is of different type to that specified)

- session terminated by management
- connection terminated by management
- broker side queue limit breached
- exclusive subscriber violation
- feature not supported exception from broker (e.g., unsupported exchange type)
- protocol violation errors (by which I mean any sort of framing error or problem that is a result of a bug in the library or broker rather than being an invalid command by the application)
- internal error in broker, i.e. some as yet unidentified bug, misconfiguration or environmental problem
- internal error in client: same thing for client-side misconfiguration or bugs.
- Exceeded client-side limitation: a client-side queue is over-full. E.g. user has not respected flow control limits on the client side.
- Transaction related errors - not sure what the set is, probably just copy from spec exceptions.

Impact of errors:

- session no longer valid[1]
- connection no longer valid[1]
- transaction aborted
- no longer term impact, specific call simply failed

[1] May want to distinguish between amqp 0-10 session and sessions in the messaging api here. E.g. even if a particular condition kills the amqp 0-10 session/connection, that may not prevent recreating the session or reconnecting.

Broker Management QMF Coverage

Broker Management via QMF

As part of the effort to bring more commonality between the Java and C++ brokers I have been attempting to implement QMF management in the Java Broker. *Currently the Java Broker can process the majority of the QMF Management commands (although it will not accept agents connecting to it).* Ultimately the aim must be to allow management and configuration of the brokers using the same set of tools.

In implementing QMF management I have noted a number of questions - some general and related to the nature of QMF, some more specific to the particular set of entities and methods in the management schema for the Qpid (C++) Broker, and some simply relating to the definition of individual properties or statistics. I have also noted where I think additions to the existing schema would be useful.

Currently, even if fully implemented, management through QMF would not be sufficient to duplicate the existing management functions available in the Java Broker (e.g. through the JMX console).

General Comments on the Schema

- Why are there no methods to create entities? Although one can use AMQP commands to accomplish this task it means you have to switch between two distinct command sets to manage the broker; it also means you cannot simply piggyback the QMF management over other transports. (further, for AMQP 1-0 we are removing "management" commands from the core protocol)

GS: See QPID-2317; I'd like using this to be as simple as sending a correctly formatted/encoded message to a special address. I believe with QMFv2 that will indeed be the pattern. It is a useful addition as the new APIs in c++ and python aim to be protocol independent and thus do not expose the 0-10 declare methods.

- There are no tools for inspecting the contents of messages within the broker. Where a message is stuck on the queue, it is useful to be able to inspect its properties and/or content.

GS: You could do this by browsing the queue (though there is no support then for moving or modifying a message). A scheme that avoids duplicating too much of the standard messaging behaviour would be good.

RG: the problem with that is that it requires you to be on an AMQP 0-10 connection. I would like it to be that we can conduct QMF over other protocols, namely AMQP0-9 or JMX. I understand your concern about duplicating functionality... as you already mentioned, moving can't be done easily anyway, but I see this inspection somewhat differently to browsing... and (in particular) without some form of server side selector I don't think the C++ broker could do what I am talking about (pick an arbitrary message from the queue and inspect it).

- Values with High/Low watermarks... the historic low for these will almost invariably be 0, so what is the point?
- Absolute times are specified in nanoseconds... this is "unusual" and it's pretty much impossible to get an accurate absolute time in nanoseconds. Milliseconds for absolute time and nanoseconds for deltas would make more sense.
- The model and methods are not designed to cope with a broker which can support multiple virtual hosts.
- It would be nice to have the ability to configure dynamic event configuration... i.e. the ability to say "publish an event to this address

every time this condition is met". You can build this on top of the statistic updates, but it requires you to have a fairly frequent management publish period and a process actively monitoring the output.

* GS: I'm very much in favour of aligning the two brokers. We would however need to come up with a mechanism for allowing the c++ broker to retain backwards compatibility with a schema as currently defined. E.g. support for multiple schemas?

- The java broker exposes an interface for conducting live User Management, eg add/delete/view users, set password, set management access rights (read only, read/write, or admin which is read/write plus access to the more sensitive management mbeans such as UserManagement itself). It would be good to expose this type of management via QMF as well.
- The Java broker exposes an interface for viewing and adjusting its logging levels while running. This too would be useful to expose via QMF.

General comments on QMF

- have a generic way to be able to create new entities,
- have a way to modify the mutable properties of existing entities.
- when sending schema, where a particular value is of a restricted domain (e.g. has min and/or max values; or takes a value from a restricted enumerated set) then that restricted domain is communicated.
- add ability to "reset" statistic counters... This would actually make low watermark counts useful.
- add ability to query for objects based on criteria (e.g. get me all exchanges whose vhostRef is X)

Comments on Individual Schema Classes

System

Properties

General: what is this supposed to represent - the machine the broker is running on, or the process that contains the broker?

| Name | Type | Description | C++ | Java | Notes |
|----------|------|-----------------------|-----|------|---|
| systemId | uuid | | Y | | Is this a qpid specific property? |
| osName | sstr | Operating System Name | Y | | |
| nodeName | sstr | Node Name | Y | | What exactly is this supposed to be... hostname? |
| release | sstr | | Y | | What exactly is this supposed to be... release of the OS? |
| version | sstr | | Y | | What's the difference between this and release? |
| machine | sstr | | Y | | What is this? |

Broker

Properties

| Name | Type | Description | C++ | Java | Notes |
|------------------|--------|---|-----|------|---|
| systemRef | objId | System ID | Y | | |
| port | uint16 | TCP Port for AMQP Service | Y | | A broker may be listening on more than one port - I suggest that we want to have a new entity to represent listening ports/transports |
| workerThreads | uint16 | Thread pool size | Y | | |
| maxConns | uint16 | Maximum allowed connections | N | | Java broker currently doesn't support a connection limit |
| connBacklog | uint16 | Connection backlog limit for listening socket | N | | What is this? It "defines the maximum length to which the queue of pending connections for sockfd may grow (from man page for the listen() system call)" |
| stagingThreshold | uint32 | Broker stages messages over this size to disk | N | | Java Broker does not currently support staging. The 'staging' functionality is poorly conceived on the c++ broker and needs reviewed; it is really just about handling very large messages by not requiring that the full contents ever be held in memory |
| mgmtPublInterval | uint16 | Interval for management broadcasts | Y | | |

| | | | | | |
|---------|------|---|--|---|--|
| version | sstr | Running software version | | Y | |
| dataDir | sstr | Persistent configuration storage location | | Y | Java Broker currently displays the value of \$QPID_HOME here |

Statistics

| Name | Type | Description | C++ | Java | Notes |
|--------|-----------|-------------|-----|------|---|
| uptime | deltaTime | | | Y | Why have this as a statistic that is going to constantly change? start time as a property would do equally well |

Suggested Additions

- number of current / high watermark of connections

Methods

| Signature | Description | C++ | Java | Notes |
|---|--|-----|------|---|
| (uint32 sequence, lstr body)
echo (uint32 sequence, lstr body) | Request a response to test the path to the management broker | | Y | |
| void connect (sstr host, uint32 port, bool durable, sstr authMechanism, sstr username, sstr password, sstr transport) | Establish a connection to another broker | | Y | Doesn't allow a vhost to be specified, nor is transport defined (is ssl a transport?) yes, ssl would be a transport as would e.g. rdma. Perhaps a URL would be simpler? |
| void queueMoveMessages (sstr srcQueue, sstr destQueue, uint32 qty) | Move messages from one queue to another | | N | Queues are local to vhosts, not brokers... since you may have queues with the same name on two separate vhosts on the same broker - this method cannot be implemented on a broker which supports multiple vhosts. Should also add an arguments map to allow filters to be applied |

Suggested Additions

- ability to create virtual hosts
- shutdown/restart broker?
- reload (security) configuration.

Agent

The Java Broker does not currently support the Agent class

Properties

| Name | Type | Description | C++ | Java | Notes |
|---------------|--------|--|-----|------|-------|
| connectionRef | objId | | | N | |
| label | sstr | Label for agent | | N | |
| registeredTo | objId | Broker agent is registered to | | N | |
| systemId | uuid | Identifier of system where agent resides | | N | |
| brokerBank | uint32 | Assigned object-id broker bank | | N | |
| agentBank | uint32 | Assigned object-id agent bank | | N | |

Vhost

Properties

| Name | Type | Description | C++ | Java | Notes |
|---------------|-------|-------------|-----|------|-------|
| brokerRef | objId | | Y | | |
| name | sstr | | Y | | |
| federationTag | sstr | | Y | | |

Statistics

Suggested Additions

- number (and high watermark) of queues/exchanges
- number (and high watermark) of connections

Methods

Suggested Additions

- create queue
- create exchange
- create binding
- delete queue
- delete exchange
- delete this vhost

Queue

Properties

| Name | Type | Description | C++ | Java | Notes |
|-------------|-------|-------------------------------------|-----|------|-------|
| vhostRef | objId | | Y | | |
| name | sstr | | Y | | |
| durable | bool | | Y | | |
| autoDelete | bool | | Y | | |
| exclusive | bool | | Y | | |
| arguments | map | Arguments supplied in queue.declare | Y | | |
| altExchange | objId | | Y | | |

Statistics

| Name | Type | Description | C++ | Java | Notes |
|--------------------|---------|---------------------------------|-----|------|-------|
| msgTotalEnqueues | count64 | Total messages enqueued | Y | | |
| msgTotalDequeues | count64 | Total messages dequeued | Y | | |
| msgTxnEnqueues | count64 | Transactional messages enqueued | | | |
| msgTxnDequeues | count64 | Transactional messages dequeued | | | |
| msgPersistEnqueues | count64 | Persistent messages enqueued | Y | | |
| msgPersistDequeues | count64 | Persistent messages dequeued | Y | | |

| | | | | |
|---------------------|---------|-------------------------------------|---|---|
| msgDepth | count32 | Current size of queue in messages | Y | Is this with or without unacked messages (i.e. does it include all messages for which the dequeue has not yet been committed)? It includes unacked messages
Also would seem to make sense to provide high watermark for queue depth (bytes and msg) GS: +1 on high watermark for queue depth; would be very useful |
| byteDepth | count32 | Current size of queue in bytes | Y | |
| byteTotalEnqueues | count64 | Total messages enqueued | Y | |
| byteTotalDequeues | count64 | Total messages dequeued | Y | |
| byteTxnEnqueues | count64 | Transactional messages enqueued | | |
| byteTxnDequeues | count64 | Transactional messages dequeued | | |
| bytePersistEnqueues | count64 | Persistent messages enqueued | Y | |
| bytePersistDequeues | count64 | Persistent messages dequeued | Y | |
| consumerCount | hilo32 | Current consumers on queue | Y | |
| bindingCount | hilo32 | Current bindings | Y | |
| unackedMessages | hilo32 | Messages consumed but not yet acked | | It would seem more useful on a per subscription basis (so we can see which subscriptions are holding on to messages without acking them). GS: I agree |
| messageLatency | mmaTime | Broker latency through this queue | | This generates a number of statistics (Max/Min/Avg/Samples) each of which is said to be measured in nanoseconds. What does Samples mean in this context? this isn't actually implemented on c++ yet |

Suggested Additions

- age of oldest message on the queue (and high watermark),
- high watermark for message size
- high watermark for queue depth (bytes)
- high watermark for message count
- messages/bytes expired due to TTL

Methods

| Signature | Description | C++ | Java | Notes |
|------------------------------|---|-----|------|--|
| void purge (uint32 request) | Discard all or some messages on a queue | Y | | Would be nice to add an arguments map so that filters could be supplied. Also making request an IO parameter so you can return how many messages were purged. Finally are the messages supposed to be purged from the head, the tail or randomly throughout the queue? |

Suggested Additions:

- mechanisms to move / copy messages to another queue (with filters to allow specifying arbitrary messages, not first X msg)
- rename this queue
- delete this queue
- mechanism to inspect messages on the queue

Exchange

Properties

| Name | Type | Description | C++ | Java | Notes |
|-------------|-------|--|-----|------|-------|
| vhostRef | objId | | | Y | |
| name | sstr | | | Y | |
| type | sstr | | | Y | |
| durable | bool | | | Y | |
| autoDelete | bool | | | Y | |
| altExchange | objId | | | Y | |
| arguments | map | Arguments supplied in exchange.declare | | | |

Statistics

| Name | Type | Description | C++ | Java | Notes |
|---------------|---------|--|-----|------|--|
| producerCount | hilo32 | Current producers on exchange | N | N | How is this supposed to be calculated? RGem: This is not actually implemented on the C++ broker either |
| bindingCount | hilo32 | Current bindings | | Y | |
| msgReceives | count64 | Total messages received | | Y | |
| msgDrops | count64 | Total messages dropped (no matching key) | | Y | |
| msgRoutes | count64 | Total routed messages | | Y | |
| byteReceives | count64 | Total bytes received | | Y | |
| byteDrops | count64 | Total bytes dropped (no matching key) | | Y | |
| byteRoutes | count64 | Total routed bytes | | Y | |

Methods

Suggested Additions

- rename
- set AlternateExchange
- delete

Binding

Properties

| Name | Type | Description | C++ | Java | Notes |
|-------------|-------|-------------|-----|------|---|
| exchangeRef | objId | | Y | | |
| queueRef | objId | | Y | | |
| bindingKey | sstr | | Y | | |
| arguments | map | | Y | | |
| origin | sstr | | Y | | definition would be good... why is this not in "arguments" (since that is how it is set)? |

Statistics

| Name | Type | Description | C++ | Java | Notes |
|------------|---------|-------------|-----|------|-------|
| msgMatched | count64 | | | Y | |

Methods

Suggested Additions

- delete

Subscription

Properties

| Name | Type | Description | C++ | Java | Notes |
|--------------|-------|------------------|-----|------|---|
| sessionRef | objId | | | Y | |
| queueRef | objId | | | Y | |
| name | sstr | | | Y | |
| browsing | bool | | | Y | is this supposed to be a pre/non acquire indicator? Is this true only when pre-acquiring and accept-mode is none? This is true of the acquire-mode was not-acquired (for 0-10). It is independent of the accept-mode which is exposed via the acknowledged property |
| acknowledged | bool | | | Y | is this simply accept-mode = explicit? yes |
| exclusive | bool | | | Y | |
| creditMode | sstr | WINDOW or CREDIT | | Y | |
| arguments | map | | | Y | |

Statistics

| Name | Type | Description | C++ | Java | Notes |
|-----------|---------|--------------------|-----|------|-------|
| delivered | count64 | Messages delivered | | Y | |

Suggested Additions

- available credit
- unacknowledged msg/bytes
- delivered bytes

Methods

Suggested Additions

- delete

Connection

Properties

| Name | Type | Description | C++ | Java | Notes |
|----------|-------|-------------|-----|------|-------|
| vhostRef | objId | | | Y | |

| | | | | |
|-------------------|--------|--|---|---|
| address | sstr | | Y | Is this the local or remote address? Whichever, we should also have the other one. It is the remote address |
| incoming | bool | | Y | |
| SystemConnection | bool | Infrastructure/ Inter-system connection (Cluster, Federation, ...) | Y | |
| federationLink | bool | Is this a federation link | Y | |
| authIdentity | sstr | authId of connection if authentication enabled | Y | |
| remoteProcessName | sstr | Name of executable running as remote client | | These are a subset of the clientProperties sent by the client - should we not just show the client Props? |
| remotePid | uint32 | Process ID of remote client | | |
| remoteParentPid | uint32 | Parent Process ID of remote client | | |

Suggested Additions

- vhost
- server/client properties maps
- protocol version

Statistics

| Name | Type | Description | C++ | Java | Notes |
|------------------|---------|--|-----|------|----------------------------|
| closing | bool | This client is closing by management request | | | Why is this a statistic??? |
| framesFromClient | count64 | | | | |
| framesToClient | count64 | | | | |
| bytesFromClient | count64 | | | | |
| bytesToClient | count64 | | | | |

Suggested Additions

- attached sessions (with high watermark)

Methods

| Signature | Description | C++ | Java | Notes |
|---------------|-------------|-----|------|-------|
| void close () | | | | |

Link

Properties

| Name | Type | Description | C++ | Java | Notes |
|-----------|--------|-------------|-----|------|---|
| vhostRef | objId | | Y | | |
| host | sstr | | Y | | |
| port | uint16 | | Y | | |
| transport | sstr | | Y | | What are the valid transports? at present, tcp, rdma and ssl(which is over tcp) |
| durable | bool | | Y | | |

Suggested Additions

- remote vhost

Statistics

| Name | Type | Description | C++ | Java | Notes |
|-----------|------|--------------------------------|-----|------|--|
| state | sstr | Operational state of the link | | | Are the valid states defined anywhere? c++ broker currently includes the following states: WAITING, CONNECTING, OPERATIONAL, FAILED, CLOSED, PASSIVE (from Link.h) |
| lastError | sstr | Reason link is not operational | | | |

Methods

| Signature | Description | C++ | Java | Notes |
|--|-------------------------------|-----|------|-------|
| void close () | | | | |
| void bridge (bool durable, sstr src, sstr dest, sstr key, sstr tag, sstr excludes, bool srclsQueue, bool srclsLocal, bool dynamic, uint16 sync) | Bridge messages over the link | | Y | |

Bridge

Properties

| Name | Type | Description | C++ | Java | Notes |
|------------|--------|-------------|-----|------|--|
| linkRef | objId | | | Y | |
| channelId | uint16 | | | Y | |
| durable | bool | | | Y | |
| src | sstr | | | Y | |
| dest | sstr | | | Y | |
| key | sstr | | | Y | |
| srclsQueue | bool | | | Y | |
| srclsLocal | bool | | | Y | |
| tag | sstr | | | | what is this and what does it do? It is used for loop detection and is the value to be used for qpid.trace.id on the subscription queue |
| excludes | sstr | | | | what is this and what does it do? Also used for loop detection and is the value to be used for qpid.trace.excludes on the subscription queue |
| dynamic | bool | | | Y | |
| sync | uint16 | | | | |

Methods

| Signature | Description | C++ | Java | Notes |
|---------------|-------------|-----|------|-------|
| void close () | | | | |

Session

Properties

| Name | Type | Description | C++ | Java | Notes |
|------|------|-------------|-----|------|-------|
| | | | | | |

| | | | | | |
|------------------|---------|--|--|---|--|
| vhostRef | objId | | | Y | |
| name | sstr | | | Y | |
| channelId | uint16 | | | Y | |
| connectionRef | objId | | | Y | |
| detachedLifespan | uint32 | | | | |
| attached | bool | | | Y | |
| expireTime | absTime | | | | |
| maxClientRate | uint32 | | | | |

Suggested Additions

- transactional mode

Statistics

| Name | Type | Description | C++ | Java | Notes |
|-------------------|---------|------------------------------|-----|------|--|
| framesOutstanding | count32 | | | | |
| TxnStarts | count64 | Total transactions started | | | Are these supposed to be tx or dtx transactions? or either |
| TxnCommits | count64 | Total transactions committed | | | |
| TxnRejects | count64 | Total transactions rejected | | | |
| TxnCount | count32 | Current pending transactions | | | |
| clientCredit | count32 | Client message credit | | | |

Suggested Additions:

- subscription count/watermark
- unacked messages/bytes

Methods

| Signature | Description | C++ | Java | Notes |
|------------------------|-------------|-----|------|-------|
| void solicitAck () | | | | |
| void detach () | | | | |
| void resetLifespan () | | | | |
| void close () | | | | |

Java Client Design

Client Design

This page seeks to capture the current state of the Java client and serve as a place to discuss future changes.

Current Design

Change Proposals.

0.6 Java Client Dispatcher Changes

0.6 Java Client Dispatcher Changes

Java Client Dispatcher Changes.

Investigation of QPID-1871 has highlighted a race condition between the Dispatcher and the clients request to rollback.

- Problem Summary
- Operation Details
- Code Problem
- Further Details
- Comment Responses

Problem Summary

The problem here is that the Dispatcher has the ability to hold on to a message so when the rollback process is believed to have completed the *Dispatcher* then rejects the final message AFTER the *TxRollback* so that one message gets sent ahead of the other messages. The reject is dropped as the message has been resent. This is specific to the Java Client causing the Java Broker to return messages out of order. This may be the reason that the *RollbackOrderTest* has been disabled. It is not clear currently if this will also affect the CPP broker. Further investigation is in required.

Operation Details

Due to the way that the *AMQSession.Dispatcher* is paused when a rollback operation is in progress it is possible that the Dispatcher thread is 'holding' a message for dispatch. The main loop of *AMQSession.Dispatcher* is shown here:

```
while (!closed.get() && ((disp = (Dispatchable) _queue.take()) != null))
{
    disp.dispatch(AMQSession.this);
}
```

The problem is highlighted in the *dispatchMessage* call below (which is the result of *disp.dispatch()* on an *UnprocessedMessage*). If the *Dispatcher* is in the process of dispatching messages when a second thread calls rollback then the connection will be stopped and the dispatcher can remove a message from *_queue* and then stop in the *dispatchMessage*

```
private void dispatchMessage(UnprocessedMessage message)
{
    long deliveryTag = message.getDeliveryTag();

    synchronized (_lock)
    {
        try
        {
            while (connectionStopped())
            {
                _lock.wait();
            }
        }
        catch (InterruptedException e)
        {
            // pass
        }

        if (!(message instanceof CloseConsumerMessage)
            && tagLE(deliveryTag, _rollbackMark.get()))
        {
            rejectMessage(message, true);
        }
        else
        {
            synchronized (_messageDeliveryLock)
            {
                notifyConsumer(message);
            }
        }
    }

    long current = _rollbackMark.get();
    if (updateRollbackMark(current, deliveryTag))
    {
        _rollbackMark.compareAndSet(current, deliveryTag);
    }
}
```

When the connection is resumed the deliveryTag of the current message will be 'less than or equal' to the *_rollbackMark* as this has been set to the highest deliveryTag received prior to rollback.

```
_rollbackMark.set(_highestDeliveryTag.get());
```

There are no guards in the code to stop the IO layer adding a new message to `_queue` whilst rollback is in progress. However, both 0-8 and 0-10 ensure that message flow has stopped whilst recovery is processed. The 0-8 sets `ChannelFlow=false` and waits for the `Ok`, in 0-10 the consumers are stopped and a sync performed.

Code Problem

The investigation of this problem has highlighted a two areas which need to be addressed:

1. The ability to ensure the dispatcher is not holding a message.
2. The ability to confirm when the dispatcher will not process any more messages.

How the *Dispatcher* holds a message

The `_queue.take()` call is guaranteed never to return null and once we have entered the `take()` call there is no way to stop the Dispatcher.

```
while (!closed.get() && ((disp = (Dispatchable) _queue.take()) != null))
```

Hence we perform the stop as soon as possible after the `take()`, but this results in us holding on to a message.

Ideally we need to be able to stop the *Dispatcher* whilst it is in the `take()` method.

How the *Dispatcher* can keep processing.

The *Dispatcher* is currently uses the `connectionStopped()` call to suspend its activities when the connection has been marked as stopped. However, we need to know that the *Dispatcher* has actually hit this section otherwise we need to guarantee that the `_queue` is empty.

```
synchronized (_lock)
{
    try
    {
        while (connectionStopped())
        {
            _lock.wait();
        }
    }
    catch (InterruptedException e)
    {
        // pass
    }
}
```

Having the *Dispatcher* signal that it has stopped processing will allow us to know that we have hit the stopped state. However, this will mean that we have the opportunity to process one extra message AFTER the rollback command has been requested.

Further Details

After a discussion with Rafi/Rob on the recent Python changes expending effort in refactoring the client is probably not worth the effort. If the client message delivery were re-written to mirror the approach taken in the Python codebase then it would be simpler and easier to reason about. As a result I have devised a much smaller, though slightly ugly approach that will address our immediate rollback issues. The approach can be found [here](#).

Comment Responses

| User | Comment | via | Response |
|------|---|-------|---|
| rhs | AMQSession.syncDispatchQueue is used in 0-10 for this | email | This will not work if the dispatcher is performing the rollback (Deadlock).
Also we need to stop processing the messages immediately and not allow any further processing. |
| rhs | Agree the client is badly in need of some improvements in maintainability and readability, however in this particular case I don't think moving the rollback processing from one thread to another actually improves the situation significantly. | email | It is not so much moving from one thread to another but from moving from the AMQSession / Dispatcher objects to just the Dispatcher. |

| | | | |
|-----|---|-------|---|
| rhs | I suspect in order do this properly we really need to stop thinking in terms of code being associated with a given thread, and think instead about what locks we have, what data structures those locks protect, and which locks need to be held in order to execute a given piece of code. | email | The focus of this change was to consolidate the operations on the received message. I would like to see a clean interface where messages are passed in for dispatching. The cleaning operations should then be fully contained in that interface not in a couple of locations as it is currently. |
| rhs | Really we need to be able to articulate exactly what locks the client has, what data structure(s) each lock protects, and what order should be used to acquire multiple locks when necessary. | email | Agreed, documenting what we have and how it works would be very useful for this discussion. |

0.6 Java Client Dispatcher Changes - Details

Approach to addressing Dispatcher issues.

Looking at the current rollback implementation we have two different approaches. One for 0-8 and one for 0-10. The change is being driven because the 0-8 has a race condition causing message ordering of a single message to change. The 0-10 implementation works for the *receive()* case but has a deadlock in the *onMessage()* case.

Recent work in the Python client would suggest that the better approach to implementing a client would be to use fewer locks and to remove the distinction between the *onMessage()* and the *receive()* cases. Putting the Message Listener thread in to the application space brings a large amount of simplification as there is no difference between the synchronous(*receive()*) and the asynchronous(*onMessage()*) cases. This change however is much larger than can reasonably be scoped in to 0.6.

So given that the desired solution is to large and we have a solution that works for the synchronous 0-10 case the simplest approach is to address the race condition and then move the logic up to be shared between 0-8 and 0-10.

- How the 0-10 synchronous approach addresses rollback.
- Nature of deadlock
- Steps to address issue.
- Other usages of *syncDispatchQueue()* to investigate
- Changes to 0-8 code path to use the 0-10 rollback functionality.
- Potential improvements.

How the 0-10 synchronous approach addresses rollback.

To ensure that all the received messages have been processed, the session is stopped and then a special message is added to the IO/Dispatcher *_queue* that the caller can wait to be processed. This is done in *syncDispatchQueue()*

```
syncDispatchQueue()

void syncDispatchQueue()
{
    final CountDownLatch signal = new CountDownLatch(1);
    _queue.add(new Dispatchable() {
        public void dispatch(AMQSession ssn)
        {
            signal.countDown();
        }
    });
    try
    {
        signal.await();
    }
    catch (InterruptedException e)
    {
        throw new RuntimeException(e);
    }
}
```

This is tested via *RollbackOrderTest.testOrderingAfterRollback()*

Nature of deadlock

The problem is that the test only covers calling the synchronous *receive()*. The addition of a *testOrderingAfterRollbackOnMessage()* test that calls *rollback()* from the Dispatcher Thread highlights the deadlock. When the Dispatcher Thread calls *syncDispatchQueue()* it places an entry on *_queue* then *awaits()* for it to be processed. However, the Dispatcher Thread is the thread that does the processing so suspending it means that the entry will never be processed.

Steps to address issue.

A similar issue has already occurred and been addressed. The *startDispatcherIfNecessary()* method performs a check to see if it is the Dispatcher thread.

AMQSession.startDispatcherIfNecessary()

```
void startDispatcherIfNecessary()
{
    //If we are the dispatcher then we don't need to check we are started
    if (Thread.currentThread() == _dispatcherThread)
    {
        return;
    }
    ...
}
```

The change to the `syncDispatcherQueue()` is to only do the `await()` if the current thread is not the Dispatcher. If the thread is the Dispatcher thread then we should proceed to process the contents of `_queue`.

As long as the rollback mark is correctly set before we call `syncDispatcherQueue()` then the messages in the `_queue` will be correctly rejected/released. After that the rollback can proceed as before by calling `dispatcher.rollback()`. There is no danger of having a single message stuck on the Dispatcher thread as we just ensured that queue was fully processed.

The processing of the `_queue` may vary from the normal run as we cannot allow any message to be dispatched to the consumer. So whilst the `_rollbackMark` should cover all messages in the `_queue` we should also reject/release any other message in the queue rather than dispatch it to a consumer. It is not expected that we would have any other messages in the queue however, we should ensure that this error case is covered.

Other usages of `syncDispatcherQueue()` to investigate

Searching for usages of `syncDispatcherQueue()` yields three hits:

1. AMQSession failoverPrep()
2. AMQSession_0_10.releaseForRollback()
3. BasicMessageConsumer_0_10.getMessageFromQueue(long)

They are all currently in the 0-10 codebase (as expected) there is nothing in the changes that will affect the other two usages; `failoverPrep()` and `getMessageFromQueue(long)`. Only `releaseForRollback()` has the possibility of being called from `onMessage()` and hence being subject to the new logic. `getMessageFromQueue(long)` is part of the JMS `receive()` calls and `failoverPrep()` is called from the IO layer when failover needs to occur.

Changes to 0-8 code path to use the 0-10 rollback functionality.

Currently `releaseForRollback()` is called from the `AMQSession.rollback()` method:

AMQSession.rollback()

```
public void rollback() throws JMSException
{
    ...
    releaseForRollback();
    ...
}
```

In the 0-10 code path the following sync and rollback of the `dispatcher` is done before it performs message release.

0-10.releaseForRollback()

```
public void releaseForRollback()
{
    startDispatcherIfNecessary();
    syncDispatcherQueue();
    _dispatcher.rollback();
    ...
}
```

If these step are brought up to `AMQSession.rollback()` then the 0-8 and 0-10 components need only perform their block rejects/release, and the 0-8 path need not call `AMQSession.rollback()`.

Potential improvements.

Part of the goal of doing this, less than elegant, solution is to share as much code between 0-8 and 0-10 as possible. While this will align the process of performing rollback the method of recording the deliveryTags of delivered messages is still duplicated between the two protocols. 0-10 uses `_txRangeSet` and 0-8 uses `_deliveredMessageTags`. Attempting to consolidate these may not be needed but they both perform the same function so would be beneficial to the client to investigate if the RangeSet could be used in the 0-8 code path.

Qpid extensions to AMQP

Overview

This page is an attempt to collect in a single place all the extensions that have been made (through the use of arguments/options/tec.) to AMQP across the Qpid Java and C++ Brokers.

Ultimately the aim is to try to get both brokers implementing as much common functionality as possible through common extensions - and to advertise which extensions are available in a common way, so that clients can take advantage of functions that are present (or work around functions that are not).

Connection

Connection.Start

Options are carried in the server-properties field

| Name | C++ | Java | Description |
|---------------------|-----|------|-------------|
| qpid.federation_tag | Y | Y | |

Connection.Start-Ok

Options are carried in the client-properties field

| Name | C++ | Java | Description |
|---------------------|-----|------|---|
| qpid.client_pid | Y | N | Allows the process id of a client to be reported by mgmt tools |
| qpid.client_ppid | Y | N | Allows the parent process id of a client to be reported by mgmt tools |
| qpid.client_process | Y | N | Allows the process name of a client to be reported by mgmt tools |

Session

???

Exchange

Exchange.Declare

| Name | C++ | Java | Description |
|-------------------|-----|------|---|
| qpid.ive | Y | N | Specifies 'initial value exchange' behaviour is desired |
| qpid.msg_sequence | Y | N | Requests that the exchange sequences all messages routed through it and adds the sequence number to the message headers |

Binding

Exchange.Bind

Options are carried in the arguments field

| Name | C++ | Java | Description |
|-----------------------|-----|------|---|
| qpid.fed.origin | Y | Y | |
| x-filter-jms-selector | | Y* | (Java Broker topic exchange only currently) add a JMS Selector to the binding to filter messages against an SQL style query |

Queue

Queue.Declare

Options are carried in the arguments field

| Name | C++ | Java | Description |
|----------|-----|------|---|
| no-local | Y | | Specifies that the queue should discard any messages enqueued by sessions on the same connection as that which declares the queue |

| | | | |
|---------------------------------|---|---|--|
| qpid.policy_type | Y | N | Valid values "reject", "flow_to_disk", "ring", "ring_strict" |
| qpid.max_size | Y | N | Defines the maximum number of messages that a queue can contain before the action dictated by the policy_type is taken. |
| qpid.max_count | Y | N | Defines the maximum size of message data (in bytes) that a queue can contain before the action dictated by the policy_type is taken. |
| qpid.file_count | Y | N | This is really a property of a particular store implementation (sets the number of files to use for the queue's 'journal') |
| qpid.file_size | Y | N | This is really a property of a particular store implementation (sets the size of the files to use for the queue's 'journal') |
| qpid.last_value_queue | Y | N | Enables last value queue behaviour |
| qpid.last_value_queue_no_browse | Y | N | Enables special mode for last value queue behaviour (see QPID-2104) |
| qpid.msg_sequence | Y | N | Causes a sequence number to be added to headers of enqueued messages |
| qpid.queue_event_generation | Y | N | Causes an event to be generated for enqueues and dequeues, currently used for asynchronous state replication |
| qpid.trace.id | Y | Y | Adds the given trace id as to the application header "x-qpid.trace" in messages sent from the queue |
| qpid.trace.excludes | Y | Y | Does not send on messages which include one of the given (comma separated) trace ids |
| x-qpid-priorities | | Y | Defines the number of distinct priority levels supported by the queue |
| x-qpid-maximum-message-age | | Y | Specifies that if the oldest message on the queue gets above this age then alerts should be sent |
| x-qpid-maximum-message-size | | Y | Specifies that if the queue gets above this size (in bytes) an alert should be sent |
| x-qpid-maximum-message-count | | Y | Specified that if the queue gets above this size (in message count) an alert should be sent |
| x-qpid-minimum-alert-repeat-gap | | Y | Specified the minimum time gap between consecutive alerts |
| x-qpid-capacity | | Y | Defines the size of the queue in bytes at which flow control on producers will be brought into affect |
| x-qpid-flow-resume-capacity | | Y | Defines the size on bytes of the queue when flow control will be rescinded |

Subscription

Message.Subscribe (Basic.Consume in 0-8/0-9)

| Name | C++ | Java | Description |
|-----------------------|-----|------|---|
| x-filter-jms-selector | | Y | add a JMS Selector to the subscription to filter messages against an SQL style query |
| x-filter-no-consume | | Y | (0-8/0-9 only) Implements browsing for 0-8/0-9 - messages sent on the subscription are not acquired |
| x-filter-auto-close | | Y | (0-8/0-9 only) The server closes the subscription when the queue becomes empty |

Qpid Java Broker - Guidance for 64Bit VM

User Guidance for large heaps using 64Bit VM

Background

The Qpid Java Broker's performance and scalability is bound by the availability of heap to hold in flight data and message references particularly for transient only brokers (i.e. those using the MemoryMessageStore). Historically, we have been limited to ~3GB of useable heap memory.

Using a 64Bit VM opens this up, with increased addressable memory space. We performed testing on a variety of heap sizes, to provide user guidance on utilising a 64Bit VM.

Testing/Results

Heap Size & Broker Performance

The testing looked at two areas performance and cpu usage whilst increasing the heap up to 18GB.

Testing has shown that a 6GB heap performs very similarly to a 3GB Heap on the 32Bit VM so if your storage requirements are just above the 3GB limit then there should be no impact to moving to a 6GB 64Bit Heap. Performance/throughput is largely similar.

Heap Size & Max Message

The maximal performance testing was performed with heaps ranging up to 18GB. This allows a 6 fold increase in the amount of transient messages that the broker can hold.

Testing with 32KB messages showed that over 180,000 messages can safely be stored using an 18GB heap with in-memory storage (i.e. transient only broker). Depending on your usage pattern and volumes the impact of using a large heap is an approximate drop in performance of 2.5% for every 1GB over 6GB for point to point (single publisher single consumer) and 3.3% for publish & subscribe (single publisher to 5 consumers).

For a broker backed with a BDB persistent store (BDBMessageStore) then the number of messages that can be stored in the broker is bound instead by available disk. We tested up to 500,000 messages in a 256MB heap successfully retained.

Heap Size & CPU Utilisation

During both point to point and publish & subscribe testing the CPU usage scaled linearly with the additional data stored in the broker.

Testing has shown that for every extra 6GB approximately one additional core is required for GC processing, housekeeping tasks, etc. So if your current testing shows you need 3 cores then to use a 12GB heap this requirement would increase to 4 cores.

Additional Test Info

Testing has been performed on a 64Bit VM with a variety of Xmx values.

Testing was performed on a 8-way server with 32GB of RAM using JDK 1.6.0_18(b07).

Download

Production Releases

These releases are well tested and appropriate for production use. 0.6 is the latest release of Qpid.

Qpid supports the latest version of AMQP 0-10, and some components also the AMQP 0-8 and 0-9, earlier versions. The Java Broker and Client provide protocol negotiation. [Other versions can be found here](#)

For details on cross component compatibility among releases, see: [AMQP Release Compatibility for Qpid](#)

If you have any questions about these releases, please mail the user list [user list](#)

Latest Release

Multiple Component Packages

| Component | Download | AMQP 0-10 | AMQP 0-8/0-9 |
|-----------------------------|---|-----------|--------------|
| Full release & keys | http://www.apache.org/dist/qpid/0.6/ | Y | Y |
| C++ broker & client | http://www.apache.org/dist/qpid/0.6/qpid-cpp-0.6.tar.gz | Y | |
| Java broker, client & tools | http://www.apache.org/dist/qpid/0.6/qpid-java-0.6.tar.gz | Y | Y |

Single Component Package

Broker

| Language | Download | AMQP 0-10 | AMQP 0-8/0-9 |
|----------|---|-----------|--------------|
| Java | http://www.apache.org/dist/qpid/0.6/qpid-java-broker-0.6.tar.gz | Y | Y |

Client

| Language | Download | AMQP 0-10 | AMQP 0-8/0-9 |
|--|---|-----------|--------------|
| C# (.NET, WCF) WCF channel (C++ Broker Compatible) | http://www.apache.org/dist/qpid/0.6/qpid-wcf-0.6.zip | Y | |

| | | | |
|---|---|---|---|
| C# (.NET, WCF, Excel) 0-10 client (C++ Broker Compatible) | http://www.apache.org/dist/qpid/0.6/qpid-dotnet-0-10-0.6.zip | Y | |
| C# (.NET) 0-8 client (Java Broker Compatible) | http://www.apache.org/dist/qpid/0.6/qpid-dotnet-0-8-0.6.zip | | Y |
| Java | http://www.apache.org/dist/qpid/0.6/qpid-java-client-0.6.tar.gz | Y | Y |
| Python | http://www.apache.org/dist/qpid/0.6/qpid-python-0.6.tar.gz | Y | Y |
| Ruby | http://www.apache.org/dist/qpid/0.6/qpid-ruby-0.6.tar.gz | Y | Y |

Management tools

C++ broker management

| Component | Download | AMQP 0-10 |
|---------------------------------|---|-----------|
| cmd line (packaged with python) | http://www.apache.org/dist/qpid/0.6/qpid-python-0.6.tar.gz | Y |
| QMan JMX bridge, WS-DM | http://www.apache.org/dist/qpid/0.6/qpid-management-client-0.6.tar.gz | Y |

Java broker management

| Component | Download |
|------------------------|--|
| JMX Management Console | Linux x86 Linux x86_64 Mac OS X Solaris 10 Sparc Windows x86 |

Windows Installer

The Windows installer is available from <http://www.riverace.com/qpid/downloads.htm>. It is built from the 0.6 C++ broker & client and C# WCF Channel source distributions listed above. It has been tested for Windows XP SP3 and above.

Previous Release

Multiple Component Packages

| Component | Download | AMQP 0-10 | AMQP 0-8/0-9 |
|-----------------------------|---|-----------|--------------|
| Full release & keys | http://www.apache.org/dist/qpid/0.5/ | Y | Y |
| C++ broker & client | http://www.apache.org/dist/qpid/0.5/qpid-cpp-0.5.tar.gz | Y | |
| Java broker, client & tools | http://www.apache.org/dist/qpid/0.5/qpid-java-0.5.tar.gz | client | Y |

Single Component Package

Broker

| Language | Download | AMQP 0-10 | AMQP 0-8/0-9 |
|----------|---|-----------|--------------|
| Java | http://www.apache.org/dist/qpid/0.5/qpid-java-broker-0.5.tar.gz | | Y |

Client

| Language | Download | AMQP 0-10 | AMQP 0-8/0-9 |
|---|---|-----------|--------------|
| C# (.NET, WCF, Excel) 0-10 client (C++ Broker Compatible) | http://www.apache.org/dist/qpid/0.5/qpid-dotnet-0-10-0.5.zip | Y | |
| C# (.NET) 0-8 client (Java Broker Compatible) | http://www.apache.org/dist/qpid/0.5/qpid-dotnet-0-8-0.5.zip | | Y |
| Java | http://www.apache.org/dist/qpid/0.5/qpid-java-client-0.5.tar.gz | Y | Y |
| Python | http://www.apache.org/dist/qpid/0.5/qpid-python-0.5.tar.gz | Y | Y |
| Ruby | http://www.apache.org/dist/qpid/0.5/qpid-ruby-0.5.tar.gz | Y | Y |

Management tools

C++ broker management

| Component | Download | AMQP 0-10 |
|-----------|----------|-----------|
| | | |

| | | |
|---------------------------------|---|---|
| cmd line (packaged with python) | http://www.apache.org/dist/qpid/0.5/qpid-python-0.5.tar.gz | Y |
| QMan JMX bridge, WS-DM | http://www.apache.org/dist/qpid/0.5/qpid-management-client-0.5.tar.gz | Y |

Java broker management

| Component | Download |
|------------------------|---|
| JMX Management Console | Linux x86 Linux x86_64 Mac OS X Windows x86 |
| Command line interface | http://www.apache.org/dist/qpid/0.5/qpid-management-tools-qpid-cli-0.5.tar.gz |

QpidComponents.org

QpidComponents.org provides further components for Apache Qpid, including both persistence and management tools. These components are open source, but are not developed as part of the Apache Qpid project due to licensing or other restrictions.

Contributed C++ Packages

Pre-built Linux Packages

Fedora

On Fedora, Qpid can be installed using yum. Because Java RPMs are not yet available in Fedora repos, the Java client is not in these distributions.

To install the server:

```
# yum install qpidd
```

To install C++ and Python clients:

```
# yum install qpidc-devel
```

```
# yum install amqp python-qpid
```

To install documentation:

```
# yum install rhm-docs
```

To install persistence using an external store module:

```
# yum install rhm
```

Windows Installer

The Windows installer is available from <http://www.apache.org/dist/qpid/0.5-windows/qpidc-0.5.msi>. It is built from the 0.5 C++ broker and client source distribution listed above. It has been tested for Windows XP SP2 and above.

The Windows executables require the Visual C++ 2008 SP1 run-time components. If the Visual C++ 2008 SP1 runtime is not available, the Qpid broker will not execute. If you intend to run the broker and Visual C++ 2008 is not installed, you must install the Visual C++ 2008 SP1 Redistributable. Please see <http://www.microsoft.com/downloads/details.aspx?familyid=A5C84275-3B97-4AB7-A40D-3802B2AF5FC2&displaylang=en> for download and installation instructions.

If you intend to develop Qpid client applications using this kit, you should install [Boost version 1.35](#) (please be sure to select VC9 support when installing) in addition to Visual Studio 2008 SP1.

Source Code Repository

The latest version of the code is always available in the [Source Repository](#).

The AMQP Distributed Transaction Classes (Java)

The distributed transaction classes provide support for the X-Open XA architecture. The `dtx-demarcation` class is used to demarcate transaction boundaries on a given channel that is subsequently used to perform AMQP native transactional work (produce/publish messages). Transaction coordination and recovery operations are provided by the `dtx-coordination` class.

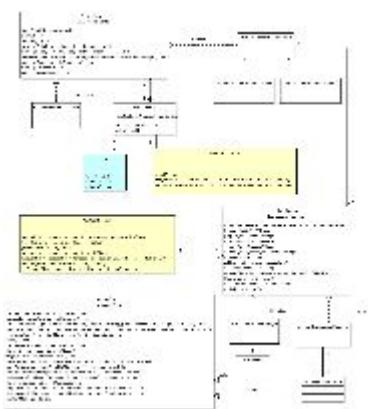
A Transaction Manager uses RM Client XA interface to demarcate transaction boundaries and coordinate transaction outcomes. RM Clients use the dtx-demarcation class to associate transactional work with a transactional channel. The transactional channel is exposed to the application driving the transaction. The application can then use the transactional channel to transactionally produce and consume messages. RM clients use dtx-coordination to propagate transaction outcomes and recovery operations to the AMQP broker. A second coordination channel can be used for that purpose.

More details about can be found at:

- https://wiki.108.redhat.com/wiki/index.php/AMQP:Transaction_SIG_dtx_XML
 - <http://cwiki.apache.org/confluence/download/attachments/55787/dtx-classes-specification-document-v1.2.pdf>
 - <http://cwiki.apache.org/confluence/download/attachments/55787/dtx-classes-presentation-v0.10-PMC-03142007.pdf>

The Qpid Implementation in Java

As shown on the following class diagram, there are two protocol specific dtx classes, that is to say DtxDemarcation and DtxCoordination that are highlighted in yellow.



DtxDemarcation

DtxDemarcation interacts with the corresponding AMQChannel. The operation select creates the corresponding TransactionalContext (a channel has by default a non-transactional context). The operations start and end associate and disassociate a provided xid with the current TransactionalContext that percolates the call to the TransactionManager (operations begin and end respectively). Note that the operation end is responsible for acknowledging the messages against the context i.e. those messages are seen as being consumed under the currently associated xid.

DtxCoordination

DtxCoordination directly interacts with the TransactionManager. Note that it is a requirement that the operation end is called on all involved channels (i.e. all the acknowledged messages have been specifically consumed under the provided xid).

TransactionalContext

There are three flavours of `TransactionalContext`: the non transactional one, the local and distributed ones. The distributed and local contexts are very similar and both extend the abstract context. Note that the distributed context does not implement commit and rollback as this is `DtxCoordination` that is responsible for deciding of a transaction outcome.

TransactionManager and MessageStore

Transaction manager and message store are linked as the message store may need to add transaction records to the transaction identified by a given `xid`. Moreover, the transaction manager may need to use the transactional facilities of the underlying store. This is the case of the `JDBCStore` and `JDBCTransactionManager`. The `JDBCTransactionManager` uses the transaction facilities of the `JDBCStore` for performing ACID operations during `prepare` and `commit`.

This is the responsibility of the MessageStore to recover queues and exchanges and messages. Note that the JDBCTransactionManager delegates the responsibility of getting the list of in-doubt transactions to the JDBCStore but another implementation of TransactionManager may handle that directly.

The MessageStore implementation should not load the messages in memory during recovery but only set the messageID. The message header, publish info and payload are lazily loaded. Note that the message payloads are currently loaded in memory. We can however easily implement a direct streaming of message payload on the wire (The MessageStore interface can be extended for supporting that).

AMQP compatibility

Qpid provides the most complete and compatible implementation of AMQP. And is the most aggressive in implementing the latest

version of the specification. Qpid can be downloaded here

There are two brokers:

C++ with support for AMQP 0-10

Java with support for AMQP 0-8, 0-9, and 0-10.

There are client libraries for C++, Java (JMS), .Net (written in C#), python and ruby.

- All clients support 0-10 and interoperate with both brokers as of 0.6.
- The JMS client supports 0-8, 0-9 and 0-10 and interoperates with both brokers.
- The python and ruby clients will also support all versions, but the API is dynamically driven by the specification used and so differs between versions. To work with the C++ broker you must use 0-10. To work with the Java broker you can use any version as of 0.6, or prior to 0.6 you can use 0-8 or 0-9.
- There are two separate C# clients, one for 0-8 that interoperates only with the Java broker, and one for 0-10.
- There is also a WCF channel, which wraps the 0-10 native C++ client library.

QMF Management is supported in Ruby, Python, C++, and can be translated to Java JMX & WS-DM via the QMan management tool.

AMQP Compatibility of Qpid releases:

Qpid implements the AMQP Specification, and as the specification has progressed Qpid is keeping up with the updates. This means that different Qpid versions support different versions of AMQP. Here is a simple guide on what use.

Here is a matrix that describes the different versions supported by each release

Y = supported

N = unsupported

IP = in progress

P = planned

| Component | Spec | | | | | |
|--------------------------|------|------|----|----|-----|-----|
| | | M2.1 | M3 | M4 | 0.5 | 0.6 |
| Java client | 0-10 | | Y | Y | Y | Y |
| | 0-9 | Y | Y | Y | Y | Y |
| | 0-8 | Y | Y | Y | Y | Y |
| Java broker | 0-10 | | | | | Y |
| | 0-9 | Y | Y | Y | Y | Y |
| | 0-8 | Y | Y | Y | Y | Y |
| C++ client/broker | 0-10 | | Y | Y | Y | Y |
| | 0-9 | Y | | | | |
| Python client | 0-10 | | Y | Y | Y | Y |
| | 0-9 | Y | Y | Y | Y | Y |
| | 0-8 | Y | Y | Y | Y | Y |
| Ruby client | 0-10 | | | Y | Y | Y |
| | 0-8 | Y | Y | Y | Y | Y |
| C# client | 0-10 | | | Y | Y | Y |
| | 0-8 | Y | Y | Y | Y | Y |
| WCF channel | 0-10 | | | | | Y |

Interop table by AMQP specification version

Above table represented in another format.

| | release | 0-8 | 0-9 | 0-10 |
|-------------|---------------|-----|-----|------|
| Java client | M3 M4 0.5 0.6 | Y | Y | Y |
| Java client | M2.1 | Y | Y | N |

| | | | | |
|-------------------|---------------|---|---|---|
| Java broker | 0.6 | Y | Y | Y |
| Java broker | M3 M4 0.5 | Y | Y | N |
| Java broker | M2.1 | Y | Y | N |
| C++ client/broker | M3 M4 0.5 0.6 | N | N | Y |
| C++ client/broker | M2.1 | N | Y | N |
| Python client | M3 M4 0.5 0.6 | Y | Y | Y |
| Python client | M2.1 | Y | Y | N |
| Ruby client | M4, 0.5, 0.6 | Y | Y | Y |
| Ruby client | M3 | Y | Y | N |
| C# client 0-10 | M4 0.5 0.6 | N | N | Y |
| C# client 0-8 | M3 M4 0.5 0.6 | Y | N | N |
| WCF channel | 0.6 | N | N | Y |

Queue Replay

Background

A lengthy discussion on replay in [January 2007](#)(page #7, Thread) highlighted a number of requirements and possible implementation options for adding replay to Qpid and AMQP. The requirements come from the desire to speed up the rate a consumer can read messages and to simplify its recovery when it starts. This page is to give some background, a proposal and finally some implementation options for discussion.

History

When a an application updates the state of a single **Resource Manager**, e.g a database or queue manager, it normally does so within the context of a local **Transaction** and this transaction exhibits the following ACID properties:

- **Atomicity** The result of the transaction are either all committed or all rolled back.
- **Consistency** The completed transaction transformed the resource from one known state to another. Inserting a row into a database or removing a message from a queue are common examples.
- **Isolation** Changes the the resources state effected by the transaction does not become visible ouside of the transaction until the transaction commits.
- **Durability** The changes that result from the transactions commitment survive subsequent system or media faulures.

Distributed Transaction

A distributed transaction is typically implemented by performing a **Two Phase Commit** (2PC) over which there are several varients the most well know being the X/Open XA specification. Where both the middleware and the consumer support XA, a separate **Transaction Manager** is used to coordinate the local transactions. The transaction manager coordinates atomicity at the global level whilst each resource manager is responsible for the ACID properties of its local transactions.

These benefits do not come without cost.

- Increased transaction processing latency, typically due to the additional forced disk writes.
- Applications can become blocked pending the resolution of an in-doubt global transaction.
- Reduced concurrency
- Multi-system deadlock
- Administration complexity
- Backing up the transaction manager involves co-ordinating all transaction logs at the same time or processing must be suspended.

Idempotence and Replaying

When a message is being moved from system A to system B (e.g. from WebSphereMQ to ORACLE), distributed transactions can be avoided if;

- System B can handle duplicates (or can detect them and deal with them accordingly) - i.e. it is idempotent.
- System A can replay messages from a known stable point in history.

The most common way of doing this is simply managing the local transactions so that system B commits before system A. The start of replay is them the end of the last transaction on system A.

Typically a MOM will immediately delete a message once it has been committed by all of its consumers.

Commit is Not The End

If messages are made available for replay to a consumer after it has been committed, we can stretch the point in time the consumer recovers from back to any point.

- A few minutes ago
- Trade ID FFS987654321
- Start of day
- End of yesterday
- Friday.

This larger recovery window lets downstream consumers the flexibility to recover from more failure scenarios.

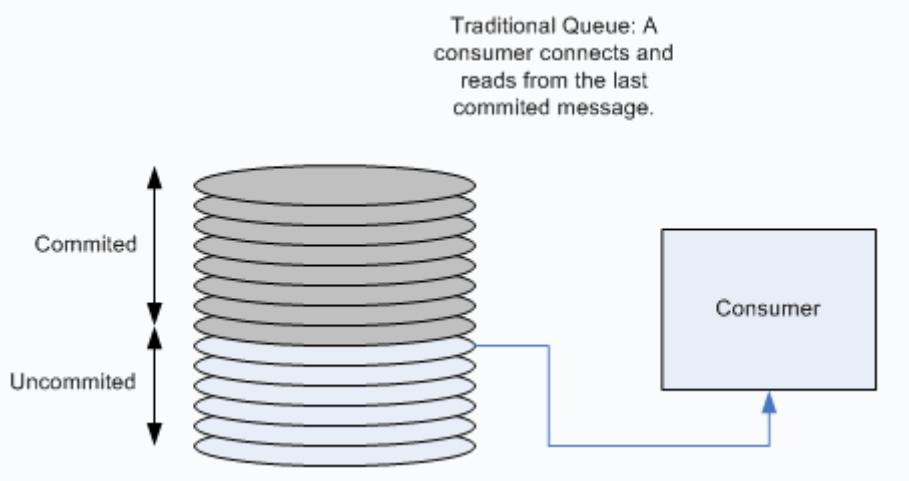
- Retry an end of day batch job.
- Replay due to reference data problem in target system.
- Replay due to database or application failure.

Replay as a First Class Service

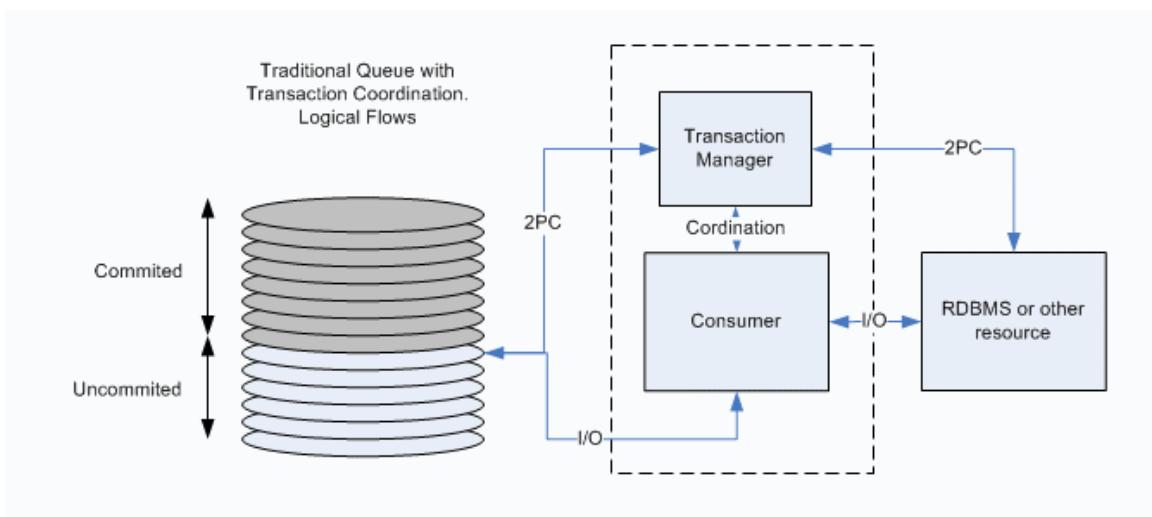
When a traditional queue is opened for reading, it is opened and the next message is the oldest one that has not been destructively read (i.e. read and committed).

In isolation, a consumer manages its own local transactions with the message broker to confirm when a message or group of messages is processed, stored and **stable**. The local transaction leads to a **disk write** in the queue storage to mark the messages as read.

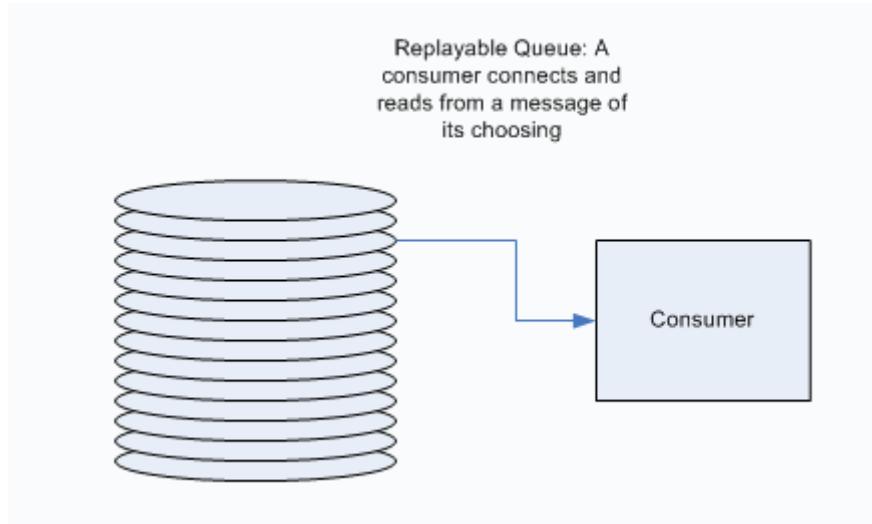
In this XA free world the consumer relies on the messaging to replay messages from the applications last good known state. As it's always reading from a queue, the only extension to the queues semantics is to let it be opened for reading from a known message, irrespective of whether the message has been committed or not and it goes without saying that they should be in the same order in which they were originally delivered.



Many consumers of a guaranteed message flow are writing to a database and this database is the consumers view of its state, it's certainly where the consumer recovers from when it starts up. The traditional model of using a transaction manager, typically XA, to co-ordinate the local transactions on the database and messaging broker is slow and not without its problems.



Another model is to have the messaging infrastructure support replay of messages from a known point in history i.e. to correlate the current state of the consumers database with a **last received** message that last caused an update to the database from this channel. This is not an all encompassing pattern but rather complements other ways to synchronize state between a message broker and a database.



Requirements

- **Replay messages** from a queue from a given message identified by a message ID or a header property.
- **Administrative support** to purge messages from a queue as part of a business process such as **End of Day**
- **Zero impact** on other queues and their consumers.

Proposal: A Replayable Queue

Queues are the storage agents in AMQP so are the logical point to provide replay. A Replayable Queue (**RQ**) is not the default queue behavior but rather has to be **explicitly configured**. In many ways an RQ is somewhere between a traditional queue and a transaction log such as **HOWL**.

An RQ has the following properties:

- An RQ can only have a **single consumer**. Multiple consumers complicate the problem so I propose discounting them for now.
- Messages are **not deleted** when consumed by a regular consumer. The act of acknowledging the message is just another property on the message. Indeed, the consumer may never acknowledge the message as this implies a write on the message broker to update the message's state.
- When an RQ is opened for reading, the consumer must give a selector that will **identify a point in the queue** to begin message delivery from.
- Administratively **defined points** in the queue exist. These points can be defined by an administration API and associated tooling and used as points to replay from.
- Queues are **purged of messages** by the administration API or associated tooling. This allows external processes such as End of Day to initiate message archiving or deletion when it is safe to do so.
- An RQ can be replicated. Implementation options? SAN replication, dual writes?

Benefits

- An RQ, by virtue of a single consumer, does not need to be written to when a consumer reads messages as it is the responsibility of the consumer to provide the synchronization point when it first connects. This can significantly speed up the consumer as its bottleneck will be its own database write.
- A complete record of all messaging activity is available.

Downsides

- The size of the store needs careful management so any implementation details do not cause performance issues.

Implementation Options in Qpid.

Configuration

Storage

Management

Usage from JMS

Getting Involved

There are many ways you can get involved in Qpid:

1. Use Qpid and post us your feedback to dev@qpid.apache.org
2. Participate on the [mailing lists](#).

Remeber All patches must be attached to a JIRA with the appropriate ASL assigned to it or we can't use it!

3. Contibute via JIRA [JIRA issues](#). We use the 'Starter' component in JIRA to indicate tasks we think are good entry points.
4. Review the current code
5. Help write user documentation or wiki documentation
6. Help write user examples.
7. Most definitely add more tests.

Please be sure to take a look at the coding guidelines for the section of the project that you contribute to

- [Java Coding Standards](#)
- [C++ Coding Standards](#)
- [C++ Tips](#)
- OS version considerations

Some Ideas to contribute

Themes and JIRA's for our road map are located here [roadmap](#) Please fell free to mail the dev list if you want to pick up any of these items on the project.

[looking to pitch in](#)

Project Etiquette

Please read and digest our [Qpid Project Etiquette Guide](#). This is a key guide for new contributers and required reading.

Becoming a committer:

Qpid uses the following guidelines for voting in new committers. First off we would like new committers to have provided meaningful contribution to the project. By contributions we include development (tests, features, examples) or documentation through patches and interactions with the project through lists and JIRA. It should be noted that as we send our JIRA to the dev list, thus some people filter the JIRA's to limit traffic. It is thus good to cross post something to the dev list every now and again, if the discussion is being held primarily on JIRA.

The key question is what does the Qpid project consider to be a meaningful contribution to be come a committer. As this bar is set for all new committers Qpid will be conservative in general with adding new committers.

The Qpid project will look to see if someone consistently provides quality contributions and interactions with the project over a period of 1 to 2 months. Based on that the PMC will vote the new committers onto the project.

If you have any question please mail dev@qpid.apache.org or private@qpid.apache.org

Many thanks for you interest,
the Qpid team.

Joining the PMC

Nominations for Qpid PMC membership will be voted on by the Qpid PMC.

GSoC

Potential GSoC Projects

qpid-java-qmf

Candidates:

Project Goal: Add QMF support to the Java Broker

Mentors:

Project Description:

qpid-java-message-store-tool

Candidates:

Project Goal: Improve message store tool

Mentors:

qpid-java-monitoring-alerting

Candidates:

Project Goal: Improve the Java servers reporting, accounting and alerting

Mentors:

Project Description:

qpid-scheduling

Candidates:

Project Goal: provide a mechanism to allow better scheduling of tasks ?

Mentors:

Project Description:

qpid-java-bridging

Candidates:

Project Goal: Facilitate moving messages between JMS providers, such as Qpid -> ActiveMQ

Mentors:

Project Description:

qpid-java-xml-exchange

Candidates:

Project Goal: Implement an XQuery capable exchange, equivalent to and compatible with C++ servers

Mentors:

Project Description:

qpid-stonehenge-integration

Candidates:

Project Goal: Investigate integration with Apache Stonehenge project, potentially other Apache projects

Mentors:

Project Description:

OSVC

A RHEL4 Grimoire

by Michael Goulish on this 11th day of April, 2008

Introduction

Programmer! Turn back now, if you can, to the daylit world!

But if you must walk this road - take with you this map! Do not stray into the mires and pits where I have wandered and despaired.

Herein I will describe what I can of the perils I have encountered in the antique land of RHEL4.

Iterators and the ">" operator.

I believe this is a compiler problem with the -> operator, in the neighborhood of any kind of iterators.

Code like this will not compile:

```
ConsumerImplMap::iterator i = consumers.find(delivery.getTag());  
  
if (i != consumers.end())  
{   get_pointer(i)->acknowledged(delivery); // <--- Bad! }
```

Do this instead:

```
ConsumerImplMap::iterator i = consumers.find(delivery.getTag());  
  
if (i != consumers.end())  
{   (*i).second->complete(delivery); // <--- Good! }
```

(Thanks, Kim!)

Don't use BOOST_FIXTURE_TEST_CASE

Because it Doesn't Exist.

All it does is allow you to use a class (or struct) declaration in many test cases without declaring it in every one.

So what? Big deal! Just declare your structure in each test case, and use the QPID_AUTO_TEST_CASE macro instead!

If you have this struct:

```
struct ClientSessionFixture : public Foo  
{ int bar; }
```

Don't do this:

```
BOOST_FIXTURE_TEST_CASE(testQueueQuery, ClientSessionFixture)  
{ bar = 666; BOOST_CHECK_EQUAL ( bar, 666 ); }
```

Do do this:

```
QPID_AUTO_TEST_CASE(testQueueQuery)  
{ ClientSessionFixture fix; fix.bar = 666; BOOST_CHECK_EQUAL ( fix.bar, 666 ); }
```

(Thanks, Alan!)

Don't use the BOOST_TEST macros !

If you are tempted to use

```
BOOST_AUTO_TEST_SUITE, or  
BOOST_AUTO_TEST_CASE, or  
BOOST_AUTO_TEST_SUITE_END,
```

don't!

Use instead:

```
QPID_AUTO_TEST_SUITE, or  
QPID_AUTO_TEST_CASE, or  
QPID_AUTO_TEST_SUITE_END !
```

They turn into Appropriate Things depending on the version of Boost you are using.

Sometimes the Appropriate Thing is whitespace...

(Thanks, Alan and Kim !)

Don't use boost::iostreams.

They don't exist.

```
/usr/include/boost/iostreams/: No such file or directory
```

Instead, use low-level Unix IO, from the Dawn of Time.

```
open()  
read()  
write()
```

Qpid Project Etiquette Guide

Purpose

This guide, written by Rafael Schloming, gives both Qpid committers and submitters a useful introduction to project etiquette, shedding light on how we do things & why. Following this etiquette makes the path to righteousness less long and winding !

Maintainers

The Qpid project consists of a number of major components spread across almost as many different languages. Thus it is rare for qpid committers to be experts in every single area of the project.

As such it is expected that qpid committers make some effort to reach out to their teammates before directly modifying components that are outside their chosen areas.

You should use the dev list to reach out to Qpid developers and comment on any JIRAs you're progressing.

Patch Submission

As a committer it can be difficult to decide whether/how to provide feedback when someone submits a patch. Often it is tempting to just fix up the patch and avoid the slower and sometimes awkward process of telling someone that they got some part of it wrong.

However, it is necessary to ensure that those who submit patches get to learn what they need to know in order to become a valued qpid committer.

In that spirit, here are a few guidelines for contributing patch submissions and how we handle them:

- Submitters should produce the final patch(s) as applied to the tree. Producing a patch that needs little or no rework is a key skill for a qpid committer.
- Maintainers may make requests for 'trivial' updates to the patch. Such requests are vital to ensuring that contributors get familiar with subtle yet important aspects of the code, stylistic conventions, etc.
- Make sure the submitter is familiar with project etiquette so they understand why we make seemingly trivial requests. We'll ask new contributors to read this etiquette info for that reason !
- A one-time patch from someone passing through may need nothing more than a polite thank you regardless of the content. If a submitter does aim for committership, best to make it plain you're planning to stay around on the project.
- Break up unrelated changes. It wouldn't be considered correct for a committer to glom together too many unrelated changes within a single commit, and so we won't commit this kind of patch from submitters.
- You need a JIRA for any patch to be attached to, which should accurately describe the change.

Big Ideas

Every so often someone has a Big Idea that they get excited about and want to go do. They generally mail the list about it to give people the opportunity to comment, and then when nobody says anything they go off and do it.

Fast forward six months later they commit/merge/enable/publish the result of their Big Idea, and suddenly everyone understands the full implications, and not everyone is happy.

Guidelines

So, here are a few guidelines for making sure this doesn't happen, starting with how to write a good proposal for a Big Idea:

- Make sure your proposal is recognizable as a proposal. An easy way to avoid ambiguity is to start a new thread for your proposal and stick "proposal" somewhere in the subject.
- Understand who and what your proposal effects, and make this clear. If you think X's implementation of Y doesn't deserve to live and you're going to rewrite the whole thing from scratch then make this very clear so X can object sooner rather than later.
- Make sure you call out loudly that you intend to kill feature A, even if you think no-one on earth should care.
- Even if you're going to write an Atari emulator, and you're 100% sure that it won't overlap with anything in the rest of the project, make sure you understand how and why it relates to the rest of the project in general, and make that clear.
- Be concrete. Often a proposal of the form "hey, I did a little of this, here is a proof-of-concept, I'd like to do it for real now" is far more effective than a proposal of the form "hey, I have this vague idea about this, I'm going to vaguely suggest it would be good if someone did it".
- Talk about the long term implications of your proposal. If it's code then someone needs to maintain it, and committers will expect this to be you. Make clear your intentions.
- Never assume silence implies complicity, more likely it means people didn't understand the implications of your proposal, or didn't have time to figure out why they didn't like it. Ask again !
- The bigger your idea is, the more time and effort you should spend on the proposal, and ensuring that you get positive responses and deal with the comments and feedback provided in your actual implementation.

Talk early, talk often

No matter how incredibly excellent your proposal is, there is going to need to be some discussion before the result of your Big Idea is

blessed. Here are some things you can do to help that discussion go smoothly:

- Make frequent progress reports. Every hour/day/week/month you spend working without telling people what you're doing is an hour/day/week/month of your time that you risk wasting.
- Define milestones and make them visible to the rest of the project. Just like a concrete proof-of-concept can help a proposal, it helps to give people a concrete look at what you're doing while it's in progress. This can go a long way towards avoiding surprises.

And finally

When the time does come to commit/merge/enable/publish your Big Idea, it really shouldn't be a surprise to anyone if you've followed the steps up until now, but make sure you let people know in advance by making note in your final few progress reports of when you expect to be finished, and sending a note to the dev list a day or so before you flip the big switch.

HermesJMS

This is a draft

Background

HermesJMS is a console for JMS messaging and supports QPID. This page shows how to configure Hermes with QPID.

As the QPID codebase is moving quickly forward at the moment and there are some workarounds in place within Hermes to support it, you should ensure that you download the latest Hermes build from [HEAD](#)

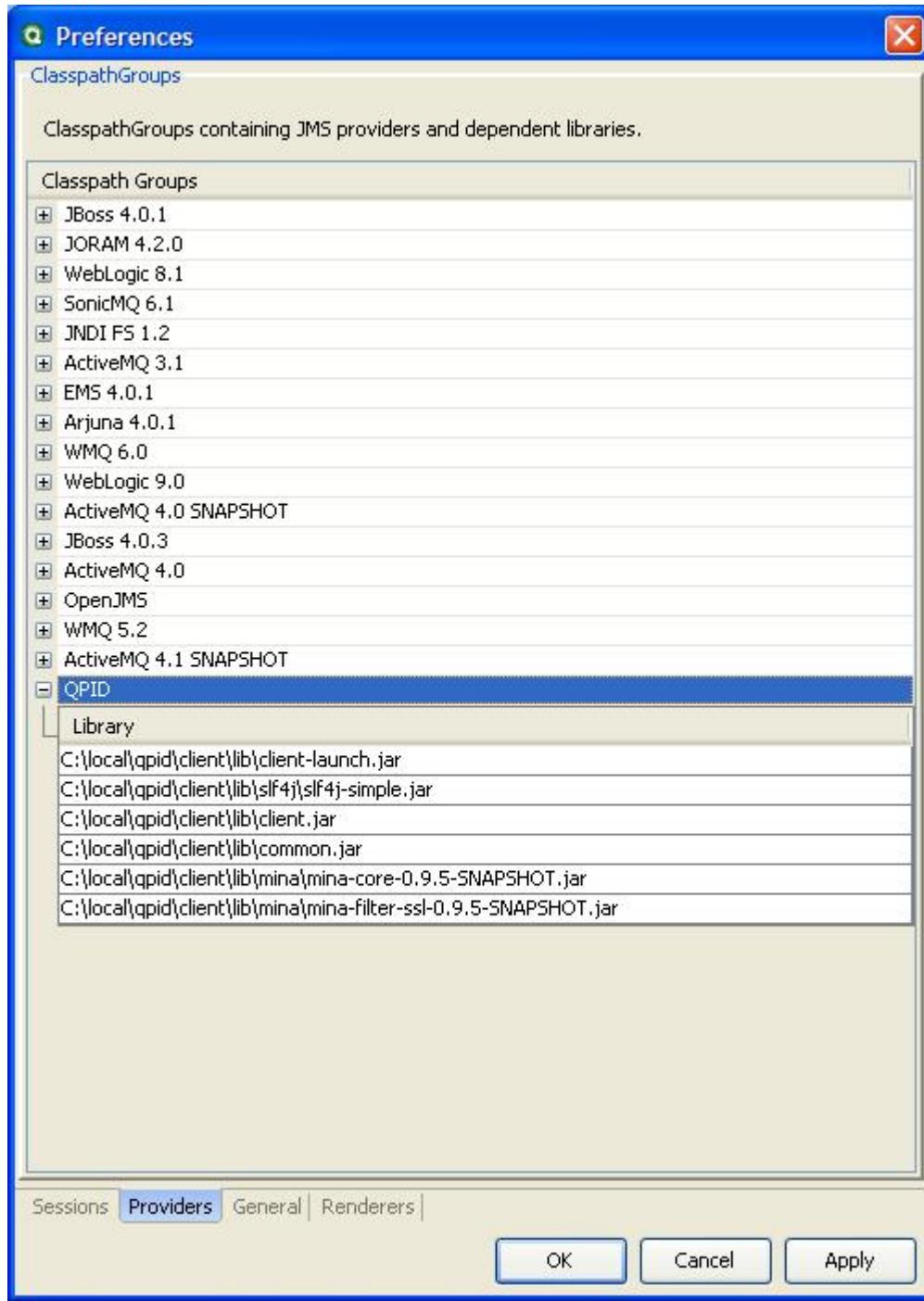
Finally, you should familiarise yourself with Hermes via the tutorials on the website before continuing. Look at the Tibco EMS and JBoss tutorials for how to use connection factories directly or via JNDI. This page uses them directly.

Configuring

To get QPID working, you need to configure the libraries to load via the GUI. Hermes loads up each provider in its own classloader to avoid any dependency problems across providers. This tutorial assumes you've downloaded and built QPID via the `ant dist` task.

Configure the CLASSPATH

Start Hermes and select *Options -> Configuration* and click on the *Providers* tab. Create a new classpath group (right click for this) and add the following JARS to it:



When asked, choose the scan option to get Hermes to search the libraries to find any classes that implement JMS connection factory interfaces, this is essential for the next step.

Finally, click OK.

Configure The Session

Right click on the tree of sessions and select *New -> Session*. In the session combo box at the top put in the name you want, for example QPID. Its important the next thing you do is select the loader so the dialog can find any connection factories.

In the *Class* combo box choose the `org.apache.qpid.client.AMQConnectionFactory`.

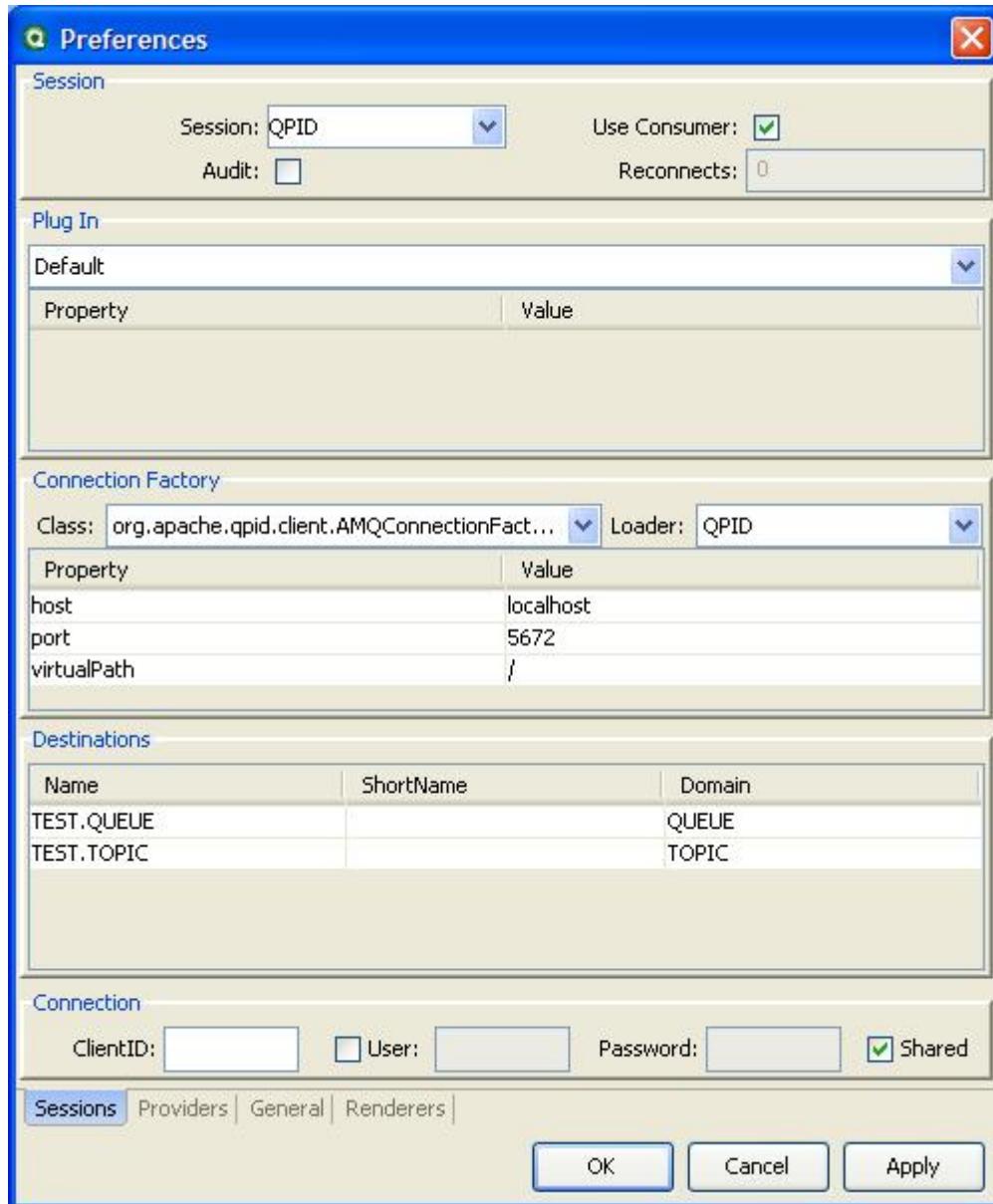
Next, in the property list for the connection factory, right click to add new properties and use the combo to select which one and add their value.

Finally, and very importantly, check the *Use consumer* checkbox at the top. QPID does not support JMS queue browsing but Hermes lets you use a `{MessageConsumer}` instead. Remember of course that if a consumer is currently active you will not get the real queue content, Hermes uses a transacted session and some messages may be uncommitted in another consumers session.

Add Destinations

Currently you must manually add queues and topics. You can add them now or later from the *New queue*, *New topic* or *New durable subscription* actions in the toolbar. If you add them now then right click in the destination list to add them.

The session and destinations should look something like this. Once happy, click OK.



Try a Browse

Finally, double click on a queue or topic in the newly created session in the tree and you should get see something like the following (assuming you've got something in the queue or being published in the topic to see):

Hermes v1.11-SNAPSHOT

File Messages Actions Options Help

Sessions QPID TEST.QUEUE

| # | JMSMessageID | JMSDestination | JMSTimestamp | JMSType | JMSReplyTo | JMSCorrelationID | JMSExpiration |
|----|--------------|----------------|----------------------|-------------------------|------------|------------------|---------------|
| 0 | ID:1 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |
| 1 | ID:2 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |
| 2 | ID:3 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |
| 3 | ID:4 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |
| 4 | ID:5 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |
| 5 | ID:6 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |
| 6 | ID:7 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |
| 7 | ID:8 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |
| 8 | ID:9 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |
| 9 | ID:10 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |
| 10 | ID:11 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |

byte[size=166]

Payload Header toString Hex FIX

Filter:

Finished, 334 messages read.

Tools Tasks Log

Browsing TEST.QUEUE on QPID: Done. 21:49:28 52M of 56M

This message is a JMS BytesMessage so there not much to see until you see its really a FIX message and clicking on the FIX tab reveals it.

Hermes v1.11-SNAPSHOT

File Messages Actions Options Help

Sessions QPID TEST.QUEUE

| # | JMSMessageID | JMSDestination | JMSTimestamp | JMSType | JMSReplyTo | JMSCorrelationID | JMSExpiration |
|----|--------------|----------------|----------------------|-------------------------|------------|------------------|---------------|
| 0 | ID:1 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |
| 1 | ID:2 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |
| 2 | ID:3 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |
| 3 | ID:4 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |
| 4 | ID:5 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |
| 5 | ID:6 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |
| 6 | ID:7 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |
| 7 | ID:8 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |
| 8 | ID:9 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |
| 9 | ID:10 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |
| 10 | ID:11 | Unknown | Sun Jan 25 15:54:... | application/octet-st... | | | 0 |

Field Name Value Description

34 MsgSeqNum 54

35 Mstyp D NewOrderSingle

49 SenderCompID CN

52 SendingTime 20060424-22:02:43.696

56 TargetCompID ISE

11 COrdID 000005-173730

21 HandInst 1 AUTOMATED_EXECUTION_ORDER_PRIVATE_NO_BROKER_I...

38 OrderQty 500

40 OrdType 3 STOP

44 Price 123

54 Side 1 BUY

Payload Header toString Hex FIX

Filter:

Finished, 334 messages read.

Tools Tasks Log

Browsing TEST.QUEUE on QPID: Done. 21:49:52 51M of 56M

Other Features

Refer to the [HermesJMS](#) site for how to use all the other feature of HermesJMS with QPID.

Issues

1. Queues must already exist before you browse them.
1. `get/setJMSDestination()` is not supported on a message so `Unknown` is shown.

Informal M2.1 code review 2008-03-18

rgodfrey, 634720, BasicMessageConsumer.java: add comment to document DUPs_OK and AUTO_ACK closenes
ritchiem, 635549, SelectorParserTest.java: fix and enable test DONE
aidan, 637146, AMQConnection.java: add comment
ritchiem, 637170: jira that only outer failover should rety [QPID-855](#)
ritchiem, 637176: change CancelTest logger to use CancelTest class DONE
ritchiem, 637977: raise Jira to fix client to not communciate on channels it has been told are closed [QPID-865](#)

Navigation

Apache Qpid

[Home](#)
[Download](#)
[Getting Started](#)
[Documentation](#)
[Mailing Lists](#)
[Issue Reporting](#)
[FAQ/How to](#)

Resources

[Getting Involved](#)
[Qpid Integrated with..](#)
[Source Repository](#)
[Building Qpid](#)
[Developer Pages](#)
[QMF](#)

About Qpid

[People](#)
[License](#)
[Project Status](#)
[Acknowledgments](#)

About AMQP

[What is AMQP ?](#)
[AMQP Specification Download](#)



Acknowledgments



We
acknowledge
ej-technologies
for
giving
us
a
free
team
license
for
profiling
Qpid
Java
code.



We
acknowledge
Headway
Software
for
giving
us
free
licenses
of
Structure101
for
analyzing
and
managing
the
architecture
of
Qpid
Java
code.

FAQ

- FAQ

- About AMQP
 - What is AMQP?
 - Where did AMQP come from
 - Why use AMQP?
- Qpid & AMQP
 - Is Qpid AMQP Compliant?
 - What Client support does Qpid have?
 - What messaging topologies are supported by AMQP and Qpid?
 - What AMQP and other exchanges does Qpid support?
- Security
 - What encryption does Qpid support?
 - What authentication does Qpid support?
 - What authorization does Qpid support?
 - How to setup Kerberos with the Java client
- Semantics of Exclusive
 - I want to be able to have an exclusive consumer, but when it dies I want another to be able to pick up the queue and then block others, can this be done?
 - When will the queue become free for a re-declare
- Performance
 - Does Qpid Perform (Latency/Throughput)?
 - How do I measure throughput?
 - How do I measure latency?
 - How do I measure performance with Java clients?
 - Can I run my Java client with JAVA-RT?
 - Does Qpid support flow control?
 - How do I configure producer side flow control
- Management
 - What Management does Qpid support
 - How do I manage a broker?
 - What logging tracing and events does Qpid support?
 - Can I get to all the management data from a client?
 - What is QMF
 - What are QMF Agents, and what do they do for me?
 - What is QMFC and what does it do for mr?
 - What is QMan

- Clustering, Federation and Disaster Recovery
 - Does Qpid provide Fault Tolerance for the cluster?
 - How do I start a fault tolerant cluster?
 - What does the cluster guarantee?
 - Do clients get notified members joining or leaving the cluster?
 - Can I specify more than one host to connect initially to the cluster to avoid single point of failure?
 - How does Clustering work?
 - What is Federation?
 - Disaster recover features are in process, Q&A will be added once they are complete.
- Heartbeats
 - What would happen when there is a no heartbeat within a predefined interval?
 - What happens if the broker is unable to send heartbeat?
 - Does the client retry?
 - Failover taking too long...
- Threading
 - Could someone provide a brief description of the worker thread duties in the current Qpid release?
 - Why was the number X chosen as the default number of worker threads?
 - What happens in parallel?
 - How are worker threads allocated to individual client sessions if there are more clients than threads in the pool?
- Persistence
 - Does Qpid support persistence (durability)?
 - Where do I get the 3rd party persistence store modules?
 - How do I build the persistence store module from subversion checkouts?
 - Which version of the store should I use when building against qpid 0.X?
 - How do I use the persistence store module?
 - How do I configure the persistence store?
 - [C++ store] What is a RHM_IORES_ENQCAPTHRESH error?
 - [C++ store] What is the TPL? What are the --tpl-* options for?
- How To
 - C++
 - How to use RDMA with Qpid
 - Message TTL, auto expire
 - How to install the qpid-tools for c++ broker?
 - Java

This page is a collection of FAQ and How to-s for Qpid. If you have a question, post it to the users list and we will place the answer here to build out our FAQ/ How to.

FAQ

About AMQP

What is AMQP?

AMQP is a wire-level protocol and model for high performance enterprise messaging.

From the AMQP website:

AMQP is an Open Standard for Messaging Middleware.

By complying to the AMQP standard, middleware products written for different platforms and in different languages can send messages to one another. AMQP addresses the problem of transporting value-bearing messages across and between organizations in a timely manner.

AMQP enables complete interoperability for messaging middleware; both the networking protocol and the semantics of broker services are defined in AMQP.

Where did AMQP come from

AMQP was born out from Frustration by John O'Hara at JPMC. He started a project internally to create commodity messaging that was easy to use. Carl Trieloff from Red Hat had started a project to build messaging for both users and for use in infrastructure, while looking around spoke to John about his work. Out of these discussion was born the AMQP working Group with 6 initial members, under an agreement that it will be eternally be licensed for everyone to use.

Since then the Working Group has had many join, and has been making solid progress working on revisions of the specification. For more details see.

Why use AMQP?

AMQP is has been designed to be able to handle the hardest workloads, scale to the largest systems, but also deal with reduction of change and maintenance costs by doing a refresh on many aged practices. The specification is also not language specific allowing the freedom from language and platform lock in, without compromise on user experience, security, scalability and consistently excellent performance.

Text mostly taken from

Qpid & AMQP

Is Qpid AMQP Compliant?

Yes, Apache Qpid implements the latest AMQP specifications, providing transaction management, queuing, distribution, security, management, clustering, federation and heterogeneous multi-platform support and a lot more. And Apache Qpid is extremely fast. [Apache Qpid aims to be 100% AMQP Compliant](#).

What Client support does Qpid have?

Apache Qpid provides AMQP Client APIs for the following languages:

- C++
- C# .NET, using WCF
- Ruby
- Python
- Java JMS, fully conformant with Java CTS1.1

If you need another client, join the lists and ask or feel free to contribute one.

What messaging topologies are supported by AMQP and Qpid?

AMQP provides the ability to do Point-to-Point, Peer-to-Peer, Pub-Sub, and Eventing. This allows many patterns to be created:

Point-to-point

This is one of the simplest use-cases. AMQP allows for this in a few ways.

- a.) A client can create a named queue allowing the producer to publish the message to the direct exchange with the key mapping the queue name. This will route the message to that queue.
- b.) The above pattern can be extended by specifying a reply-to address in the published messages allowing for the consumer to reply the producer without knowing who it was send from prior to receiving the message.

One-to-many

There are a few patterns that can be used.

- a.) AMQP provides a 'fanout' exchange which will send a message to all the queues that have been bound to it. Different domains or topics are created with the 'fanout' exchange by declaring different named fan-out exchanges.
- b.) A 'topic' or 'headers' exchange can also be used. in this case the pattern match is used to send the message to all the bound queues. It can be thought of as a filter allowing you to create just about any One-to-many routing patterns.

Pub-Sub

Topic can be created with the 'topic' or other 'direct' exchange to allow consumer to bind to into the streams of data they care about. This pattern combined with the use of reply-to and Alternate-routing is the staple of what most people use messaging for today.

FAST Reliable Messaging

AMQP 0-10 allows for fully reliable transfers between any two peers. This means that you can publish or subscribe to the broker fully reliable without requiring the need for transactions. This can all be done in async mode with the C++ broker allowing for high throughput while running entirely reliable.

Transactional

AMQP supports two types of transactions in AMQP 0-10, TX and DTX. This allows for local (1PC), and 2PC transaction and the ability to coordinate with a TM (Transaction Manager). The Java broker supports TX, the C++ broker support TX, DTX, XA, JTA for fully ACID transactions. This allows you to commit a single unit of work with may contain enqueues & dequeues either locally on the broker, or in coordination with other transactional resource like RDBMS.

Transient message delivery

By default messages are transient. Transient message can be sent to queues that are durable. They will not be safe stored or recovered, and will perform as any other transient message - fast!

Durable message delivery

There is a header on each message where the message properties are specified, one of these is durability. Messages that are marked as durable and published to a durable queue will be safe stored. Durable messages will survive restart of the broker or cluster.

Federation (Hub-spoke, Trees, graphs)

As AMQP 0-10 is symmetric for peer-to-peer communication all the building block are in place for creating networks of brokers. The C++ broker allows you to link the brokers together using 'qpid-route' and then create routes between the brokers either statically or with dynamic routes.

This allows for a message to be published to one broker and consumed from another broker in the federated broker network. This feature is great to create data-center, or project isolation, but allow cross communication. It also allows networks to be created to scaled. [For more details see](#)

And many others, including custom pattern

Message Reply, Rings, Initial Value Caches, Last Value Messaging

All the above cases can be constructed using the AMQP and features of Qpid. For example reply can be constructed using message browsing and setting TTL on the messages. The C++ broker also support ring queues, last value queues, initial value caches on exchanges. With a bit of thought many additional patterns can be constructed.

Store-and-forward

Store-and-forward can be achieved by publishing to well known durable queues, that are not marked with auto delete. Consumers will be able to 'come back' to consume them at any time, even after restarts.

What AMQP and other exchanges does Qpid support?

Both brokers support:

- Direct Exchange
- Topic Exchange
- Fanout Exchange
- Headers Exchange

In addition the C++ broker supports

- XML Exchange - Query routing
- Custom exchange via plug-in.

Custom exchanges allow you to provide your own custom routing logic and algorithms via a plug-in. If you build an interesting exchange, please feel free to contribute it back to the Qpid project.

Security

What encryption does Qpid support?

- Qpid supports SSL/TSL as per the AMQP specification.
- In addition the C++ broker supports Kerberos encryption of messages independent on which transport is used. Support is not yet included in all clients for this but is in process.

What authentication does Qpid support?

SASL Authentication is supported. All Clients support PLAIN, and Kerberos support if being added to all the clients. The C++ broker supports Kerberos authentication.

What authorization does Qpid support?

Full ACL is supported in the brokers. [For details on configuring ACL see.](#)

ACL supports realms and allows for granular permission to be set on all the broker actions including management on an user or group basis.

How to setup Kerberos with the Java client

You could force the java client to use kerberos auth by specifying it in the connection URL as follows.

```
amqp://guest:guest@clientid/testpath?brokerlist='tcp://localhost:5672?&sasl_mechs='GSSAPI'
```

You would then need to pass in the following jvm arguments

```
-Djavax.security.auth.useSubjectCredsOnly=false  
# (This will force the SASL GSSAPI client to obtain the kerberos credentials explicitly instead of  
obtaining from the "subject" that owns the current thread)  
-Djava.security.auth.login.config=myjas.conf (this specifies the jass config file)  
-Dsun.security.krb5.debug=true (to enable detailed debug info for troubleshooting)
```

Before running the java client you would need to do kinit and grab a kerberos ticket. Alternative you could set useTicketCache=false and when the client loads, it will prompt you for the user/pass and will obtain the ticket (You would also need to setup your kerberos environment properly -refer to doc links below).

Sample JASS Config file

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

Semantics of Exclusive

I want to be able to have an exclusive consumer, but when it dies I want another to be able to pick up the queue and then block others, can this be done?

Yes, Declare you queue exclusive. this will prevent anyone else from connecting to the queue. If the consumer dies the next consumer can attach to the queue by redeclaring it using the exclusive flag. Make sure not to set auto delete. Any consumer trying to declare, while a consumer is attached to the queue will receive an exception.

When will the queue become free for a re-declare

Once the session that held the consumer is closed.

Performance

Does Qpid Perform (Latency/Throughput)?

Yes, The Qpid C++ broker has been achieved great benchmark results in published papers by those that redistribute it. Red Hat MRG product build on Qpid has shown 760,000msg/sec ingress on an 8 way box or 6,000,000msg/sec OPRA messages.

Latencies have been record as low as 180-250us (.18ms-.3ms) for TCP round trip and 60-80us for RDMA round trip using the C++ broker.

How do I measure throughput?

There is a great resource supplied in the C++ broker test directory called perftest. It allows you to create load on a broker for all the exchanges, multiple queues, multiple connection, coordinate multiple publishing and consuming processes, benchmark transactions and much much more such as acquire mode, txn size, message size.

For all the options

```
./perftest --help
```

How do I measure latency?

There is a great resource supplied in the C++ broker test directory called latencytest. It is a loopback test that produces messages by count or at a rate, time stamps them and then consumes them back and record the latency. It supports many of the Qpid options, including the ability to vary things like frame-size.

Latencies to expect round trip:

- 1G TCP ~ .3ms -.5ms
- 10G TCP - .18ms - .22ms
- RDMA transport - 40us - 80us

Don't forget to set tune the machine and set --tcp-nodelay on both the C++ broker & client.

For all the options

```
./latencytest --help
```

How do I measure performance with Java clients?

In Java we provide a utility called QpidBench. It allows you to test the performance of the native AMQP API in Java for 0-10 and the JMS API against both brokers.

Can I run my Java client with JAVA-RT?

Yes, recently a thread abstraction layer has been added to the Java client allowing it to be used with both the SUN and IBM RT JVMs. This increases the determinism of latency when using the Java client.

Does Qpid support flow control?

yes, AMQP 0-10 allows for flow control on the consumer and producer.

How do I configure producer side flow control

from qpidd --help

set the following in the config file on via cmd line options.

| | |
|--|---|
| --max-session-rate MESSAGES/S (0)
(0=unlimited) | Sets the maximum message rate per session |
|--|---|

Management

What Management does Qpid support

The Java broker supports JMX and provides an Eclipse plug-in and command line tool to manage via JMX. The C++ broker has far more extensive management support via QMF which will be added to the Java broker in a future release.

The C++ Broker supports a layered management protocol over AMQP called QMF. This allows for the management of resource either in the broker or connected to the broker via the AMQP fabric. This management includes statistics, control, eventing, and reporting/updating properties.

How do I manage a broker?

A set of tools are provided to manage the C++ broker, they include

- qpid-tool - telnet type tool to access data, view schema, issue command an and QMF resource
- qpid-config - tool to configure queues, exchanges, etc. all the details on the AMQP model
- qpid-route - tool to configure broker federation
- qpid-events - utility that will print to cmd line or syslog event from a broker like, userconnected, user created/deleted a queue.
- qpid-stats - utility that will print out queue statistics to the cmd line or syslog like rate and message depth.

Then you can also access all this information via JMX or WS-DM (work in progress) using QMan.

What logging tracing and events does Qpid support?

Qpid support the ability to output events from any the broker or any managed object via QMF, or to do a variety of logging from the broker & clients. for tracing options run qpid --help.

Multiple levels of of logging are supported in the C++ broker from debug, warning, error, info, etc – all of which can be filtered.

Can I get to all the management data from a client?

yes, All the management data is just AMQP messages on specially named queues. An API is provided for working with the management data called QMFC

What is QMF

QMF is the layered Management protocol used to manage the C++ broker. For details on teh protocol see the Development pages.

QMF allows you to manage any resource and provides the following infrsstructure:

- Properties
- Statistics
- Commands
- Events
- Schema for resources and versioning
- tools for creating agents and consuming QMF data.

What are QMF Agents, and what do they do for me?

An Agent is any client (producer or consumer) that generates a QMF schema and registers itself to be management by QMF.

A great use case of this is a consumer that is processing order from a queue can reference itself to that queue and for example provide a schema for the number of successful orders process and a method to suspend processing. Now it becomes possible to use qpid-tool to connect to the broker, see which order processors are on queue via the reference and the via the stats of the order processor client. It is also possible to issue a command to the client via qpid-tool to suspend processing. ACL in the broker can be applied to all these actions if desired.

What is QMFC and what does it do for mr?

QMFC is the API used to consume QMF data, event and issue commands to QMF agents from an AMQP client.

What is QMan

Qman is a tool that dynamically reads the QMF Schema information and creates JMX objects that consumed by any JMX console or application server to manage Qpid. QMan is also adding support for WS-DM management of QMF resources.

Clustering, Federation and Disaster Recovery

Does Qpid provide Fault Tolerance for the cluster?

The C++ broker has plug-ins for Active-Active clustering which keep all the nodes of the cluster in sync. This means that any action that is performed on one of the brokers on the cluster is performed on all of them at the same time. New nodes can be added to the cluster at any time, and removed at any time with no consequences, except for the extra multi-cast load created for the sync on joining.

How do I start a fault tolerant cluster?

See [Starting a cluster](#)

What does the cluster guarantee?

Everything! All configuration, all messages and all actions are replicated in a cluster. This means that two consumers can be connected to different nodes in the cluster and they will behave EXACTLY the same as if they were on a single broker.

Do clients get notified members joining or leaving the cluster?

yes, All clients are updated with the addresses of node add/removed as supported by the AMQP 0-10 specification. This means that the client can dynamically track the nodes in the cluster and reconnect as required.

Can I specify more than one host to connect initially to the cluster to avoid single point of failure?

yes, the AMQP address is multi-homed and more than one IP address can be specified at the initial connection. The client will then iterate through the host until it makes a successful connection. This feature can also be used in non-clustered brokers.

How does Clustering work?

When C++ brokers are configured into a cluster, the nodes communicate with each other over a multicast protocol called AIS, an open Telco multicast protocol that provides all the quorum and group services.

Every action that is performed on any node of the cluster is then sequenced via totem and then performed on each node of the cluster in sync. As the cluster backbone is multicast, a separate network can be used for cluster communication and there is little impact adding additional nodes to the cluster with-in reason.

What is Federation?

Federation provides the ability to create networks of brokers that communicate with each other in all types of typologies. This allows a producer to publish messages to one broker and someone to consume the messages from another broker somewhere on the broker federated network.

For more details see [Federation](#)

Disaster recovery features are in process, Q&A will be added once they are complete.

Heartbeats

Heartbeat can be configured to allow clients to detect when a broker has failed and connect to another broker or cluster member. Heartbeats are sent by the broker at a client-specified, per-connection frequency. If the client does not receive a heartbeat or any other traffic for two heartbeat intervals, the connection will be made to fail.

What would happen when there is a no heartbeat within a predefined interval?

If there is no traffic for two heartbeat intervals, the client will fail the connection. The application will see the exact same response as when the connection is killed.

What happens if the broker is unable to send heartbeat?

As above, if there is no other traffic the client will eventually kill the connection.

Does the client retry?

You can control the heartbeat interval on the client through the heartbeat member of ConnectionSettings (it is measured in seconds). Some of the options on policies do vary for different clients.

Failover taking too long...

First check to make sure a heartbeat has been specified in the connection properties for the connection.

Then make sure that the interfaces on each broker are reachable from the host you run my clients, else it will take a long time for the socket to timeout until it gets to one that can be reached.

Make sure the list of URL's on the client are the ones you want the client to try

Make sure that the broker is only exporting URL's that the client can connect to, use the --cluster-url option on the broker to specify this.

Threading

Could someone provide a brief description of the worker thread duties in the current Qpid release?

The broker uses IO threads for all the work it does. This means that when work is signalled via an event (socket, RDMA, timer) an IO thread is scheduled and it runs until it completes the work and then returns back to the IO thread pool. This allows the CPUs to be utilized efficiently. The general rule is that we allocate 1 thread per core +1. So on a 8 way machine you see worker-threads default to 9. On a 4 way it will be 5. Sometimes it if work changing the default allocation if:

- a.) you run on high core count machine >8 to a lower number
- b.) if you taskset, then set to the cores allocated +1

Why was the number X chosen as the default number of worker threads?

Qpid defaults to cores + 1

What happens in parallel?

Concurrency in the broker is at the session level. So yes. If you want more concurrency, create another session on the same connection.

How are worker threads allocated to individual client sessions if there are more clients than threads in the pool?

They are not allocated to a specific client

Persistence

Does Qpid support persistence (durability)?

Yes, there are third-party (non-Apache) modules for both C++ and Java. Historically, BDB has been used to provide persistence for both C++ and Java. However, this has created a licensing conflict with Apache, and thus the store modules are maintained off-site.

The Java broker includes a fully Apache licensed persistent store that uses Derby DB.

The terms *durable* and *persistent* are used interchangeably in this FAQ.

Where do I get the 3rd party persistence store modules?

The 3rd party persistence store modules may be obtained through anonymous subversion at the following locations:

C++: <http://anonsvn.jboss.org/repos/rhmessaging/store/trunk/cpp>

Java: <http://anonsvn.jboss.org/repos/rhmessaging/store/trunk/java/bdbstore>

For further details see 3rd Party Libraries

How do I build the persistence store module from subversion checkouts?

C++*The README file contains detailed instructions, but here is a summary:

1. Make sure that both the db4-devel and libaio-devel packages are installed prior to building.
2. Make sure that qpid is built and you know the location of the qpid directory (ie the top-level directory containing the python and cpp sub-directories).
3. In the store directory, run:

```
./bootstrap  
./configure --with-qpid-checkout=/abs/path/to/qpid/dir  
make
```

4. When built, the store library **msgstore.so** will be located in the **lib.libs** directory.

JavaTODO

Which version of the store should I use when building against qpid 0.X?

C++

If you build qpid from svn trunk, you should be able to build the store against it using the store trunk. However, if you build the store from a released version of qpid, you will need to check out a specific version of the store to get it to compile:

| release | store tag | store revision |
|---------|------------------|----------------|
| 0.5 | qpid-0.5-release | 3373 |
| 0.6 | qpid-0.6-release | 3793 |

To check out revision revno, use:

```
svn co http://anonsvn.jboss.org/repos/rhmessaging/store/trunk/cpp -r [revno]
```

To check out tag `tagname`, use:

```
svn co http://anonsvn.jboss.org/repos/rhmessaging/store/tags/[tagname]/cpp
```

JavaTODO

How do I use the persistence store module?

C++

1. Start the broker making sure that the store module is loaded, ie

```
qpidd --load-module=/path/to/msgstore.so --data-dir=/path/to/store-files ...
```

2. Make sure that queues that will handle persistent messages are set durable.



Note: Existing non-persistent queues cannot be made persistent

If a queue has been declared without persistence, doing so again with persistence enabled while the old queue still exists in the broker will be ignored. Make sure that when a queue is declared persistent, there is no non-persistent queue of the same name in existence.

3. For each message sent to a durable queue, make sure that it is set durable.

JavaTODO

How do I configure the persistence store?

C++

The broker loads help information from each module. To see the help options for the store, load the store module and specify help:

```
qpidd --load-module /abs/path/to/store/lib/.libs/msgstore.so --help
```

Note that a set of journal files will be created for each queue declared and marked persistent. Each persistent queue has its own private journal. These are stored in the data directory by default (ie it uses the broker's `--data-dir` setting) or can be overridden with the `--store-dir` option. Note that if the broker is started with the `--no-data-dir` option, then no store default exists, and the `--store-dir` option MUST be specified.

The store file details - or "store geometry" - can be set with command-line options. These include the size and number of files that make up the journal for each queue. The `--num-jfiles` option sets the number of files to use (between 4 and 64) and the `--file-size-pgs` sets the size of the file in 64kB blocks.

The size of the pages in the write page cache is set with the `--wcache-page-size` option, and sets a size in KiB. (Legal values are powers of 2, ie: 1, 2, 4, 8, 16, 32, 64, 128). Typically small page sizes give improved latency (especially for small messages), but are bad for message throughput, while large page sizes improve throughput but may cause some messages to have higher latencies.

[Java Derby Store](#)For details of configuring the Derby Store see [here]

3rd Party Stores

For details of using the 3rd party persistent modules see [here](#)

[C++ store] What is a RHM_IORES_ENQCAPTHRESH error?

The journal ran out of space (ENQueue CAPacity THRESHold). The journal is a circular file buffer of fixed capacity set by the journal file size and number of files. When an attempt to write a record causes the journal to exceed an approx. 80% threshold, then the enqueue is rejected with this error code. Dequeues (a written record of a consumed message) may continue, however, as these free up space in the journal. Once space has been freed up, enqueues may continue as normal.

This error may be caused by:

1. The journal is too small for the size and number of messages being stored. The journal must be made large enough to hold all of the messages you expect to be on the queue at any one moment (a worst-case scenario). Make the journal capacity larger through the use of the `--num-jfiles` and `--file-size-pgs` parameters.



Rule of thumb for sizing the journal

Make the journal twice the size of all the messages you need to store at any one moment in time.

2. Messages are not being dequeued (consumed) as expected. Since the store is a circular file buffer, if one un-dequeued (not consumed) message remains, it can eventually "block" the storage of new messages as the buffer gets overwritten.

[C++ store] What is the TPL? What are the --tpl-* options for?

The TPL stands for **Transaction Prepared List**. The store creates a single instance of a journal for storing transaction boundaries called the Transaction Prepared List. Because the TPL is frequently flushed and has very different usage patterns to a normal store, it has been provided with its own set of configuration parameters:

- **--tpl-num-jfiles:** The number of files in the TPL journal
- **--tpl-jfile-size-pgs:** The file size in 64KiB blocks of the TPL journal.
- **--tpl-wcache-page-size:** The size of the write cache in the TPL in KiB, which is typically set a lot smaller than the average message store.

How To

C++

How to use RDMA with Qpid

The RDMA plugin uses native OFED1.3 and puts AMQP directly onto the DMA. When using the RDMA plug-in for Qpid note the following

- IP over IB or Fibre needs to be setup for the initial negotiation
- You need to make sure you have enough memory to pin for DMA use ulimit -l something large
- you might need to edit /etc/security/limits.conf first then log in again

Once you have it up and running, use latencytest to make sure it is working. You should see latencies between 50 and 80us round trip.

Message TTL, auto expire

I need to be able to set time for a message that I send to be removed from the queue if it is not read by my subscriber. For example: I enqueue a message and I want it to be automatically dequeued after a certain amount of time has passed. Is there a feature like this in qpid?

yes, the TTL can be set in the message headers and the messages get dequeued if TTL expires

E.g. from c++:

```
Message m( "Hello World!" );
m.getDeliveryProperties().setTtl(500);
```

Sets a 500 millisecond timeout.

How to install the qpid-tools for c++ broker?

I see

```
[commands]$ ./qpid-queue-stats
Traceback (most recent call last):
File "./qpid-queue-stats", line 29, in
from qmf.console import Session, Console
ImportError: No module named qmf.console
```

This problem occurs because the PYTHONPATH environment variable does not include the location of the qpid python files. If you are running from the SVN checkout, add <path>/qpid/python to PYTHONPATH (where <path> is the location of your SVN tree). If you are installing from source, make sure you configure with the same prefix where Python is installed. This is most likely:

```
# configure --prefix=/usr
# make
# make install
```

If you are running from vendor RPMs, this should work automatically.

Java

- Add New Users
- Configure ACLs
- Configure Broker and Client Heartbeating
- Configure Java Qpid to use a SSL connection.

- Configure Log4j CompositeRolling Appender
- Configure Operational Status Logging
- Configure the Broker via config.xml
- Configure the Virtual Hosts via virtualhosts.xml
- Debug using log4j
- Firewall Configuration
- How to Tune M3 Java Broker Performance
- How to Use JNDI
- Interact with a JMX MBean
- Qpid Java Build How To
- Split configuration files
- Tune Broker and Client Memory Usage
- Use Last Value Queues (LVQ)
- Use Priority Queues
- Use Producer Flow Control

License

Qpid is licensed in under the ASL 2.0. Please look at the notice files provided with the downloads to see the list of embedded components.

Apache License

Version 2.0, January 2004

<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

1. You must give any other recipients of the Work or Derivative Works a copy of this License; and
1. You must cause any modified files to carry prominent notices stating that You changed the files; and
1. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
1. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

Project Status

Qpid is graduated November 2008 as TLP

Qpid will be transitioning to it new home out of the incubator. If you find pages that need updating, please mail the qpid-dev list.

If you would like to become a committer and join the PMC, [this is how we do it](#). We would love to have you on the project!

Our Resolution, Approved November 2008.

Here is a copy of our resolution:

WHEREAS, the Board of Directors deems it to be in the best interests of the Foundation and consistent with the Foundation's purpose to establish a Project Management Committee charged with the creation and maintenance of open-source software related to distributed messaging, for distribution at no charge to the public.

NOW, THEREFORE, BE IT RESOLVED, that a Project Management Committee (PMC), to be known as the "Apache Qpid Project", be and hereby is established pursuant to Bylaws of the Foundation; and be it further

RESOLVED, that the Apache Qpid Project be and hereby is responsible for the creation and maintenance of software related to distributed messaging; a multiple language implementation providing daemons and APIs for publish & subscribe, eventing and a wide range of message distribution patterns based on the Advanced Message Queuing Protocol (AMQP) and related technologies such as (transaction management, federation, security, management); and be it further

RESOLVED, that the office of "Vice President, Qpid" be and hereby is created, the person holding such office to serve at the direction of the Board of Directors as the chair of the Apache Qpid Project, and to have primary responsibility for management of the projects within the scope of responsibility of the Apache Qpid Project; and be it further

RESOLVED, that the persons listed immediately below be and hereby are appointed to serve as the initial members of the Apache Qpid Project:

- Aidan Skinner aidan.skinner@gmail.com
- Alan Conway aconway@redhat.com
- Arnaud Simon asimon@redhat.com
- Carl Trieloff cctrielloff@redhat.com
- Craig Russell Craig.Russell@sun.com
- Gordon Sim gsim@redhat.com
- Jonathan Robie jonathan.robie@redhat.com
- John O'Hara john.r.ohara@gmail.com
- Kim van der Riet kim.vdriet@redhat.com
- Marnie McCormack marnie.mccormack@googlemail.com
- Martin Ritchie ritchiem@apache.org
- Manuel Teira mteira@tid.es
- Paul Fremantle paul@wso2.com
- Nuno Santos nsantos@redhat.com
- Rafael Scholmeling rafaels@redhat.com
- Rajith Attapattu rattapat@redhat.com
- Robert Greig robert.j.greig@gmail.com
- Robert Godfrey rob.j.godfrey@gmail.com
- Steve Huston shuston@riverace.com
- Ted Ross tross@redhat.com
- Yoav Shapira yoavs@apache.org

NOW, THEREFORE, BE IT FURTHER RESOLVED, that Carl Trieloff be appointed to the office of Vice President, Qpid, to serve in accordance with and subject to the direction of the Board of Directors and the Bylaws of the Foundation until death, resignation, retirement, removal or disqualification, or until a successor is appointed; and be it further

RESOLVED, that all responsibility pertaining to the Qpid encumbered upon the Apache Incubator be hereafter discharged.

General items

- The Qpid project proposal can be found at <http://wiki.apache.org/incubator/QpidProposal>
- The project containing the initial source and mail-list for reference prior to the incubator can be found at [here](#)

People

Apache Qpid Committers

The people listed below have made significant contributions to Qpid by working long and hard to make quality software for the rest of the world to use.

In addition to providing us contributions, or being committers some of the following people are also members of the Project Management Committee (PMC). Refer to the How the ASF works for details on meritocracy.

If you would like to contribute to Qpid please look at the Get Involved page to see how you can contribute.

| | | | |
|--------------------|----------------|------------------|------------------|
| Aidan Skinner | Alan Conway | Arnaud Simon | Andrea Gazzarini |
| Andrew Stitzer | Carl Trieloff | Gordon Sim | Jim Meyering |
| Ken Giusti | John O'Hara | Jonathan Robie | Kim van der Riet |
| Lahiru Gunathilake | Manuel Teira | Marnie McCormack | Martin Ritchie |
| Michael Goulish | Nuno Santos | Paul Fremantle | Rafael Schloming |
| Rajith Attapattu | Robbie Gemmell | Robert Godfrey | Robert Greig |
| Rupert Smith | Steve Huston | Ted Ross | Yoav Shapira |

Many thanks to the following people for providing contributions:

| | | | |
|-------------|--------------------|-------------|---------------|
| Colin Crist | Bhupendra Bhardwaj | Kevin Smith | Steve Vinoski |
| Steven Shaw | Tomas Restrepo | | |

And many thanks to our project's mentors:

Scott Deboy
Paul Fremantle
Craig Russell
Cliff Schmidt
Yoav Shapira

MartinRitchie

This is just a sandbox test area for Martin Ritchie

Cloaked Content2

Robbie Gemmell

GSoC 2009 Progress

The table below gives an overview of current progress.

| Issue # | Description | Target Week | % Complete | Notes |
|-----------|--|-------------|------------|-----------------------|
| QPID-1926 | make qpid-management-common an OSGI bundle | | 100 | Done, merged to trunk |
| QPID-1927 | move JMX MBean interface definitions to management-common | | 100 | Done, merged to trunk |
| QPID-1928 | remove Queue MBean reliance on AMQException for unused throws clause | | 100 | Done, merged to trunk |
| QPID-1929 | create a factory class to generate the views for the various MBeans | | 100 | Done, merged to trunk |
| QPID-1930 | create a new view for the User Management mbean | | 100 | Done, merged to trunk |
| QPID-1931 | create a new view for the Logging Management mbean | | 100 | Done, merged to trunk |
| QPID-1942 | create a new queue / exchange / connection selection view | | 100 | Done, merged to trunk |
| QPID-1945 | create a new view for the VirtualHostManager mbeans | | 100 | Done, merged to trunk |

| | | | | |
|-----------|--|----|-----|--------------------------|
| QPID-1941 | moved messages remain listed on original queue when viewing messages using JMX, but are not actually considered to still be on the queue | 7 | 100 | Done, merged to trunk |
| QPID-1932 | create a new view for the Queue mbeans | 7 | 100 | Done, merged to trunk |
| QPID-1944 | create a new view for the Connection mbeans | 7 | 100 | Done, merged to trunk |
| QPID-1943 | create a new view for the Exchange mbeans | 7 | 100 | Done, merged to trunk |
| QPID-1946 | add an mbean to present system information, including API versioning for the JMX interface | 8 | 100 | Done, committed to trunk |
| QPID-1966 | add status bar for feedback reports | 8 | 100 | Done, committed to trunk |
| QPID-1947 | enable automated update to negate need for users to manually refresh the data | 8 | 100 | Done, committed to trunk |
| QPID-1991 | remove the containing Type folders for the top-level single mbeans in the server navigation tree | 8 | 100 | Done, committed to trunk |
| QPID-1990 | provide ability to determine queue order of messages when viewing via JMX | 8 | 100 | Done, committed to trunk |
| QPID-1969 | per-virtualhost notification areas show all notifications sent by the server | 9 | 100 | Done, committed to trunk |
| QPID-1967 | gather list of available exchange types from broker instead of hardcoding in console | 9 | 100 | Done, committed to trunk |
| QPID-1961 | widen viewMessages(int From, int To) AMQQueueMBean method to use Long values | 9 | 100 | Done, committed to trunk |
| QPID-1968 | expose ability to delete arbitrary message from queue | 9 | 100 | Done, committed to trunk |
| QPID-1981 | Expose message copying ability through JMX | 9 | 100 | Done, committed to trunk |
| QPID-1994 | auto-refresh mechanism can generate NPE during application shutdown | 9 | 100 | Done, committed to trunk |
| QPID-1995 | Notification tabs all start a thread on creation, and never halt them | 9 | 100 | Done, committed to trunk |
| QPID-1996 | Notifications view clears the table and forces deselection at every refresh | 9 | 100 | Done, committed to trunk |
| QPID-2000 | retrieving attributes for the queue selection list takes excessively long with large numbers of queues | 9 | 100 | Done, committed to trunk |
| QPID-1978 | New list views in MC should allow multiple selection | 9 | 100 | Done, committed to trunk |
| QPID-2006 | enable opening queues and exchanges directly from other mbean views | 9 | 100 | Done, committed to trunk |
| QPID-1977 | No way back from object view | 10 | 100 | Done, committed to trunk |
| QPID-2007 | new UI dialogs open in the upper left corner of the screen on Windows | 10 | 100 | Done, committed to trunk |
| QPID-2008 | queue names are not sorted when presented for user selection in some new UI dialogs | 10 | 100 | Done, committed to trunk |
| QPID-2009 | password fields are not masked in the new UserManagement UI | 10 | 100 | Done, committed to trunk |
| QPID-2013 | Previous/Next N Message button in queue browser should take current size as step size | 11 | 100 | Done, committed to trunk |
| QPID-2014 | Clear notifications should prompt when clearing all of them | 11 | 100 | Done, committed to trunk |

| | | | | |
|-----------|---|----|-----|---|
| QPID-2015 | Add ability to configure attributes shown on queue list | 11 | 100 | Done, committed to trunk |
| QPID-2021 | provide new icons to distinguish the various 'manager' mbeans | 11 | 100 | Done, committed to trunk |
| QPID-2032 | AddUser operation in the new UserManagement UI does not MD5 hash the password when connected to pre-0.5 brokers | 11 | 100 | Done, committed to trunk |
| QPID-2036 | SimpleAMQQueue getMessagesRangeOnTheQueue(from,to) incorrectly shows the last message on a queue following its deletion | 11 | 100 | Done, committed to trunk |
| QPID-2037 | navigation tree node doesnt auto-expand upon connection when adding a new server | 11 | 100 | Done, committed to trunk |
| QPID-2018 | Clear queue doesn't clear acquired messages which is a little confusing if that's all there is | 12 | 100 | Done, committed to trunk |
| QPID-2016 | Reload Log4J configuration file | 12 | 60 | Developing a custom XMLWatchdog to enforce stricter validation to the WatchDog reloading, also to be used on conjunction with the forced reload. May prompt some changes to logging configuration during broker startup |
| QPID-2039 | the JMX ConnectorServer is not closed during shutdown of the JMXManagedObjectRegistry | 12 | 0 | |
| QPID-2043 | create a new JMX management console testing spec | 12 | 0 | |
| QPID-2044 | create a new JMX management console user guide | 12 | 0 | |
| QPID-2040 | failed update to the PlainPasswordFilePrincipalDatabase on disk may go unnoticed, preventing broker restart | | 0 | |
| QPID-2041 | failed update to the Base64MD5PasswordFilePrincipalDatabase on disk may go unnoticed, preventing broker restart | | 0 | |
| QPID-2042 | update to the management access rights may not be saved to disk, but still report success | | 0 | |

Proposal for a new JMS Destination configuration

The proposal is organized as follows.

1. Use cases
2. Design concepts/notes
3. Configuration format with examples
4. Complete list of options available
5. Code patch (attached to JIRA)

Use cases

The following were requested by Qpid users via JIRA's and user list.

- Arbitrary exchange types (Ex XML exchange).
- Any kind of queue declare options (Ex. qpid.max-size, alt-exchange)
- Any kind of queue binding options
- Ability to support destination specific parameters like
 - msg credits, byte credits
 - sync-publish, sync-ack
 - whether a queue should be created/bound by producer side
- Bind a queue to multiple exchange/binding key pairs.

Design concepts/notes

- I have moved away from the previous URL format as it,
 - Does not clearly identify a resource, hence against the concept of a URL
 - It is impossible to fit all information in to a URL
- The new format is integrated alongside the old system with absolutely no change to existing way of doing things. A mix and match of both the old and new system could be used (if really needed).
- The new format takes ideas from the AMQP 1.0 spec.
But it is not intended to support AMQP 1.0 when it comes out. If it ends up being a pre-cursor for supporting AMQP 1.0 it would just be a bonus.
- The new format clearly identifies the dual role of a javax.jms.Destination. That being the producer and consumer's view of a destination.
- The new format allows a way to support,
 - Arbitrary exchange types (Ex XML exchange).
 - Any kind of queue declare options (Ex. qpid.max-size, alt-exchange)
 - Any kind of queue binding options
 - Ability to support destination specific parameters like
 - msg credits, byte credits
 - sync-publish, sync-ack
 - whether a queue should be created/bound by producer side
 - Bind a queue to multiple exchange/binding key pairs.
- Define queues, links and then compose them to create destinations.
- Provides sensible defaults . At least I tried to 😊.

Configuration format with examples

The new format consists of definitions for queues, publisher/consumer links and destinations in key/value pairs.

```
xqueue.<id> = name='value1'[,key2='value2';key3='value3'.....]
pub.link.<id> = key1='value1';key2='value2';key3='value3'.....
sub.link.<id> = key1='value1';key2='value2';key3='value3'.....
xdestination.<jndiName> = queue='<id>'[,pub.link='<id>';sub.link='<id>']
```

Using queue, pub/sub links def's you can compose destinations.

Examples

In the simplest form

```
xqueue.myQueue = name='myQueue'
xdestination.myQueue = queue=myQueue
```

This is equivalent to the old queue = myQueue format.

Using qpid specific options and per destination switches

```
xqueue.tradeQueue1 = name='trade-queue1';durable='true'
xqueue.tradeQueue2 = name='trade-queue2';qpid.max_size='5000';qpid.policy_type='ring'

pub.link.trade1 = filter='amq.direct/tradeQueue1';sync-publish='all'
pub.link.trade2 = filter='amq.direct/tradeQueue2';create-queue='true'

sub.link.mylink = msg-credits='1000';byte-credits='1000';sync-ack='true'

xdestination.myLocalTrades = queue='tradeQueue1';pub.link='trade1';sub.link='myLink'

xdestination.myDailyTrades = queue='tradeQueue2';pub.link='trade2';sub.link='myLink'
```

Binding a queue to multiple exchange/routing key pairs

Using the above queue definition.

```
sub.link.multiLink =  
msg-credits='1000';bindings='{amq.topic/stocks.*};{amq.match//x-match='any',sym='RHT'}'  
  
xdestination.myDailyTrades = queue='tradeQueue2';sub.link='multiLink'
```

Complete list of options

- xqueue
 - name : name of the queue
 - durable
 - exclusive
 - auto-delete
 - alt-exchange
 - no-local (??)
 - qpid.max_count
 - qpid.max_size
 - qpid.policy_type { reject | flow_to_disk | ring | ring_strict }
 - qpid.last_value_queue {1}
 - qpid.last_value_queue_no_browse {1}
 - qpid.LVQ_key
 - qpid.persist_last_node {1}
 - qpid.queue_event_generation { 0,1,2 } (0 to disable,1 to replicate, only enqueue events)
- sub.link
 - filter
 - filter-type
 - msg-credits
 - byte-credits
 - sync-ack
 - bindings - format as follows
- pub.link
 - filter
 - filterType
 - sync-publish {persistent|all}
 - create-queue (producer side will declare/bind the queue)

```
{exchange-name/bindingkey[/key=value,key=value,...];{....}...]
```

Proposal for a new JMS Destination configuration2

The proposal is organized as follows.

This proposal is written with input from Rafael and Rob and is intend to support AMQP 1.0 and AMQP 0-8/9/10 as well.

1. Use cases
2. Configuration format with examples
3. Complete list of options available
4. Code patch (attached to JIRA)

1.0 Use cases

The following were requested by Qpid users via JIRA's and user list.

- Arbitrary exchange types (Ex XML exchange).
- Any kind of queue declare options (Ex. qpid.max-size, alt-exchange)
- Any kind of queue binding options
- Ability to support destination specific parameters like
 - o msg credits, byte credits
 - o sync-publish, sync-ack
 - o whether a queue should be created/bound by producer side

2.0 Configuration format with examples

```

queueex.<name> = {key='value', ...}
destinationx.<name> = <address>                                # destination without
options
destinationx.<name> = <address>; {a='b', c={x='y', z='w', ...}, ...} # destination with options

```

- In a production environment the queues will most likely be pre-configured using an admin tool. However if you need to create queues dynamically the queueex definition allows you to configure your queues in the jndi.properties file it self. It's just a simple key-value pair.
- The address will encode the following (scheme, node-name, node-type*, subject*) with * denoting optional elements.
- **The exact format is not decided yet. Rafael is planning to use a similar struct for the python client. It would be great if all clients use a similar struct and encoding scheme to represent an address scheme**
- The use of curly braces in the option key-value pairs in the destination is used to facilitate nesting.

'For the purpose of the examples we will assume the following encoding for the address struct'.

```
<node-type>::<node-name>/<subject>
```

2.1 Examples

```

<node-name>
<node-type>::<node-name>
<node-type>::<node-name>/<subject>

exchange::amq.topic/foo.bar
my-node
my-topic/my-subject.
my-queue/my-subject
my-exchange/my-Subject

```

2.2 Rules for interpreting the format

- For 0-10/9/8, node names without a specified type will be resolved to either a queue or an exchange by querying the broker.
- Subscribing to an exchange will result in a private queue being automatically created and bound (the subject if provided will be used as the binding key).
- Publishing to an exchange will result in the subject being used as the routing key. Publishing to a queue will result in the default exchange being used to route directly to the queue (note that this will result in the routing key being set to the queue name).
- In both cases the subject will be set as a message property.

Complete list of options

The options here are not a static list. The format is extensible and new options could be introduced any time in the future.

- queueex
 - name : name of the queue
 - durable
 - exclusive
 - auto-delete
 - alt-exchange
 - no-local (??)
 - qpid.max_count
 - qpid.max_size
 - qpid.policy_type { reject | flow_to_disk | ring | ring_strict }
 - qpid.last_value_queue {1}
 - qpid.last_value_queue_no_browse {1}
 - qpid.LVQ_key
 - qpid.persist_last_node {1}
 - qpid.queue_event_generation { 0,1,2 } (0 to disable, 1 to replicate, only enqueue events)
- destinationx.
 - filter
 - filter-type
 - msg-credits
 - byte-credits

- sync-ack
- sync-publish {persistent|all}
- bindings - format as follows - a Qpid specific extension for binding a queue to multiple exchange/binding key pairs

```
{ bind={exchange-name=<name>, binding-key=<key>, [args={a='b',c='d',...}]} [,  
bind={} ] }
```

Qpid .Net Documentation

Purpose

Introduction

Currently the .NET code base provides two client libraries that are compatible respectively with AMQP 0.8 and 0.10.
The 0.8 client is located in qpid\dotnet and the 0.10 client in: qpid\dotnet\client-010

You will need an AMQP broker to fully use those client libraries. Qpid trunk currently provide a C++ 0.10 broker and a 0.8/0.9 Java broker.

User Guide

[.NET User Guide](#)
[Excel AddIn](#)
[WCF](#)

Developer Information

- [Qpid Developer Documentation](#)
- [\[Coding Standards\]](#)
- How Tos
 - [Build .NET Client](#)
 - [Releasing](#)
 - [Run tests](#)
 - [Setup .Net Client on Windows](#)

.NET User Guide

Tutorial

This tutorial consists of a series of examples using the three most commonly used exchange types - Direct, Fanout and Topic exchanges. These examples show how to write applications that use the most common messaging paradigms.

- direct
In the direct examples, a message producer writes to the direct exchange, specifying a routing key. A message consumer reads messages from a named queue. This illustrates clean separation of concerns - message producers need to know only the exchange and the routing key, message consumers need to know only which queue to use on the broker.
- fanout
The fanout examples use a fanout exchange and do not use routing keys. Each binding specifies that all messages for a given exchange should be delivered to a given queue.
- pub-sub
In the publish/subscribe examples, a publisher application writes messages to an exchange, specifying a multi-part key. A subscriber application subscribes to messages that match the relevant parts of these keys, using a private queue for each subscription.
- request-response
In the request/response examples, a simple service accepts requests from clients and sends responses back to them. Clients create their own private queues and corresponding routing keys. When a client sends a request to the server, it specifies its own routing key in the reply-to field of the request. The server uses the client's reply-to field as the routing key for the response.

Running the Examples

Before running the examples, you need to unzip the file Qpid.NET-net-2.0-M4.zip, the following tree is created:

```

<home>
|-qpid
  |-lib (contains the required dlls)
  |-examples
    |- direct
      |-example-direct-Listener.exe
      |-example-direct-Producer.exe
    |- fanout
      |-example-fanout-Listener.exe
      |-example-fanout-Producer.exe
    |- pub-sub
      |-example-pub-sub-Listener.exe
      |-example-pub-sub-Publisher.exe
    |- request-response
      |-example-request-response-Client.exe
      |-example-request-response-Server.exe

```

Make sure your PATH contains the directory <home>/qpid/lib
The examples can be run by executing the provided exe files:

```

$ cd <home>/qpid/examples/examplefolder
$ example-...-.exe [hostname] [portnumber]

```

where [hostname] is the qpid broker host name (default is localhost) and [portnumber] is the port number on which the qpid broker is accepting connection (default is 5672).

Creating and Closing Sessions

All of the examples have been written using the Apache Qpid .NET 0.10 API. The examples use the same skeleton code to initialize the program, create a session, and clean up before exiting:

```

using System;
using System.IO;
using System.Text;
using System.Threading;
using org.apache.qpid.client;
using org.apache.qpid.transport;

...
private static void Main(string[] args)
{
    string host = args.Length > 0 ? args[0] : "localhost";
    int port = args.Length > 1 ? Convert.ToInt32(args[1]) : 5672;
    Client connection = new Client();
    try
    {
        connection.connect(host, port, "test", "guest", "guest");
        ClientSession session = connection.createSession(50000);

        //----- Main body of program -----

        connection.close();
    }
    catch (Exception e)
    {
        Console.WriteLine("Error: \n" + e.StackTrace);
    }
}
...

```

Writing Direct Applications

This section describes two programs that implement direct messaging using a Direct exchange:

- org.apache.qpid.example.direct.Producer (from example-direct-producer) publishes messages to the amq.direct exchange, using the

routing key routing_key.
•org.apache.qpid.example.direct.Listener (from example-direct-Listener) uses a message listener to receive messages from the queue named message_queue.

Running the Direct Examples

1) Make sure your PATH contains the directory <home>/qpid/lib

2) Make sure that a qpid broker is running:

```
$ ps -eaf | grep qpidd
```

If a broker is running, you should see the qpidd process in the output of the above command.

3) Read the messages from the message queue using direct listener, as follows:

```
$ cd <home>/qpid/examples/direct
```

With cygwin:

```
$ ./example-direct-Listener.exe [hostname] [portnumber]
```

or with mono:

```
$ mono ./example-direct-Listener.exe [hostname] [portnumber]
```

This program is waiting for messages to be published, see next step:

4) Publish a series of messages to the amq.direct exchange by running direct producer, as follows:

```
$ cd <home>/qpid/examples/direct
```

With cygwin:

```
$ ./example-direct-Producer.exe [hostname] [portnumber]
```

or with mono:

```
$ mono ./example-direct-Producer.exe [hostname] [portnumber]
```

This program has no output; the messages are routed to the message queue, as instructed by the binding.

5) Go to the windows where you are running your listener. You should see the following output:

```
Message: Message 0
Message: Message 1
Message: Message 2
Message: Message 3
Message: Message 4
Message: Message 5
Message: Message 6
Message: Message 7
Message: Message 8
Message: Message 9
Message: That's all, folks!
```

Now we will examine the code for each of these programs. In each section, we will discuss only

the code that must be added to the skeleton shown in Section "Creating and Closing Sessions".

Reading Messages from the Queue

The program , listener.cs, is a message listener that receives messages from a queue.

First it creates a queue named message_queue, then binds it to the amq.direct exchange using the binding key routing_key.

```
//----- Main body of program -----
// Create a queue named "message_queue", and route all messages whose
// routing key is "routing_key" to this newly created queue.
session.queueDeclare("message_queue");
session.exchangeBind("message_queue", "amq.direct", "routing_key");
```

The queue created by this program continues to exist after the program exits, and any message whose routing key matches the key specified in the binding will be routed to the corresponding queue by the broker. Note that the queue could have been deleted using the following code:

```
session.queueDelete("message_queue");
```

To create a message listener, create a class derived from IMessageListener, and override the messageTransfer method, providing the code that should be executed when a message is received.

```
public class MessageListener : IMessageListener
{
    .....
    public void messageTransfer(IMessage m)
    {
        .....
    }
}
```

The main body of the program creates a listener for the subscription; attaches the listener to a message queue; and subscribe to the queue to receive messages from the queue.

```
lock (session)
{
    // Create a listener and subscribe it to the queue named "message_queue"
    IMessageListener listener = new MessageListener(session);
    session.attachMessageListener(listener, "message_queue");
    session.messageSubscribe("message_queue");
    // Receive messages until all messages are received
    Monitor.Wait(session);
}
```

The MessageListener's messageTransfer() function is called whenever a message is received. In this example the message is printed and tested to see if it is the final message. Once the final message is received, the messages are acknowledged.

```
BinaryReader reader = new BinaryReader(m.Body, Encoding.UTF8);
byte[] body = new byte[m.Body.Length - m.Body.Position];
reader.Read(body, 0, body.Length);
ASCIIEncoding enc = new ASCIIEncoding();
string message = enc.GetString(body);
Console.WriteLine("Message: " + message);
// Add this message to the list of message to be acknowledged
_range.add(m.Id);
if( message.Equals("That's all, folks!") )
{
    // Acknowledge all the received messages
    _session.messageAccept(_range);
    lock(_session)
    {
        Monitor.Pulse(_session);
    }
}
```

Publishing Messages to a Direct Exchange

The second program in the direct example, Producer.cs, publishes messages to the amq.direct exchange using the routing key routing_key.

First, create a message and set a routing key. The same routing key will be used for each message we send, so you only need to set this property once.

```
IMessage message = new Message();
// The routing key is a message property. We will use the same
// routing key for each message, so we'll set this property
// just once. (In most simple cases, there is no need to set
// other message properties.)
message.DeliveryProperties.setRoutingKey("routing_key");
```

Now send some messages:

```
// Asynchronous transfer sends messages as quickly as
// possible without waiting for confirmation.
for (int i = 0; i < 10; i++)
{
    message.clearData();
    message.appendData(Encoding.UTF8.GetBytes("Message " + i));
    session.messageTransfer("amq.direct", message);
}
```

Send a final synchronous message to indicate termination:

```
// And send a synchronous final message to indicate termination.
message.clearData();
message.appendData(Encoding.UTF8.GetBytes("That's all, folks!"));
session.messageTransfer("amq.direct", "routing_key", message);
session.sync();
```

Writing Fanout Applications

This section describes two programs that illustrate the use of a Fanout exchange.

- Listener.cs makes a unique queue private for each instance of the listener, and binds that queue to the fanout exchange. All messages sent to the fanout exchange are delivered to each listener's queue.
- Producer.cs publishes messages to the fanout exchange. It does not use a routing key, which is not needed by the fanout exchange.

Running the Fanout Examples

1) Make sure your PATH contains the directory <home>/qpid/lib

2) Make sure that a qpid broker is running:

```
$ ps -eaf | grep qpidd
```

If a broker is running, you should see the qpidd process in the output of the above command.

3) In separate windows, start one or more fanout listeners as follows:

```
$ cd <home>/qpid/examples/direct
```

With cygwin:

```
$ ./example-fanout-Listener.exe [hostname] [portnumber]
```

or with mono:

```
$ mono ./example-fanout-Listener.exe [hostname] [portnumber]
```

The listener creates a private queue, binds it to the amq.fanout exchange, and waits for messages to arrive on the queue. When the listener starts, you will see the following message:

```
Listening
```

This program is waiting for messages to be published, see next step:

4) In a separate window, publish a series of messages to the amq.fanout exchange by running fanout producer, as follows:

```
$ cd <home>/qpid/examples/direct
```

With cygwin:

```
$ ./example-fanout-Producer.exe [hostname] [portnumber]
```

or with mono:

```
$ mono ./example-fanout-Producer.exe [hostname] [portnumber]
```

This program has no output; the messages are routed to the message queue, as prescribed by the binding.

5) Go to the windows where you are running listeners. You should see the following output for each listener:

```
Message: Message 0
Message: Message 1
Message: Message 2
Message: Message 3
Message: Message 4
Message: Message 5
Message: Message 6
Message: Message 7
Message: Message 8
Message: Message 9
Message: That's all, folks!
```

Now we will examine the code for each of these programs. In each section, we will discuss only the code that must be added to the skeleton shown in Section "Creating and Closing Sessions".

Consuming from a Fanout Exchange

The first program in the fanout example, Listener.cs, creates a private queue, binds it to the amq.fanout exchange, and waits for messages to arrive on the queue, printing them out as they arrive. It uses a Listener that is identical to the one used in the direct example:

```

public class MessageListener : IMessageListener
{
    private readonly ClientSession _session;
    private readonly RangeSet _range = new RangeSet();
    public MessageListener(ClientSession session)
    {
        _session = session;
    }

    public void messageTransfer(IMessage m)
    {
        BinaryReader reader = new BinaryReader(m.Body, Encoding.UTF8);
        byte[] body = new byte[m.Body.Length - m.Body.Position];
        reader.Read(body, 0, body.Length);
        ASCIIEncoding enc = new ASCIIEncoding();
        string message = enc.GetString(body);
        Console.WriteLine("Message: " + message);
        // Add this message to the list of message to be acknowledged
        _range.add(m.Id);
        if (message.Equals("That's all, folks!"))
        {
            // Acknowledge all the received messages
            _session.messageAccept(_range);
            lock (_session)
            {
                Monitor.Pulse(_session);
            }
        }
    }
}

```

The listener creates a private queue to receive its messages and binds it to the fanout exchange:

```

string myQueue = session.Name;
session.queueDeclare(myQueue, Option.EXCLUSIVE, Option.AUTO_DELETE);
session.exchangeBind(myQueue, "amq.fanout", "my-key");

```

Now we create a listener and subscribe it to the queue:

```

lock (session)
{
    Console.WriteLine("Listening");
    // Create a listener and subscribe it to my queue.
    IMessageListener listener = new MessageListener(session);
    session.attachMessageListener(listener, myQueue);
    session.messageSubscribe(myQueue);
    // Receive messages until all messages are received
    Monitor.Wait(session);
}

```

Publishing Messages to the Fanout Exchange

The second program in this example, Producer.cs, writes messages to the fanout queue.

```

// Unlike topic exchanges and direct exchanges, a fanout
// exchange need not set a routing key.
IMessage message = new Message();
// Asynchronous transfer sends messages as quickly as
// possible without waiting for confirmation.
for (int i = 0; i < 10; i++)
{
    message.clearData();
    message.appendData(Encoding.UTF8.GetBytes("Message " + i));
    session.messageTransfer("amq.fanout", message);
}

// And send a synchronous final message to indicate termination.
message.clearData();
message.appendData(Encoding.UTF8.GetBytes("That's all, folks!"));
session.messageTransfer("amq.fanout", message);
session.sync();

```

Writing Publish/Subscribe Applications

This section describes two programs that implement Publish/Subscribe messaging using a topic exchange.

- Publisher.cs sends messages to the amq.topic exchange, using the multipart routing keys usa.news, usa.weather, europe.news, and europe.weather.
- Listener.cs creates private queues for news, weather, usa, and europe, binding them to the amq.topic exchange using bindings that match the corresponding parts of the multipart routing keys.

In this example, the publisher creates messages for topics like news, weather, and sports that happen in regions like Europe, Asia, or the United States. A given consumer may be interested in all weather messages, regardless of region, or it may be interested in news and weather for the United States, but uninterested in items for other regions. In this example, each consumer sets up its own private queues, which receive precisely the messages that particular consumer is interested in.

Running the Publish-Subscribe Examples

- 1) Make sure your PATH contains the directory <home>/qpidd/lib
- 2) Make sure that a qpid broker is running:

```
$ ps -eaf | grep qpidd
```

If a broker is running, you should see the qpidd process in the output of the above command.

- 3) In separate windows, start one or more topic subscribers as follows:

```
$ cd <home>/qpidd/examples/direct
```

With cygwin:

```
$ ./example-pub-sub-Listener.exe [hostname] [portnumber]
```

or with mono:

```
$ mono ./example-pub-sub-Listener.exe [hostname] [portnumber]
```

You will see output similar to this:

```
Listening for messages ...
Declaring queue: usa
Declaring queue: europe
Declaring queue: news
Declaring queue: weather
```

Each topic consumer creates a set of private queues, and binds each queue to the amq.topic exchange together with a binding that indicates which messages should be routed to the queue.

- 4) In another window, start the topic publisher, which publishes messages to the amq.topic exchange, as follows:

```
$ cd <home>/qpid/examples/direct
```

With cygwin:

```
$ ./example-pub-sub-Producer.exe [hostname] [portnumber]
```

or with mono:

```
$ mono ./example-pub-sub-Producer.exe [hostname] [portnumber]
```

This program has no output; the messages are routed to the message queues for each topic_consumer as specified by the bindings the consumer created.

- 5) Go back to the window for each topic consumer. You should see output like this:

```
Message: Message 0 from usa
Message: Message 0 from news
Message: Message 0 from weather
Message: Message 1 from usa
Message: Message 1 from news
Message: Message 2 from usa
Message: Message 2 from news
Message: Message 3 from usa
Message: Message 3 from news
Message: Message 4 from usa
Message: Message 4 from news
Message: Message 5 from usa
Message: Message 5 from news
Message: Message 6 from usa
Message: Message 6 from news
Message: Message 7 from usa
Message: Message 7 from news
Message: Message 8 from usa
Message: Message 8 from news
Message: Message 9 from usa
...
Message: That's all, folks! from weather
Shutting down listener for control
Message: That's all, folks! from europe
Shutting down listener for control
```

Now we will examine the code for each of these programs. In each section, we will discuss only the code that must be added to the skeleton shown in Section "Creating and Closing Sessions".

Publishing Messages to a Topic Exchange

The first program in the publish/subscribe example, Publisher.cs, defines two new functions: one that publishes messages to the topic exchange, and one that indicates that no more messages are coming.

The publishMessages function publishes a series of five messages using the specified routing key.

```

private static void publishMessages(ClientSession session, string routing_key)
{
    IMessage message = new Message();
    // Asynchronous transfer sends messages as quickly as
    // possible without waiting for confirmation.
    for (int i = 0; i < 10; i++)
    {
        message.clearData();
        message.appendData(Encoding.UTF8.GetBytes("Message " + i));
        session.messageTransfer("amq.topic", routing_key, message);
    }
}

```

The noMoreMessages function signals the end of messages using the control routing key, which is reserved for control messages.

```

private static void noMoreMessages(ClientSession session)
{
    IMessage message = new Message();
    // And send a synchronous final message to indicate termination.
    message.clearData();
    message.appendData(Encoding.UTF8.GetBytes("That's all, folks!"));
    session.messageTransfer("amq.topic", "control", message);
    session.sync();
}

```

In the main body of the program, messages are published using four different routing keys, and then the end of messages is indicated by a message sent to a separate routing key.

```

publishMessages(session, "usa.news");
publishMessages(session, "usa.weather");
publishMessages(session, "europe.news");
publishMessages(session, "europe.weather");

noMoreMessages(session);

```

Reading Messages from the Queue

The second program in the publish/subscribe example, Listener.cs, creates a local private queue, with a unique name, for each of the four binding keys it specifies: usa.#, europe.#, #.news, and #.weather, and creates a listener.

```

Console.WriteLine("Listening for messages ...");
// Create a listener
prepareQueue("usa", "usa.#", session);
prepareQueue("europe", "europe.#", session);
prepareQueue("news", "#.news", session);
prepareQueue("weather", "#.weather", session);

```

The prepareQueue() method creates a queue using a queue name and a routing key supplied as arguments it then attaches a listener with the session for the created queue and subscribe for this receiving messages from the queue:

```

// Create a unique queue name for this consumer by concatenating
// the queue name parameter with the Session ID.
Console.WriteLine("Declaring queue: " + queue);
session.queueDeclare(queue, Option.EXCLUSIVE, Option.AUTO_DELETE);

// Route messages to the new queue if they match the routing key.
// Also route any messages to with the "control" routing key to
// this queue so we know when it's time to stop. A publisher sends
// a message with the content "That's all, Folks!", using the
// "control" routing key, when it is finished.

session.exchangeBind(queue, "amq.topic", routing_key);
session.exchangeBind(queue, "amq.topic", "control");

// subscribe the listener to the queue
IMessageListener listener = new MessageListener(session);
session.attachMessageListener(listener, queue);
session.messageSubscribe(queue);

```

Writing Request/Response Applications

In the request/response examples, we write a server that accepts strings from clients and converts them to upper case, sending the result back to the requesting client. This example consists of two programs.

- Client.cs is a client application that sends messages to the server.
- Server.cs is a service that accepts messages, converts their content to upper case, and sends the result to the amq.direct exchange, using the request's reply-to property as the routing key for the response.

Running the Request/Response Examples

1) Make sure your PATH contains the directory <home>/qpid/lib

2) Make sure that a qpid broker is running:

```
$ ps -eaf | grep qpidd
```

If a broker is running, you should see the qpidd process in the output of the above command.

3) Run the server.

```
$ cd <home>/qpid/examples/direct
```

With cygwin:

```
$ ./example-request-response-Server.exe [hostname] [portnumber]
```

or with mono:

```
$ mono ./example-request-response-Server.exe [hostname] [portnumber]
```

You will see output similar to this:

Waiting for requests

4) In a separate window, start a client:

```
$ cd <home>/qpid/examples/direct
```

With cygwin:

```
$ ./example-request-response-Client.exe [hostname] [portnumber]
```

or with mono:

```
$ mono ./example-request-response-Client.exe [hostname] [portnumber]
```

You will see output similar to this:

```
Activating response queue listener for: clientSystem.Byte[]  
Waiting for all responses to arrive ...  
Response: TWAS BRILLIG, AND THE SLITHY TOVES  
Response: DID GIRE AND GYMBLE IN THE WABE.  
Response: ALL MIMSY WERE THE BOROGROVES,  
Response: AND THE MOME RATHS OUTGRABE.  
Shutting down listener for clientSystem.Byte[]  
Response: THAT'S ALL, FOLKS!
```

4) Go back to the server window, the output should be similar to this:

```
Waiting for requests  
Request: Twas brillig, and the slithy toves  
Request: Did gire and gymble in the wabe.  
Request: All mimsy were the borogroves,  
Request: And the mome raths outgrabe.  
Request: That's all, folks!
```

Now we will examine the code for each of these programs. In each section, we will discuss only the code that must be added to the skeleton shown in Section "Creating and Closing Sessions".

The Client Application

The first program in the request-response example, Client.cs, sets up a private response queue to receive responses from the server, then sends messages to the server, listening to the response queue for the server's responses.

```
string response_queue = "client" + session.getName();  
// Use the name of the response queue as the routing key  
session.queueDeclare(response_queue);  
session.exchangeBind(response_queue, "amq.direct", response_queue);  
  
// Create a listener for the response queue and listen for response messages.  
Console.WriteLine("Activating response queue listener for: " + response_queue);  
IMessageListener listener = new ClientMessageListener(session);  
session.attachMessageListener(listener, response_queue);  
session.messageSubscribe(response_queue);
```

Set some properties that will be used for all requests. The routing key for a request is `request`. The `reply-to` property is set to the routing key for the client's private queue.

```
IMessage request = new Message();  
request.DeliveryProperties.setRoutingKey("request");  
request.MessageProperties.setReplyTo(new ReplyTo("amq.direct", response_queue));
```

Now send some requests...

```

string[] strs = {
    "Twas brillig, and the slithy toves",
    "Did gire and gymble in the wabe.",
    "All mimsy were the borogroves",
    "And the mome raths outgrabe.",
    "That's all, folks!"
};
foreach (string s in strs)
{
    request.clearData();
    request.appendData(Encoding.UTF8.GetBytes(s));
    session.messageTransfer("amq.direct", request);
}

```

And wait for responses to arrive:

```

Console.WriteLine("Waiting for all responses to arrive ...");
Monitor.Wait(session);

```

The Server Application

The second program in the request-response example, Server.cs, uses the reply-to property as the routing key for responses.

The main body of Server.cs creates an exclusive queue for requests, then waits for messages to arrive.

```

const string request_queue = "request";
// Use the name of the request queue as the routing key
session.queueDeclare(request_queue);
session.exchangeBind(request_queue, "amq.direct", request_queue);

lock (session)
{
    // Create a listener and subscribe it to the request_queue
    IMessageListener listener = new MessageListener(session);
    session.attachMessageListener(listener, request_queue);
    session.messageSubscribe(request_queue);
    // Receive messages until all messages are received
    Console.WriteLine("Waiting for requests");
    Monitor.Wait(session);
}

```

The listener's messageTransfer() method converts the request's content to upper case, then sends a response to the broker, using the request's reply-to property as the routing key for the response.

```

BinaryReader reader = new BinaryReader(request.Body, Encoding.UTF8);
byte[] body = new byte[request.Body.Length - request.Body.Position];
reader.Read(body, 0, body.Length);
ASCIIEncoding enc = new ASCIIEncoding();
string message = enc.GetString(body);
Console.WriteLine("Request: " + message);

// Transform message content to upper case
string responseBody = message.ToUpper();

// Send it back to the user
response.clearData();
response.appendData(Encoding.UTF8.GetBytes(responseBody));
_session.messageTransfer("amq.direct", routingKey, response);

```

Excel AddIn

Excel AddIn

Qpid .net comes with Excel AddIns that are located in:

```
<project-root>\qpid\dotnet\client-010\addins
```

There are currently three projects:

ExcelAddIn: An RTD excel Addin

ExcelAddInProducer: A sample client to demonstrate the RTD AddIn

ExcelAddInMessageProcessor: A ample message processor for the RTD AddIn

Qpid RDT AddIn

Deploying the RTD AddIn

Excel provides a function called RTD (real-time data) that lets you specify a COM server via its ProgId here "Qpid" so that you can push qpid messages into Excel.

The provided RTD AddIn consumes messages from one queue and process them through a provided message processor.

For using the Qpid RTD follows those steps:

- 1) Copy the configuration Excel.exe.config into Drive\Program Files\Microsoft Office\Office12
- 2) Edit Excel.exe.xml and set the targeted Qpid broker host, port number, username and password.
- 3) Select the cell or cell range to contain the RTD information
- 4) Enter the following formula =rtd("Qpid","","myQueue") Where MyQueue is the queue from which you wish to receive messages from

Note: The Qpid RTD is a COM-AddIn that must be registered with Excel. This is done automatically when compiling the Addin with visual studio.

Defining a message processor

The default behavior of the RDT AddIn is to display the message payload. This could be altered by specifying your own message processor. A Message processor is a class that implements the API ExcelAddIn.MessageProcessor. For example, the provided processor in client-010\addins\ExcelAddInMessageProcessor displays the message body and the the header price when specified.

To use you own message processor follows those steps:

1) Write your own message processor that extends ExcelAddIn.MessageProcessor

2) Edit Excel.exe.config and uncomment the entries:

```
<add key="ProcessorAssembly"
value="\qpid\dotnet\client-010\addins\ExcelAddInMessageProcessor\bin\Debug\ExcelAddInMessageProcessor.dll"/>
<add key="ProcessorClass" value="ExcelAddInMessageProcessor.Processor"/>
```

- ProcessorAssembly is the path on the Assembly that contains your processor class
- ProcessorClass is your processor class name

3) run excel and define a rtd function

Note: the provided ExcelAddInProducer can be used for testing the provided message processor. As messages are sent to queue1 the following rtd function should be used =rtd("Qpid","queue1")

Qpid .Net How To

Collection of How Tos

- [Build .NET Client](#)

Build .NET Client

Prerequisites

[Setup environment](#)

Building 0.9 Client

Generate framing from /Qpid.Common/amqp.xml specification file:

```
$ build-framing
```

Alternatively, just switch to /Qpid.Common and run "ant" there.

You can build from Visual Studio 2005 normally. Alternatively, you can build debug releases for any supported framework from the command line using Nant:

To build .NET 2.0 executables (to bin/net-2.0):

```
$ build-dotnet20
```

To build .NET 1.1 executables (to bin/net-1.1):

```
$ build-dotnet11
```

To build for Mono on Linux (to bin/mono-2.0):

```
$ build-mono
```

Building 0.10 Client

Prerequisites:

Generate code from <project home>/dotnet/client-010/gentool:

```
$ cd <project home>/dotnet/client-010/gentool  
$ ant
```

You can build from Visual Studio 2005 normally. Alternatively, you can build debug releases for any supported framework from the command line using Nant:

To build .NET 2.0 executables (to bin/net-2.0):

```
$ cd <project home>/dotnet/client-010/  
$ nant
```

To build for Mono on Linux (to bin/mono-2.0):

```
$ cd <project home>/dotnet/client-010/  
$ nant -t:mono-2.0
```

Releasing

Releasing 0.10 Client

For .NET 2.0

```
$ cd <project home>/dotnet/client-010/  
$ nant release-pkg
```

Generates ./bin/net-2.0/release/Qpid.NET-net-2.0-yyyyMMdd.zip

For Mono

```
$ cd <project home>/dotnet/client-010/  
$ nant -t:mono-2.0 release-pkg
```

Generates ./bin/mono-2.0/release/Qpid.NET-mono-2.0-yyyyMMdd.zip

Run tests

Setup

- 1) Start an C++ 0.10 broker
- 2) Edit the file: <project home>\dotnet\client-010\test\test.config and set the host name and port number of your broker. If security is enabled you may need to change the value of username and password.

For .NET 2.0

```
$ cd <project home>/dotnet/client-010/  
$ nant test
```

For Mono on Linux

```
$ cd <project home>/dotnet/client-010/  
$ nant -t:mono-2.0 test
```

Setup .Net Client on Windows

Setup

Install:

- Microsoft Visual Studio 2005 (VS2005) or Mono
- NAnt - only required for builds outside VS2005 (.net 1.1, .net 2.0, mono 2.0)
- Ant
- Cygwin (or alternatively build via cmd but alter instructions below accordingly)

Set up PATH to include Nant.exe:

```
$ PATH=/cygdrive/c/WINDOWS/Microsoft.NET/Framework/v2.0.50727:$PATH
```

Set up PATH to include ant:

```
$ PATH=$ANT_HOME/bin:$PATH
```

WCF

Introduction

WCF (**Windows Communication Foundation**) unifies the .Net communication capabilities into a single, common, general Web service oriented framework. A good WCF tutorial can be found [here](#).

WCF separates how service logic is written from how services communicate with clients. Bindings are used to specify the transport, encoding, and protocol details required for clients and services to communicate with each other. Qpid provide a WCF binding: org.apache.qpid.wcf.model.QpidBinding. WCF Services that use the Qpid binding communicate through queues that are dynamically created on a Qpid broker.

How to use Qpid binding

WCF services are implemented using:

- A service contract with one or more operation contracts.
- A service implementation for those contracts.
- A configuration file to provide that implementation with an endpoint and a binding for that specific contract.

The following configuration file can be used to configure a Hello Service:

```

<configuration>
  <system.serviceModel>
    <services>
      <!-- the service class -->
      <service name="org.apache.qpid.wcf.demo.HelloService">
        <host>
          <baseAddresses>
            <!-- Use SOAP over AMQP -->
            <add baseAddress="soap.amqp:///"/>
          </baseAddresses>
        </host>

        <endpoint
          address="Hello"
          <!-- We use a Qpid Binding, see below def -->
          binding="customBinding"
          bindingConfiguration="QpidBinding"
          <!-- The service contract -->
          contract="org.apache.qpid.wcf.demo.IHelloContract"/>
      </service>
    </services>

    <bindings>
      <customBinding>
        <!-- cf def of the qpid binding -->
        <binding name="QpidBinding">
          <textMessageEncoding />
          <!-- specify the host and port number of the broker -->
          <QpidTransport
            host="192.168.1.14"
            port="5673" />
        </binding>
      </customBinding>
    </bindings>

    <extensions>
      <bindingElementExtensions>
        <!-- use Qpid binding element: org.apache.qpid.wcf.model.QpidTransportElement -->
        <add
          name="QpidTransport"
          type="org.apache.qpid.wcf.model.QpidTransportElement, qpidWCFModel"/>
      </bindingElementExtensions>
    </extensions>

  </system.serviceModel>
</configuration>

```

Endpoints and bindings can also be set within the service code:

```

/* set HostName, portNumber and MyService accordingly */
Binding binding = new QpidBinding("HostName", portNumber);
ServiceHost service = new ServiceHost(typeof(MyService), new Uri("soap.amqp:///"));
service.AddServiceEndpoint(typeof(IBooking), binding, "MyService");
service.Open();
...

```

Qpid 'C++' Documentation

Introduction

Contributors should read:

- README in subversion for build instructions.
- C++ coding tips
- C++ style guide
- C++ public API guidelines
- OS version considerations

- Auto tools guide

Testing guidelines:

- All classes should be unit tested with Boost.Test (Some tests are still using CppUnit, they will be converted.)
- Broker should pass PythonBrokerTest with

```
./run-tests -I cpp_ignore.tests
```

Currently built/tested with g++ on Linux using GNU make.

Design Notes

- C++ Broker Startup and Plugins
- C++ handler chaining
- Management Design notes
- Persistent Message Store Module

CppApiGuide

C++ public API guidelines.

These guidelines are for the public client API to be released in qpid 1.0. The "plugin" API exposed for bdbstore should eventually follow these guidelines but it can be deferred.

Public header files

Public headers under: qpid/cpp/src/include

Non-unit client tests should be built with -I include and *without* -I . so that missing public headers can be quickly identified.

Only src/include headers are installed with package qpidc-devel. Package qpidd-devel should only install src/include headers, but for 1.0 it may still install private headers.

PIMPL idiom

Value classes needed by the user (e.g. framing data types, message content) are exposed as normal classes in public headers.

Service classes (e.g. Session, Connection etc) use the pimpl idiom for compatibility isolation. See <http://en.wikipedia.org/wiki/Pimpl>.

Thread safety

I believe making the session thread safe will make it simpler to use in various circumstances (some of which may be unforeseen). I don't think it necessarily adds significant overhead (though this is something we can verify).

CppBrokerStartPlugins

The C++ broker automatically loads available plugins when it starts. It can get plugins from two places:

1. All loadable plugins from the directory specified with the --module-dir option. This option has a platform-specific default which is used unless the --no-module-dir is specified.
2. Individual plugins specified with the --load-module option

The broker will try to dynamically load shareable modules from the above locations. What actually registers the plugin with the Broker, though, is an action performed by the loaded module. Most (if not all) of the current plugins use a static instance of a class derived from qpid::Plugin. The object construction ends up calling the Plugin constructor which records the presence of the plugin.

There is no way to explicitly unload a plugin during execution. The scheme relies on process run-down unloading the dynamically loaded modules. However, there is a finalization procedure executed before the process runs down.

The C++ broker follows these basic steps at start time:

- Load plugins (static initializers set up basic objects)
- Broker calls qpid::Plugins::addOptions to add options from all plugins to the broker's option set. It does this by calling getOptions() on each known plugin instance. The qpid::Options-derived object is owned by the plugin and will be modified by option parsing.
- Parse the argc/argv and config file options against the known option sets.
- Call earlyInitialize(Target&) on each known plugin. Each plugin is expected to evaluate the type of the Target (by trying dynamic_cast on it) and only act on Target types it knows about. At this point the options have been parsed and are available.
- Initiate recovery
- Call initialize(Target&) on each known plugin.
- Run

CppEventChannelo

Event channel IO abstraction.

Goals: provide an IO abstraction layer that can be efficiently implemented using different techniques:

- select/poll/epoll
- aio_
- ec_ new linux event channel.
- shared mem, IPC etc.

The event channel is the central IO absctracton.

Async requests are posted to the channel as Events. When the request is complete it is returned from getEvent() with the data filled in.

We provide synchronous APIs to wrap post(event), wait for getEvent(). On posix these APIs are actually implemented using user-level context swithching so we get a simple programing model with minimal blocking and kernel context switching.

Note: this means that code before and after an apparently synchronous call "may execute in different threads". Don't use thread-local storage. The term "task" will denote the user-level execution context and we'll provide "task-locak storage" that is carried with the user context if we need it.

We can provide some simple in-process synchronization via the event channel to allow use level tasks to block on application events.

Core concepts:

EventChannel:

- Manages thread pool.
- Worker threads loop getting and processing events.
- Async requests: post request event, it will be processed when complete.
- Notification: Threads can block on a notification event to be woken when some other thread posts that event.

Task:

- like lightweight thread
- ucontext APIs for user-level context switch.

Linux ec_ + ucontext implementation:

- EventChannel is thin facade over native ec_ APIs.
- Tasks are scheduled onto threads.
- ideally our threads "never block" (but they can be preempted)
- when a thread hits a blocking point it suspends the current task and swaps to a ready task.
- when the suspended task is unblocked (e.g. async IO completes) it becomes ready and will be picked up by another thread.

Linux epoll + ucontext:

- Prove ucontext ideas.
- Use traditional polling inside EventChannel.

APR portable impl: only need client support - simple blocking socket calls.

Computing thread pool size:

- Initial size based on availabe CPU paralleism
- On linux /proc/cpuinfo? Any portable options.
- Thread pool grows automatically to avoid deadlock.

ThreadPool: Size should stay close to actual hardware parallellism + some delta due to pre-empted threads and thread-blocking synchronization calls required in the event channel implementation itself.

Questions:

- Does the thread pool need to shrink to reclaim resources?
- Is there a risk of unbounded growth? How to avoid without deadlock?

CppHandlerChains

The C++ broker uses handler chains to break complex processing into individual pieces.

- Each session has its own set of FrameHandler chains.
- Frames from the network are delivered to the first handler in the chain.
- Handlers do something with a frame, then pass it to the next handler.
- Handlers may "filter" frames by not passing some frames to the next handler.

Current status (2002/9/20)

Each chain starts with a SessionHandler. It handles L2 (session open, close etc.) and passes other frames to the next handler.

L3/L4 frames are handled by the SemanticHandler.

For clustering, a ClusterHandler is inserted at the start of the chain. It replicates frames to a backup broker so the backup can handle failover. It then passes frames on to the normal session handlers.

Approach for multi-frame segments

- The frame handler chain continues to handle individual frames.
- Each handler uses a FrameSet to accumulate its frames.
- Frame no longer contains a Body, moved to FrameSet

```
class FrameSet { // sketch
void add(const Frame& frame); // True if
const AMQMethodBody* getMethod(); // 0 means not complete.
const AMQHeaderBody* getHeader();
const AMQContentBody* getContent();
};
```

Note: All the visitor/dispatch classes using AMQFrame need to be reworked. Dispatch will always be based on an AMQMethodBody, not a frame.

Rationale:

"Frame rather than Segment/FrameSet handlers": Allows most flexibility to compose or not compose frames into segments & FrameSets. For example a cluster handler needs to replicate frame-by-frame, so we don't want to compose the full segment up front. Since the FrameSet class provides the composition logic, this is specified only once and easy to use in frame handlers.

"No Segment class": A segment by itself is not very useful. A non-content method is just a FrameSet containing a single method segment. For content bearing methods are a frameset with headers & content. There's little value for a stand alone segment class.

CppStyleGuide

Qpid C++ style guide

Qpid C++ follows this guide: <http://geosoft.no/development/cppstyle.html>

With the following amendments:

- Rule 34: Qpid source files have .h and .cpp extensions.
- Rule 71: Qpid uses 4 space indent, not 2. **No tabs**.
- Rule 75, 81: Qpid allows (but does not require) else/catch to be on same line as preceding }
- Rule 11: Qpid does *not* add an underscore to member variable names.

And the following additional rules:

- Rule q1: Unlike other blocks, the contents of namespace blocks are *not* indented to prevent excessive line splitting, and multiple namespaces may be opened or closed on a single line. For example:

```
namespace qpid { namespace common {

    class SomeClass {
        void foo();
    };

}} // namespace qpid::common
```

Debate and changes

The discussion on qpid-dev did raise debate about some points of the style guide. The exceptions and new rules above reflect:

- Points that were agreed on the list.
- De-facto style of the codebase as it stands today.

Anyone who feels strongly about further modifications to the style guide should:

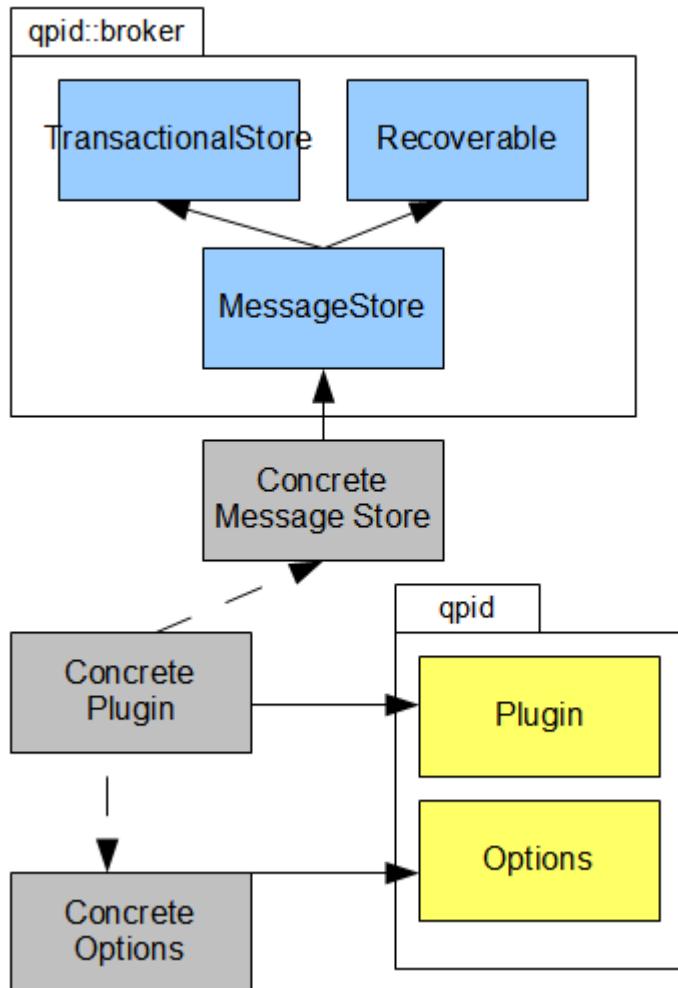
1. Raise the issue on qpid-dev.
2. Get consensus among the active qpid C++ developers for the change.
3. Reformat the entire qpid codebase to conform to the change.

If you are not willing to do step 3 then don't bother raising the issue.

Persistent Message Store Module

Persistent storage and recovery is provided by a plug-in module to the C++ broker. This page describes the relationship between the persistent message store module and the rest of the C++ broker.

A high-level view of the major classes involved in the store module plugin are shown below:



Concrete Plugin

When the plugin is loaded, the Concrete Plugin is instantiated. It is responsible for managing both the Concrete Options and the Concrete Message Store.

Concrete Options

The Concrete Options class defines and parses the options which are valid for the plugin. The particular options can vary by plugin.

Concrete Message Store

The Concrete Message Store implements the `qpid::broker::MessageStore` interface. The broker invokes methods on the `MessageStore` interface as operations requiring persistence take place during broker execution.

Note that the Concrete Message Store class can also inherit from Manageable to allow it to be managed via QMF. The particular message store should develop its own schema that makes sense.

PythonBrokerTest

Python Broker System Test Suite

This is a suite of python client tests that exercise and verify broker functionality. Python allows us to rapidly develop client test scenarios and provides a 'neutral' set of tests that can run against any AMQP-compliant broker.

The python/tests directory contains a collection of python modules, each containing several unittest classes, each containing a set of test methods that represent some test scenario. Test classes inherit `qpid.TestBase` from `qpid/testlib.py`, it inherits `unittest.TestCase` but adds some qpid-specific `setUp/tearDown` and convenience functions.

TODO: get pydoc generated up to qpid wiki or website automatically?

Running the tests

Simplest way to run the tests:

- Run a broker on the default port
- `./run_tests`

For additional options: `./run_tests --help`

Expected failures

Until we complete functionality, tests may fail because the tested functionality is missing in the broker. To skip expected failures in the C++ or Java brokers:

```
./run_tests -I cpp_failing.txt  
./run_tests -I java_failing.txt
```

If you fix a failure, please remove it from the corresponding list.

Qpid Integrations

AMQP integrations

- HermesJMS - The integration of Hermes JMS with Qpid
- Twisted AMQP - <https://launchpad.net/txampq>
- Apache Camel - <http://activemq.apache.org/camel/amqp.html>
- Apache Axis2 Java - <http://wso2.org/library/3663>
- Apache Axis2 C - http://ws.apache.org/axis2/c/docs/axis2c_manual.html#amqptrans
- Apache Synapse
- libvirt - <http://libvirt.org/> Provides QMF access to visualization
- Ovirt - <http://ovirt.org/> QMFC - used to build management tools
- Python web plugin - Plug-in for AMQP support from web browser
- Red Hat MRG - <http://redhat.com/mrg> Distro of Qpid with added management console and persistence
- Condor - <http://www.cs.wisc.edu/condor/> QMF Agents and AMQP job routing

Qpid Java Documentation

Purpose

This is the index of all Qpid Java Documentation.

Introduction

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

Upcoming features:

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

Useful Links

General User Guides

- FAQ
- Getting Started Guide
- Broker Environment Variables
- System Properties
- Troubleshooting Guide
- URL Formats for Qpid
- Example Classes
- AMQP Error Codes
- How Tos
 - Add New Users
 - Configure ACLs
 - Configure Broker and Client Heartbeating
 - Configure Java Qpid to use a SSL connection.
 - Configure Log4j CompositeRolling Appender
 - Configure Operational Status Logging
 - Configure the Broker via config.xml
 - Configure the Virtual Hosts via virtualhosts.xml
 - Debug using log4j
 - Firewall Configuration
 - How to Tune M3 Java Broker Performance
 - How to Use JNDI
 - Interact with a JMX MBean
 - Qpid Java Build How To
 - Split configuration files
 - Tune Broker and Client Memory Usage
 - Use Last Value Queues (LVQ)
 - Use Priority Queues
 - Use Producer Flow Control

Management Tools

- JConsole
- MessageStore Tool
- Qpid JMX Management Console
- Management Design notes

Developer Information

- Build How To
- Qpid Java Run Scripts
- Developer Pages
- Coding Standards
- AMQP Version Handling
- URL format for Connections and Binding
- Creating Java unit tests with InVM broker

Testing

Interoperability Testing

Performance Testing

- Sustained Tests
- IBM JMS Performance Test Results

Release Plans

- Release Plans

3rd Party Libraries

Qpid Persistence Options

There are currently two options for persistence in Qpid, as shown in the table below.

| Persistence Style | Provider | Advantages | Disadvantages |
|-------------------|-------------------------|--|--------------------------|
| In-Memory | Qpid MemoryMessageStore | Comes as part of the Qpid package | Not persistent |
| Derby DB Store | Qpid DerbyMessageStore | Allows persistence for larger messages/volumes | Limited testing reported |
| Berkeley DB Store | Berkeley project | Allows persistence for larger messages/volumes | Not Apache licensed |

Using In-Memory Persistence

Using In-Memory persistence is the default when you install Qpid and requires no additional install/configuration.

Using Derby Message Store

Simply use the following Store class:

```
<store>
  <class>org.apache.qpid.server.store.DerbyMessageStore</class>
</store>
```

Using Berkeley DB Persistence

Install Berkeley DB

If you choose to use the Berkeley DB solution for scalability purposes then you should download & install version 3.1 from <http://www.oracle.com/technology/software/products/berkeley-db/je/index.html>

Amend your Qpid configuration to switch BDB on

The default Qpid configuration file can be found in the etc directory of your install and is named config.xml.

To use BDB, simply add the following element:

```
<store>
  <class>org.apache.qpid.server.store.berkeleydb.BDBMessageStore</class>
</store>
```

alternatively an example file is provided named persistent_config.xml

Install the Qpid bridge modules for Berkeley DB

You can either build the module from source which is available from the [JBoss Site](#).

However, as a temporary measure, you can use the bridging modules from this page [M1-BDBStore](#) or [M2-BDBStore](#). You should then ensure that this jar is included in the classpath for the broker (see more info below), along with the BDB jar (je-<version>.jar).

This can simply be done by editing your classpath to add the two jars that you need and then pass an option into qpid-server to use your classpath.

So, first set your classpath to something like this:

```
CLASSPATH=$QPID_HOME/lib/qpid-incubating.jar:$QPID_HOME/lib/bdbstore.jar:$QPID_HOME/lib/je-<version>.
```

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

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

You can check the classpath being used by adding an additional option to output the classpath in use:

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

alternatively you can edit the QPID_LIBS variable in the qpid-server script.

We hope to be able to integrate these modules into our Apache project shortly - but pending a discussion about the appropriate way to handle this process.

3rd Party Tools

Using Qpid Java with 3rd Party Tools

- Mule

Mule

Connection configuration options

There are many ways of configuring Mule and many options that can be set. The following XML snippet shows two options for setting your connection configuration details. One uses the [Connection URL](#) the other uses direct properties to set the connection details. Note: that the URL format allows for more freedom of configuration while the direct property approach will pickup the defaults for most values.

One reason you may wish to use the URL format is to specify a 'nofailover' mechanism so that you can better control via mule what occurs when the connection is lost.

See the [Connection URL Format](#) page for more details on setting the failover mechanism.

```
<connector name="jmsConnector1" className="org.mule.providers.jms.JmsConnector">
    <properties>
        <property name="connectionFactoryClass" value=
"org.apache.qpid.client.AMQConnectionFactory"/>

        ...
        <!-- Property entry for using a URL -->
        <map name="connectionFactoryProperties">
            <property name="connectionURLString" value="amqp:
//guest:guest@clientID/test?brokerlist='tcp://localhost:5672'"/>
        </map>

        <!-- Property entry for using individual values -->
        <!-- Note: that you can only set the following items -->
        <!-- All other values will be defaulted. -->
        <!--map name="connectionFactoryProperties">
            <property name="virtualPath" value="/test"/>
            <property name="host" value="localhost"/>
            <property name="port" value="5672"/>
            <property name="defaultUsername" value="guest"/>
            <property name="defaultPassword" value="guest"/>
        </map-->

    </properties>
</connector>
```

AMQP Error Codes

The Java server will return errors to the client under certain circumstances. These error codes are defined in the [AMQP Spec](#).

Common error codes include:

| Code | Name | Reason |
|------|------------------------------|---|
| 312 | No Route | The message is being sent to a destination that does not exist. |
| 313 | No Consumers | The message is marked as immediate delivery, but no consumers are able to receive the message at this time. |
| 403 | Access Refused | Implies that you've been <i>refused access</i> .
Note that incorrect user/password credentials will result in a 530 error. |
| 404 | Not Found | The client attempted an operation on an entity which does not exist. |
| 405 | Already Exists | The client attempted to create an entity which already exists. |
| 406 | In Use | The client attempted to delete an entity which is currently being used. |
| 407 | Invalid Routing Key | The client attempted to use an invalid routing key. |
| 408 | Request Timeout | The requested operation could not be completed in time. |
| 409 | Invalid Argument | The client provided an argument which the server did not recognise, eg. invalid JMS selector. |
| 530 | Not Allowed | The client attempted an operation which it does not have permission for.
Failure to supply correct user/password credentials will result in a 530. |
| 542 | Unsupported Protocol Version | The server does not support the requested AMQP version. |

Example Classes

Overview

Some example client code is provided to help you get started, using JNDI and JMS.

Downloading the source

The source code for the example client classes being provided is contained in the example package which can be found in:

<https://svn.apache.org/repos/asf/qpid/trunk/qpid/java/client/example/>

Building the Example source

You can build the example source using Maven by simply typing 'mvn' in the example directory.

Notes on the source package

Note that if you wish to build the example client code from within your IDE you will need to add the following jars contained in the Qpid binary release into your classpath:

```
qpid-client-<MILESTONE>.jar
qpid-common-<MILESTONE>.jar
geronimo-jms_1.1_spec-1.0.jar
```

At the time of writing MILESTONE is 'M4'.

NB: At runtime, you'll need to put all the jars in the client/lib dir into your classpath along with these jars.

The example packages classes currently use log4j for logging. You may instead wish to amend the logging and exception handling in line with your project standards.

Class Overview

Essentially we have provided example client classes to act as a publisher and a subscriber for handling messages using AMQ, as far as possible via JMS.

Contributions to the example classes are most welcome. Please send your classes to dev@qpid.apache.org.

Basic Classes

This section outlines the classes that you're likely to find most useful.

Publisher

This class contains the methods for publishing messages to a queue, using the example.properties file to populate the initial context and

provide an example queue and topic.

FileMessageDispatcher

This class provides a simple wrapper around the publisher for dispatching messages from file input, providing a really easy way to get started publishing messages. So, first create some files you'd like to use as payload for your test messages.

Then, you simply pass one argument to the FileMessageDispatcher class which is the path to your test message files. The dispatcher will then create a message from each file in the directory. You can opt to use the path to a single file as an alternative, in the same way.

Subscriber

This class acts as a consumer of messages sent to a queue.

Helper Classes/Exceptions

This section gives some additional information about helper classes in the example packages.

MessageFactory

This class constructs a message with payload from an input file provided and sets a property on the message using the filename.

MessageFactoryException

Exception thrown if a message cannot be created in the factory class.

Examples Using AMQP Immediate Delivery Feature

MonitorMessageDispatcher

This is an additional wrapper class to be run independently as the monitor which allows you to provide a heartbeat (set to 20 seconds intervals but can be changed at will) being consumed by the example subscriber. When we detect that the subscriber has stopped, you can opt to make a call to an application specific 'recovery process' by amending the exception handling in the main method when a UndeliveredMessageException is caught (marked by a TODO), or simply exit gracefully if appropriate.

UndeliveredMessageException

Thrown when the subscriber is not there to read the monitor traffic which has been marked as for immediate delivery.

MonitoredSubscriber

Subclass of the Subscriber which also reads messages from the monitor queue to let us know that it's consuming ok.

The **shared** package contains utility classes for file manipulation etc and constants for general use (to reduce the maintenance overhead) which could perhaps be replaced with config properties as appropriate.

Getting Started

To get started with Apache Qpid, follow the steps below.

1. Download the software [Download](#)
2. Start a broker.
 - Instructions for [running a Qpid Java broker \(AMQP 0-8, 0-9\)](#)
 - Instructions for [running a Qpid C++ broker \(AMQP 0-10\)](#)
 - [Management tools](#) (AMQP 0-10, works with the Qpid C++ broker)
3. Run an example program from the downloaded software, or from the following URLs (these are svn URLs, which you can use to browse the examples or check them out):
 - C++ (AMQP 0-10):
 - Examples: <https://svn.apache.org/repos/asf/qpid/trunk/qpid/cpp/examples/>
 - [Running the C++ Examples](#)
 - Java JMS (AMQP 0-10):
 - Examples: <https://svn.apache.org/repos/asf/qpid/trunk/qpid/java/client/example/>
 - [Script for Running the Java JMS Examples](#)
 - Python (AMQP 0-10):
 - Examples: <https://svn.apache.org/repos/asf/qpid/trunk/qpid/python/examples/>
 - [Running the Python Examples](#)
 - Ruby (AMQP 0-10):
 - Examples: <https://svn.apache.org/repos/asf/qpid/trunk/qpid/ruby/examples/>
 - .NET (AMQP 0-10):
 - Examples: <http://svn.apache.org/viewvc/qpid/trunk/qpid/dotnet/client-010/examples/>
 - [.NET Tutorial](#)
4. Read the API Guides and Documentation
 - [C++ Client API \(AMQP 0-10\)](#)
 - [JNDI Configuration for Java JMS](#)
 - [Python Client API \(AMQP 0-10\)](#)
 - [Documentation](#)
5. Get your Questions Answered
 - [Read the FAQ](#)
 - Ask a question on the user list users-subscribe@qpid.apache.org

MgmtC++

Managing the C++ Broker

There are quite a few ways to interact with the C++ broker. The command line tools include:

- qpid-route - used to configure federation (a set of federated brokers)
- qpid-config - used to configure queues, exchanges, bindings and list them etc
- qpid-tool - used to view management information/statistics and call any management actions on the broker
- qpid-printhevents - used to receive and print QMF events

Using *qpid-config*

This utility can be used to create queues exchanges and bindings, both durable and transient. Always check for latest options by running --help command.

```
$ qpid-config --help
Usage: qpid-config [OPTIONS]
    qpid-config [OPTIONS] exchanges [filter-string]
    qpid-config [OPTIONS] queues   [filter-string]
    qpid-config [OPTIONS] add exchange <type> <name> [AddExchangeOptions]
    qpid-config [OPTIONS] del exchange <name>
    qpid-config [OPTIONS] add queue <name> [AddQueueOptions]
    qpid-config [OPTIONS] del queue <name>
    qpid-config [OPTIONS] bind   <exchange-name> <queue-name> [binding-key]
    qpid-config [OPTIONS] unbind <exchange-name> <queue-name> [binding-key]

Options:
    -b [ --bindings ]                      Show bindings in queue or exchange list
    -a [ --broker-addr ] Address (localhost) Address of qpid broker
        broker-addr is in the form:  [username/password@] hostname | ip-address [:<port>]
        ex: localhost, 10.1.1.7:10000, broker-host:10000, guest/guest@localhost

Add Queue Options:
    --durable           Queue is durable
    --cluster-durable  Queue becomes durable if there is only one functioning cluster node
    --file-count N (8)  Number of files in queue's persistence journal
    --file-size N (24)  File size in pages (64Kib/page)
    --max-queue-size N Maximum in-memory queue size as bytes
    --max-queue-count N Maximum in-memory queue size as a number of messages
    --limit-policy [none | reject | flow-to-disk | ring | ring-strict]
        Action taken when queue limit is reached:
            none (default) - Use broker's default policy
            reject          - Reject enqueued messages
            flow-to-disk    - Page messages to disk
            ring             - Replace oldest unacquired message with new
            ring-strict      - Replace oldest message, reject if oldest is acquired
    --order [fifo | lvq | lvq-no-browse]
        Set queue ordering policy:
            fifo (default) - First in, first out
            lvq             - Last Value Queue ordering, allows queue browsing
            lvq-no-browse   - Last Value Queue ordering, browsing clients may lose
data
    --generate-queue-events N
        If set to 1, every enqueue will generate an event that can be processed
by
        registered listeners (e.g. for replication). If set to 2, events will be
generated for enqueues and dequeues

Add Exchange Options:
    --durable     Exchange is durable
    --sequence   Exchange will insert a 'qpid.msg_sequence' field in the message header
                 with a value that increments for each message forwarded.
    --ive        Exchange will behave as an 'initial-value-exchange', keeping a reference
                 to the last message forwarded and enqueueing that message to newly bound
                 queues.
```

Get the summary page

```
$ qpid-config
Total Exchanges: 6
    topic: 2
    headers: 1
    fanout: 1
    direct: 2
Total Queues: 7
    durable: 0
    non-durable: 7
```

List the queues

```
$ qpid-config queues
Queue Name          Attributes
=====
pub_start
pub_done
sub_ready
sub_done
perftest0          --durable
reply-dhcp-100-18-254.bos.redhat.com.20713 auto-del excl
topic-dhcp-100-18-254.bos.redhat.com.20713 auto-del excl
```

List the exchanges with bindings

```
$ ./qpid-config -b exchanges
Exchange '' (direct)
  bind pub_start => pub_start
  bind pub_done => pub_done
  bind sub_ready => sub_ready
  bind sub_done => sub_done
  bind perftest0 => perftest0
  bind mgmt-3206ff16-fb29-4a30-82ea-e76f50dd7d15 => mgmt-3206ff16-fb29-4a30-82ea-e76f50dd7d15
  bind repl-3206ff16-fb29-4a30-82ea-e76f50dd7d15 => repl-3206ff16-fb29-4a30-82ea-e76f50dd7d15
Exchange 'amq.direct' (direct)
  bind repl-3206ff16-fb29-4a30-82ea-e76f50dd7d15 => repl-3206ff16-fb29-4a30-82ea-e76f50dd7d15
  bind repl-df06c7a6-4ce7-426a-9f66-da91a2a6a837 => repl-df06c7a6-4ce7-426a-9f66-da91a2a6a837
  bind repl-c55915c2-2fda-43ee-9410-b1c1cbb3e4ae => repl-c55915c2-2fda-43ee-9410-b1c1cbb3e4ae
Exchange 'amq.topic' (topic)
Exchange 'amq.fanout' (fanout)
Exchange 'amq.match' (headers)
Exchange 'qpid.management' (topic)
  bind mgmt.# => mgmt-3206ff16-fb29-4a30-82ea-e76f50dd7d15
```

Using qpid-route

This utility is to create federated networks of brokers, This allows you for forward messages between brokers in a network. Messages can be routed statically (using "qpid-route route add") where the bindings that control message forwarding are supplied in the route. Message routing can also be dynamic (using "qpid-route dynamic add") where the messages are automatically forwarded to clients based on their bindings to the local broker.

```

$ qpid-route
Usage: qpid-route [OPTIONS] dynamic add <dest-broker> <src-broker> <exchange> [tag]
[exclude-list]
    qpid-route [OPTIONS] dynamic del <dest-broker> <src-broker> <exchange>

        qpid-route [OPTIONS] route add    <dest-broker> <src-broker> <exchange> <routing-key> [tag]
[exclude-list]
        qpid-route [OPTIONS] route del    <dest-broker> <src-broker> <exchange> <routing-key>
        qpid-route [OPTIONS] queue add    <dest-broker> <src-broker> <exchange> <queue>
        qpid-route [OPTIONS] queue del    <dest-broker> <src-broker> <exchange> <queue>
        qpid-route [OPTIONS] route list  [<dest-broker>]
        qpid-route [OPTIONS] route flush [<dest-broker>]
        qpid-route [OPTIONS] route map   [<broker>]

        qpid-route [OPTIONS] link add   <dest-broker> <src-broker>
        qpid-route [OPTIONS] link del   <dest-broker> <src-broker>
        qpid-route [OPTIONS] link list  [<dest-broker>]

Options:
-v [ --verbose ]           Verbose output
-q [ --quiet ]             Quiet output, don't print duplicate warnings
-d [ --durable ]           Added configuration shall be durable
-e [ --del-empty-link ]    Delete link after deleting last route on the link
-s [ --src-local ]          Make connection to source broker (push route)
-t <transport> [ --transport <transport> ]
                           Specify transport to use for links, defaults to tcp

dest-broker and src-broker are in the form: [username/password@] hostname | ip-address
[:<port>]
ex: localhost, 10.1.1.7:10000, broker-host:10000, guest/guest@localhost

```

A few examples:

```

qpid-route dynamic add host1 host2 fed.topic
qpid-route dynamic add host2 host1 fed.topic

qpid-route -v route add host1 host2 hub1.topic hub2.topic.stock.buy
qpid-route -v route add host1 host2 hub1.topic hub2.topic.stock.sell
qpid-route -v route add host1 host2 hub1.topic 'hub2.topic.stock.#'
qpid-route -v route add host1 host2 hub1.topic 'hub2.#'
qpid-route -v route add host1 host2 hub1.topic 'hub2.topic.#'
qpid-route -v route add host1 host2 hub1.topic 'hub2.global.#'

```

The link map feature can be used to display the entire federated network configuration by supplying a single broker as an entry point:

```

$ qpid-route route map localhost:10001

Finding Linked Brokers:
localhost:10001... Ok
localhost:10002... Ok
localhost:10003... Ok
localhost:10004... Ok
localhost:10005... Ok
localhost:10006... Ok
localhost:10007... Ok
localhost:10008... Ok

Dynamic Routes:

Exchange fed.topic:
localhost:10002 <=> localhost:10001
localhost:10003 <=> localhost:10002
localhost:10004 <=> localhost:10002
localhost:10005 <=> localhost:10002
localhost:10006 <=> localhost:10005
localhost:10007 <=> localhost:10006
localhost:10008 <=> localhost:10006

Exchange fed.direct:
localhost:10002 => localhost:10001
localhost:10004 => localhost:10003
localhost:10003 => localhost:10002
localhost:10001 => localhost:10004

Static Routes:

localhost:10003(ex=amq.direct) <= localhost:10005(ex=amq.direct) key=rkey
localhost:10003(ex=amq.direct) <= localhost:10005(ex=amq.direct) key=rkey

```

Using qpid-tool

This utility provided a telnet style interface to be able to view, list all stats and action all the methods. Simple capture below. Best to just play with it and mail the list if you have questions or want features added.

```

qpид:
qpид: help
Management Tool for QPID
Commands:
  list                                     - Print summary of existing objects by class
  list <className>                         - Print list of objects of the specified class
  list <className> all                      - Print contents of all objects of specified class
  list <className> active                  - Print contents of all non-deleted objects of specified class
  list <list-of-IDs>                      - Print contents of one or more objects (infer className)
  list <className> <list-of-IDs>          - Print contents of one or more objects
    list is space-separated, ranges may be specified (i.e. 1004-1010)
  call <ID> <methodName> <args>           - Invoke a method on an object
  schema                                    - Print summary of object classes seen on the target
  schema <className>                      - Print details of an object class
  set time-format short                   - Select short timestamp format (default)
  set time-format long                    - Select long timestamp format
  quit or ^D                            - Exit the program
qpид: list
Management Object Types:
ObjectType      Active   Deleted
=====
qpид.binding    21       0
qpид.broker     1        0
qpид.client     1        0
qpид.exchange   6        0
qpид.queue      13       0
qpид.session    4        0
qpид.system     1        0
qpид.vhost      1        0

```

```

qpidd: list qpid.system
Objects of type qpid.system
  ID      Created      Destroyed   Index
  =====
  1000  21:00:02 -          host

qpidd: list 1000
Object of type qpid.system: (last sample time: 21:26:02)
  Type     Element    1000
  =====
  config  sysId      host
  config  osName     Linux
  config  nodeName   localhost.localdomain
  config  release    2.6.24.4-64.fc8
  config  version    #1 SMP Sat Mar 29 09:15:49 EDT 2008
  config  machine   x86_64

qpidd: schema queue
Schema for class 'qpid.queue':
  Element      Type      Unit      Access      Notes      Description
  =====
  vhostRef      reference      ReadCreate      index
  name          short-string      ReadCreate      index
  durable        boolean      ReadCreate
  autoDelete    boolean      ReadCreate
  exclusive     boolean      ReadCreate
  arguments      field-table      ReadOnly      Arguments supplied in
queue.declare
  storeRef      reference      ReadOnly      Reference to persistent
queue (if durable)
  msgTotalEnqueues  uint64      message      Total messages enqueued
  msgTotalDequeues  uint64      message      Total messages dequeued
  msgTxnEnqueues   uint64      message      Transactional messages
enqueued
  msgTxnDequeues   uint64      message      Transactional messages
dequeued
  msgPersistEnqueues  uint64      message      Persistent messages
enqueued
  msgPersistDequeues  uint64      message      Persistent messages
dequeued
  msgDepth       uint32      message      Current size of queue in
messages
  msgDepthHigh   uint32      message      Current size of queue in
messages (High)
  msgDepthLow    uint32      message      Current size of queue in
messages (Low)
  byteTotalEnqueues  uint64      octet      Total messages enqueued
  byteTotalDequeues  uint64      octet      Total messages dequeued
  byteTxnEnqueues   uint64      octet      Transactional messages
enqueued
  byteTxnDequeues   uint64      octet      Transactional messages
dequeued
  bytePersistEnqueues  uint64      octet      Persistent messages
enqueued
  bytePersistDequeues  uint64      octet      Persistent messages
dequeued
  byteDepth       uint32      octet      Current size of queue in
bytes
  byteDepthHigh   uint32      octet      Current size of queue in
bytes (High)
  byteDepthLow    uint32      octet      Current size of queue in
bytes (Low)
  enqueueTxnStarts  uint64      transaction      Total enqueue
transactions started
  enqueueTxnCommits  uint64      transaction      Total enqueue
transactions committed
  enqueueTxnRejects  uint64      transaction      Total enqueue
transactions rejected
  enqueueTxnCount   uint32      transaction      Current pending enqueue
transactions
  enqueueTxnCountHigh  uint32      transaction      Current pending enqueue
transactions (High)
  enqueueTxnCountLow  uint32      transaction      Current pending enqueue
transactions (Low)
  dequeueTxnStarts  uint64      transaction      Total dequeue
transactions started

```

| dequeueTxnCommits | uint64 | transaction | Total dequeue |
|---|------------|---|--|
| transactions committed | | | |
| dequeueTxnRejects | uint64 | transaction | Total dequeue |
| transactions rejected | | | |
| dequeueTxnCount | uint32 | transaction | Current pending dequeue |
| transactions | | | |
| dequeueTxnCountHigh | uint32 | transaction | Current pending dequeue |
| transactions (High) | | | |
| dequeueTxnCountLow | uint32 | transaction | Current pending dequeue |
| transactions (Low) | | | |
| consumers | uint32 | consumer | Current consumers on queue |
| queue | | | |
| consumersHigh | uint32 | consumer | Current consumers on queue (High) |
| queue (High) | | | |
| consumersLow | uint32 | consumer | Current consumers on queue (Low) |
| bindings | uint32 | binding | Current bindings |
| bindingsHigh | uint32 | binding | Current bindings (High) |
| bindingsLow | uint32 | binding | Current bindings (Low) |
| unackedMessages | uint32 | message | Messages consumed but not yet acked |
| not yet acked | | | |
| unackedMessagesHigh | uint32 | message | Messages consumed but not yet acked (High) |
| not yet acked (High) | | | |
| unackedMessagesLow | uint32 | message | Messages consumed but not yet acked (Low) |
| not yet acked (Low) | | | |
| messageLatencySamples | delta-time | nanosecond | Broker latency through this queue (Samples) |
| messageLatencyMin | delta-time | nanosecond | Broker latency through this queue (Min) |
| messageLatencyMax | delta-time | nanosecond | Broker latency through this queue (Max) |
| messageLatencyAverage | delta-time | nanosecond | Broker latency through this queue (Average) |
| Method 'purge' Discard all messages on queue | | | |
| qpidd: list queue | | | |
| Objects of type qpid.queue | | | |
| ID | Created | Destroyed | Index |
| 1012 | 21:08:13 | - | 1002.pub_start |
| 1014 | 21:08:13 | - | 1002.pub_done |
| 1016 | 21:08:13 | - | 1002.sub_ready |
| 1018 | 21:08:13 | - | 1002.sub_done |
| 1020 | 21:08:13 | - | 1002.perftest0 |
| 1038 | 21:09:08 | - | 1002.mgmt-3206ff16-fb29-4a30-82ea-e76f50dd7d15 |
| 1040 | 21:09:08 | - | 1002.repl-3206ff16-fb29-4a30-82ea-e76f50dd7d15 |
| 1046 | 21:09:32 | - | 1002.mgmt-df06c7a6-4ce7-426a-9f66-da91a2a6a837 |
| 1048 | 21:09:32 | - | 1002.repl-df06c7a6-4ce7-426a-9f66-da91a2a6a837 |
| 1054 | 21:10:01 | - | 1002.mgmt-c55915c2-2fda-43ee-9410-b1c1cbb3e4ae |
| 1056 | 21:10:01 | - | 1002.repl-c55915c2-2fda-43ee-9410-b1c1cbb3e4ae |
| 1063 | 21:26:00 | - | 1002.mgmt-8d621997-6356-48c3-acab-76a37081d0f3 |
| 1065 | 21:26:00 | - | 1002.repl-8d621997-6356-48c3-acab-76a37081d0f3 |
| qpidd: list 1020 | | | |
| Object of type qpid.queue: (last sample time: 21:26:02) | | | |
| Type | Element | 1020 | |
| config vhostRef | | 1002 | |
| config name | | perftest0 | |
| config durable | | False | |
| config autoDelete | | False | |
| config exclusive | | False | |
| config arguments | | {'qpid.max_size': 0, 'qpid.max_count': 0} | |
| config storeRef | | NULL | |
| inst msgTotalEnqueues | | 500000 messages | |
| inst msgTotalDequeues | | 500000 | |
| inst msgTxnEnqueues | | 0 | |
| inst msgTxnDequeues | | 0 | |
| inst msgPersistEnqueues | | 0 | |
| inst msgPersistDequeues | | 0 | |
| inst msgDepth | | 0 | |
| inst msgDepthHigh | | 0 | |
| inst msgDepthLow | | 0 | |
| inst byteTotalEnqueues | | 512000000 octets | |
| inst byteTotalDequeues | | 512000000 | |
| inst byteTxnEnqueues | | 0 | |
| inst byteTxnDequeues | | 0 | |

```
inst bytePersistEnqueues 0
inst bytePersistDequeues 0
inst byteDepth 0
inst byteDepthHigh 0
inst byteDepthLow 0
inst enqueueTxnStarts 0 transactions
inst enqueueTxnCommits 0
inst enqueueTxnRejects 0
inst enqueueTxnCount 0
inst enqueueTxnCountHigh 0
inst enqueueTxnCountLow 0
inst dequeueTxnStarts 0
inst dequeueTxnCommits 0
inst dequeueTxnRejects 0
inst dequeueTxnCount 0
inst dequeueTxnCountHigh 0
inst dequeueTxnCountLow 0
inst consumers 0 consumers
inst consumersHigh 0
inst consumersLow 0
inst bindings 1 binding
inst bindingsHigh 1
inst bindingsLow 1
inst unackedMessages 0 messages
inst unackedMessagesHigh 0
inst unackedMessagesLow 0
inst messageLatencySamples 0
inst messageLatencyMin 0
inst messageLatencyMax 0
```

```
inst      messageLatencyAverage  0
qpidd:
```

Using qpid-printevents

This utility connects to one or more brokers and collects events, printing out a line per event.

```
$ qpid-printevents --help
Usage: qpid-printevents [options] [broker-addr]...

Collect and print events from one or more Qpid message brokers. If no broker-
addr is supplied, qpid-printevents will connect to 'localhost:5672'. broker-
addr is of the form: [username/password@] hostname | ip-address [:<port>] ex:
localhost, 10.1.1.7:10000, broker-host:10000, guest/guest@localhost

Options:
-h, --help  show this help message and exit
```

You get the idea... have fun!

RAJB

General User Guides Java

- [FAQ](#)
- [Getting Started Guide](#)
- [Troubleshooting Guide](#)
- [How To](#)
- [URL Formats for Qpid](#)
- [Example Classes](#)

RASC

Building the C++ Broker and Client Libraries

The root directory for the C++ distribution is named `qpidc-0.4`. The README file in that directory gives instructions for building the broker and client libraries. In most cases you will do the following:

```
[qpidc-0.4]$ ./configure}
[qpidc-0.4]$ make
```

Running the C++ Broker

Once you have built the broker and client libraries, you can start the broker from the command line:

```
[qpidc-0.4]$ src/qpidd
```

Use the `--daemon` option to run the broker as a daemon process:

```
[qpidc-0.4]$ src/qpidd --daemon
```

You can stop a running daemon with the `--quit` option:

```
[qpidc-0.4]$ src/qpidd --quit
```

You can see all available options with the `--help` option

```
[qpidc-0.4]$ src/qpidd --help
```

Most common questions getting qpidd running

Error when starting broker: "no data directory"

The qpid broker requires you to set a data directory or specify --no-data-dir (see help for more details). The data directory is used for the journal, so it is important when reliability counts. Make sure your process has write permission to the data directory.

The default location is

```
/lib/var/qpidd
```

An alternate location can be set with --data-dir

Error when starting broker: "that process is locked"

Note that when qpid starts it creates a lock file in data directory if data directory are being used. If you have an un-controlled exit, please mail the trace from the core to the dev@qpid.apache.org mailing list. To clear the lock run

```
./qpidd -q
```

It should also be noted that multiple brokers can be run on the same host. To do so set alternate data directories for each qpid instance.

Using a configuration file

Each option that can be specified on the command line can also be specified in a configuration file. To see available options, use --help on the command line:

```
./qpidd --help
```

A configuration file uses name/value pairs, one on each line. To convert a command line option to a configuration file entry:

- a.) remove the '--' from the beginning of the option.
- b.) place a '=' between the option and the value (use yes or true to enable options that take no value when specified on the command line).
- c.) place one option per line.

For instance, the --daemon option takes no value, the --log-to-syslog option takes the values yes or no. The following configuration file sets these two options:

```
daemon=yes  
log-to-syslog=yes
```

Can I use any Language client with the C++ Broker?

Yes, all the clients work with the C++ broker; it is written in C+ but uses the AMQP wire protocol. Any broker can be used with any client that uses the same AMQP version. When running the C+ broker, it is highly recommended to run AMQP 0-10.

Note that JMS also works with the C++ broker. For more details on using the Java client refer to these pages:

- How to Use JNDI
- URL Formats for Qpid
- Example Classes

Authentication

Linux

The PLAIN authentication is done on a username+password, which is stored in the sasldb_path file. Usernames and passwords can be added to the file using the command:

```
saslpasswd2 -f /var/lib/qpidd/qpidd.sasldb -u <REALM> <USER>
```

The REALM is important and should be the same as the --auth-realm option to the broker. This lets the broker properly find the user in the sasldb file.

Existing user accounts may be listed with:

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

NOTE: The sasldb file must be readable by the user running the qpid daemon, and should be readable only by that user.

Windows

On Windows, the users are authenticated against the local machine. You should add the appropriate users using the standard Windows tools (Control Panel->User Accounts). To run many of the examples, you will need to create a user "guest" with password "guest".

If you cannot or do not want to create new users, you can run without authentication by specifying the no-auth option to the broker.

Slightly more complex configuration

The easiest way to get a full listing of the broker's options are to use the --help command, run it locally for the latest set of options. These options can then be set in the conf file for convenience (see above)

```
./qpidd --help

Usage: qpidd OPTIONS
Options:
-h [ --help ]                      Displays the help message
-v [ --version ]                    Displays version information
--config FILE (/etc/qpidd.conf)     Reads configuration from FILE

Module options:
--module-dir DIR (/usr/lib/qpidd)   Load all .so modules in this directory
--load-module FILE                 Specifies additional module(s) to be loaded
--no-module-dir                   Don't load modules from module directory

Broker Options:
--data-dir DIR (/var/lib/qpidd)    Directory to contain persistent data generated by the broker
--no-data-dir                      Don't use a data directory. No persistent
                                    configuration will be loaded or stored
-p [ --port ] PORT (5672)          Tells the broker to listen on PORT
--worker-threads N (3)             Sets the broker thread pool size
--max-connections N (500)           Sets the maximum allowed connections
--connection-backlog N (10)         Sets the connection backlog limit for the
                                    server socket
--staging-threshold N (5000000)    Stages messages over N bytes to disk
--m [ --mgmt-enable ] yes|no (1)   Enable Management
--mgmt-pub-interval SECONDS (10)  Management Publish Interval
--ack N (0)                         Send session.ack/solicit-ack at least every
                                    N frames. 0 disables voluntary ack/solictit
                                    -ack

Daemon options:
-d [ --daemon ]                   Run as a daemon.
-w [ --wait ] SECONDS (10)        Sets the maximum wait time to initialize the
                                    daemon. If the daemon fails to initialize, prints
                                    an error and returns 1
-c [ --check ]                   Prints the daemon's process ID to stdout and
                                    returns 0 if the daemon is running, otherwise
                                    returns 1
-q [ --quit ]                    Tells the daemon to shut down

Logging options:
--log-output FILE (stderr)        Send log output to FILE. FILE can be a file name
                                    or one of the special values:
                                    stderr, stdout, syslog
-t [ --trace ]                   Enables all logging
--log-enable RULE (error+)      Enables logging for selected levels and component
                                    s. RULE is in the form 'LEVEL+:PATTERN'
                                    Levels are one of:
                                    trace debug info notice warning error critical
                                    For example:
                                    '--log-enable warning+' logs all warning, error
                                    and critical messages.
                                    '--log-enable debug:framing' logs debug messages
                                    from the framing namespace. This option can be
                                    used multiple times
--log-time yes|no (1)            Include time in log messages
--log-level yes|no (1)            Include severity level in log messages
--log-source yes|no (0)           Include source file:line in log messages
--log-thread yes|no (0)           Include thread ID in log messages
--log-function yes|no (0)         Include function signature in log messages
```

Loading extra modules

By default the broker will load all the modules in the module directory, however it will NOT display options for modules that are not loaded. So to see the options for extra modules loaded you need to load the module and then add the help command like this:

```

./qpidd --load-module libbdbstore.so --help
Usage: qpidd OPTIONS
Options:
-h [ --help ]           Displays the help message
-v [ --version ]        Displays version information
--config FILE (/etc/qpid.conf) Reads configuration from FILE

/ .... non module options would be here ... /

Store Options:
--store-directory DIR   Store directory location for persistence (overrides
                        --data-dir)
--store-async yes|no (1) Use async persistence storage - if store supports
                        it, enables AIO O_DIRECT.
--store-force yes|no (0) Force changing modes of store, will delete all
                        existing data if mode is changed. Be SURE you want
                        to do this!
--num-jfiles N (8)      Number of files in persistence journal
--jfile-size-pgs N (24) Size of each journal file in multiples of read
                        pages (1 read page = 64kiB)

```

Getting Started Guide

Installing & Using Qpid (Java)

Introduction

The information below details how to install and use the main Broker and Client packages.

Essentially, to make use of the Qpid AMQP infrastructure you need to be able to run a broker instance to handle messaging traffic and talk to your client code. Your own application code will make use of the Qpid client package provided to interface with the broker.

Related pages to get you going

- [Troubleshooting Guide](#)
- [How to Use JNDI](#)
- [URL Formats for Qpid](#)
- [Example Classes](#)

Minor apologies since these instructions are heavily linux/unix focused. If you have difficulty using our .bat script (see below) please email [qpid-users](#) for assistance.

Prerequisites

The Qpid broker requires Java 5 or later to be available. For maximum performance Java 6 is recommended. Note that JDK 5 has a bug which may cause problems, so please use versions later than 1.5.0_15 !

The Java JMS client can be run using Java 1.4, 5 or 6. Note that the 1.4 client libraries come in a separate package.

Downloading & Installing Qpid

The latest binary and source distributions of Qpid Broker and Client packages are available from the [downloads](#) page.

If you want to use a newer version than an official release then you should check out the code from the [Subversion repository](#) and then consult the [Build How To](#) page.

Broker Install

Unpack the archive into any directory of your choice e.g c:/qpid

Once unpacked, the package will be installed in a directory with a release label (i.e. qpid-broker-0.5) and the directories underneath should look something like this:

/bin

Contains various startup utilities:

qpid-server is a bash script which runs the broker on linux/unix. You should set the environment variable QPID_HOME to point at your install path e.g. C:/qpid/qpid-broker-0.5. This enables the startup script to find default config files in the installed etc directory.

qpid-run is a script which allows shell commands to be executed which the qpid-server script utilises.

qpid-server.bat is a dos script which runs the broker on Windows. See note above about QPID_HOME.

create-example-ssl-stores(.sh/.bat) bash shell / dos batch script to create an example SSL keystore and truststore for use by the brokers JMX management connections, which now ship with SSL enabled by default. Either provide your own keystore by modifying the broker configuration, or run the appropriate script from the /etc directory to create example stores to allow initial broker startup. Alternatively you can modify the configuration files in /etc to turn SSL use off.

The other scripts in this directory can be safely ignored for now.

/lib

Contains all the jars used by the broker.

The **qpid-all.jar** contains a manifest file which puts the requisite jars into the classpath for broker startup.

Other files & jars in here can be safely ignored for now.

/etc

Contains the config files used by the broker on startup.

If running on a unix or linux platform check that the appropriate permissions have been applied to the .sh scripts. If not, then update i.e. chmod 755 *.sh

Environment Variables

Qpid Locations

You should set the following variables:

QPID_HOME - specifies where your install of Qpid exists, used for broker lookups of files etc

QPID_WORK - defines location of all working files created by the broker including log and db (i.e. BDB if used)

PATH - ensure that the QPID_HOME/bin directory is added to your path so that the server scripts can run

1. First set the QPID_HOME variable to reflect the root of your installation. For example, if you have installed the broker package into a directory <homedir>/broker/qpidd-xx then you should set the QPID_HOME variable to <homedir>/broker/qpidd-xx.
2. You can also set the QPID_WORK variable in order to control the destination directory for the log dir into which broker logging is generated. For example, if you wish to set the working directory to be <homedir>/working you should set the QPID_WORK directory to be <homedir>/working. Note that the QPID_WORK variable defaults to the current user's home directory if not set.
3. Then set your PATH variable appropriately to include the bin dir

Setting JAVA environment

You must make the JDK available available by setting the JAVA_HOME environment variable and adding the JAVA_HOME/bin directory to your PATH.

You should use JDK 1.6, or at least a version later than 1.5_15.

For example, if you have installed the JDK in /home/jdk1.6 then:

JAVA_HOME should be set to /home/jdk1.6

PATH should include /home/jdk1.6/bin

To check that you have completed this change successfully, simply type

```
java -version
```

You should see something like

```
java version "1.6.0_02"
Java(TM) SE Runtime Environment (build 1.6.0_02-b06)
Java HotSpot(TM) Server VM (build 1.6.0_02-b06, mixed mode)
```

The Qpid scripts set the classpath and other flags required for the broker to run.

Configuration

We ship two example configuration files with the Java broker:

persistent_config.xml - when you want to use any persistent messages with the Qpid broker (currently with BDB)

transient_config.xml - for transient messaging only

You can simply use one of these config files to get started, using the -c option to specify to the qpid-server script. See details next in the next section for more info on command line options.

Please visit our [3rd Party Libraries](#) page to get more generic information on how to set up your chosen persistence implementation for Qpid.

Running the Qpid Broker

There are scripts provided to run the broker on Windows and on Linux/Unix.

Running the Qpid Broker on Linux/Unix

- Make sure you have set the QPID_HOME variable as specified above, and QPID_WORK if you don't want the default
- Make sure you have provided an SSL keystore for the JMX management connections, or disabled the SSL usage, as detailed above in the 'broker install' section.
- One of the config files (persistent_config.xml or transient_config.xml) supplied in the etc directory one level below your root dir probably doesn't need modification and should be passed in using -c and the path to your config file
- Then run the qpid-server script from the root dir of your install. (The qpid-server script also supports cygwin environments.)

Command Line Arguments

You can get a list of all command line arguments by using the -h argument.

The following command line options are available:

| 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 |
| 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 |

For more detailed information on configuration, please see [Qpid Design - Configuration](#)

Checking the broker has started up

You can check that the broker has started up successfully by viewing the output it sends to stdout and looking for the start up port info:

```
2009-07-15 14:04:49,411 WARN [main] management.JMXManagedObjectRegistry (JMXManagedObjectRegistry.java:187) - Starting JMX ConnectorServer on port '8999' (+9099) with SSL
2009-07-15 14:04:49,842 INFO [main] server.Main (Main.java:279) - Starting Qpid Broker 0.5 build: xxxxxxx
2009-07-15 14:04:49,910 INFO [main] server.Main (Main.java:387) - Qpid.AMQP listening on non-SSL address 0.0.0.0/0.0.0.0:5672
```

Running the Qpid Broker on Windows

Simply set the QPID_HOME variable and run the .bat script. All other details as identical to running on Linux/Unix.

So for example:
qpid-server.bat

Getting Help

You can view our [FAQ](#) and [Troubleshooting Guide](#) for assistance. If you can't find the information that you need there, then email our [\[qpid users list\]](#)

Java broker log monitoring

Alert Monitoring

To set up required checks match log output on the following strings:

"ERROR"
"WARN"

With the following exclusions for warnings that are to be ignored:

"Requested requeue of message:"
"Routing map contains:"
"Dropping message as queue not required and there is no dead letter queue"

"Compressing Buffers on queue."
 "No additional SASL providers registered."
 "No Database or no mechanisms to initialise authentication"
 "VirtualHost authentication Managers require spec change to be operational."

Errors.

| Message Text | Reason |
|--|--|
| Error decrementing ref count on message <message id>: | Application coding error. Error logged but not rethrown and no other action taken. Broker keeps running. |
| Unsupported field type <class> for <field> IGNORING configured value | Error during configuration, application configuration error. Ignored. |
| Unable to expand property: | Error during configuration, application configuration error. Ignored. |
| Unable to access field <field> IGNORING configured value | Error during configuration, application configuration error. Ignored. |
| Exception occurred in creating the direct exchange mbean | Rethrown as AMQException. |
| MESSAGE LOSS: Message should be sent on a Dead Letter Queue | Non-mandatory message not routable. Route for message not set up correctly. |
| Exception occurred in creating the topic exchange mbean | Could not create mbean for exchange. Rethrown as AMQException. |
| Exception occurred in creating the direct exchange mbean | Could not create mbean for exchange. Rethrown as AMQException. |
| Exception occurred in creating the HeadersExchangeMBean | Could not create mbean for exchange. Rethrown as AMQException. |
| Default XPath evaluator could not be loaded | Coding error. |
| Error closing protocol session: | Error logged but not rethrown and no other action taken. Broker keeps running. |
| Error disposing of Sasl server: | Error logged but not rethrown and no other action taken. Broker keeps running. |
| Error disposing of Sasl server: | Error logged but not rethrown and no other action taken. Broker keeps running. |
| Unable to listen on SSL port: | Broker won't start. Port is unavailable. Port may be in use on server |
| Unable to bind service to registry: | Broker won't start. |
| AMQProtocolSession MBean creation has failed | Could not create mbean for session. Rethrown as AMQException. |
| Received incorrect protocol initiation | Client failed to open session. Possibly out of date client being used? |
| Error in protocol initiation | Client fail to open session. Possibly out of date client being used? |
| IOException caught in <session id>, session closed implicitly: | Connection lost due to io error. |
| Exception caught in <session id>, closing session explicitly: | Connection closed due to error/bad operation. |
| Unable to get body count: | The message store lost part of the message? or corrupt message taken off the wire? |
| Error getting body count: | The message store lost part of the message? or corrupt message taken off the wire? |
| Error getting size of message body. | The message store lost part of the message? or corrupt message taken off the wire? |
| Message was dequeued, but could not then be deleted though it is no longer referenced: | According to the comment this is a rare but harmless error. |
| Just send message: <message id> BUT removed this from queue: <message id> | Unexpected condition on the broker. Message sent should be message removed from queue. |
| Unable to deliver message as dequeue failed: | Message could not be delivered. |
| Attempt to send Null message | Application coding error. |
| Sending <message> when subscriber(<client id>) is closed! | Application coding error. |
| Sending <message> when subscriber(<client id>) is closed! | Application coding error. |
| MESSAGE LOSS : Unable to re-deliver messages | Failed messages cannot be redelivered. |
| [MESSAGES LOST]Unable to re-deliver messages as queue is null. | Failed messages cannot be redelivered. |

| | |
|--|---|
| Unable to re-deliver messages as queue is null. | Failed messages cannot be redelivered. |
| Queue is null won't be able to resend messages | Failed messages cannot be redelivered. |
| Unable to remove from index(<index>) subscription: | Unsubscription failed, due to unknown subscriber. Broker ignores and keeps running. |
| Error configuring application: | Bad configuration. Rethrown as runtime. Ignore. |
| Unable to instantiate configuration class <class> - ensure it has a public default constructor | Error in application configuration. |
| State manager received error notification[Current State:<state>]: | Protocol error. Client application may have used protocol incorrectly. Broker should handle this error correctly. |
| On committing transaction, unable to determine whether delivered to a consumer immediately: | Application coding error. |
| Failed to deliver messages following txn commit: | Application coding error. |
| Unable to instantiate configuration class <class> - ensure it has a public default constructor | Configuration exception. Rethrown as illegal argument. |
| Could not load version.properties resource: | The application has not been compiled with a version stamp. Bad build. |
| Error decoding FieldTable in deferred decoding mode | Rethrown as illegal argument. |

Warnings.

| Message Text | Reason |
|--|---|
| <object id> Requested requeue of message(<message id>): <delivery tag> but no queue defined and no DeadLetter queue so DROPPING message. | Undeliverable non-mandatory message. |
| Requested requeue of message: <delivery tag> but no such delivery tag exists.
<num unacked messages> | Coding error. Ignore. |
| No queues found for routing key <routing key> | Unroutable message, no route set up. |
| Routing map contains: <routing keys> | Debug trace of above. |
| Ignoring special header: <key> | Badly configured headers exchange. Application has not set up its headers exchange correctly. |
| Ignoring unrecognised match type: <value> | Badly configured headers exchange. Application has not set up its headers exchange correctly. |
| Invalid <class> implementation: | Coding error. |
| Dropping reject request as message is null for tag: <delivery tag> | Coding error. |
| Dropping message as requeue not required and there is no dead letter queue | Seems to be handled else where. |
| Unrecognised frame <class> | Bad frame. |
| Requesting rejection by null subscriber: <id> | Coding error. |
| Compressing Buffers on queue. | Warn about (costly?) internal broker actions. |
| No Configuration specified. Using default access controls for VirtualHost:'<vhost name>' | Security not configured for the application. |
| No access control specified. Using default access controls for VirtualHost:'<vhost name>' | Security not configured for the application. |
| Database '<database id>' cannot perform access management | Application configuration error. |
| Specified PrincipalDatabase is not an AccessManager so using default AccessManager | Application configuration error. |
| No Principal databases specified. Broker running with NO AUTHENTICATION. | Application configuration error. |
| No authentication specified for '<host>'. Using Default authentication manager | Possible application configuration error. |
| No authentication specified. Using Default authentication manager | Possible application configuration error. |
| Unable to set order of providers. | Coding or configuration error. |
| No additional SASL providers registered. | Configuration warning. |

| | |
|--|--|
| More than one principle database provided currently authentication mechanism will override each other. | Configuration error. |
| No Database or no mechanisms to initialise authentication | Reported elsewhere so ignore. |
| we need a server that will correctly convert the incoming plain text for comparison to file. | Application coding error. |
| Setting Accessible Name for VirtualHost is not allowed. | Application coding error. |
| VirtualHost authentication Managers require spec change to be operational. | Warning about future AMQP spec changed. |
| Unable to find resource <resource path> from classloader | The application has not been compiled with a version stamp. Bad build. |
| Could not find protocol conversion classes for <major version>-<minor version> | Application coding error. |
| Any access denied to vHost '<vhost name>' by <authorizer name> | Potential attempted security breach. |

Errors for Persistent Messaging (BDB Store) Only.

| Message Text | Reason | Action |
|--|---|--|
| Unable to create exchange: | Failed to instantiate exchange from BDB, possibly corrupt data store. | Shutdown broker. Restart broker to recover, if this fails, initiate restore from BDB backup. |
| Unable to create binding: | Failed to instantiate binding from BDB, possibly corrupt data store. | Shutdown broker. Restart broker to recover, if this fails, initiate restore from BDB backup. |
| Unable to create queue: | Failed to instantiate queue from BDB, possibly corrupt data store. | Shutdown broker. Restart broker to recover, if this fails, initiate restore from BDB backup. |
| Unknown queue: <queue name> cannot be bound to exchange: | Failed to instantiate queue from BDB, possibly corrupt data store. | Shutdown broker. Restart broker to recover, if this fails, initiate restore from BDB backup. |
| Failed to enqueue: | Message could not be placed on queue. Rethrown as AMQException. | Shutdown broker. Restart broker to recover, if this fails, initiate restore from BDB backup. |
| Failed to dequeue message <message id>: | Failed to load message from BDB, possibly corrupt data store. Rethrown as AMQException. | Shutdown broker. Restart broker to recover, if this fails, initiate restore from BDB backup. |
| Error: | BDB Database error prevented message delivery. Rethrown. | Shutdown broker. Restart broker to recover, if this fails, initiate restore from BDB backup. |
| Error converting entry to object: | Failed to instantiate meta data from BDB, possibly corrupt data store. | Shutdown broker. Restart broker to recover, if this fails, initiate restore from BDB backup. |
| Error shutting down message store: | Message store could not be cleanly shut down. Possibly corrupt message store? | Restart broker to recover, if this fails, initiate restore from BDB backup. |

Java Environment Variables

Setting Qpid Environment Variables

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.

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.

JMS Compliance

Strickt JMS Compliance

By default where AMQP is more flexible than JMS the AMQP option is used. To restrict operation to the JMS 1.1 specification set the system property:

```
-Dstrict-jms=true
```

Management Design notes

Status of This Document

This document does not track any current development activity. It is the specification of the management framework implemented in the M3 release of the C++ broker and will be left here for user and developer reference.

Development continues on the Qpid Management Framework (QMF) for M4. If you are using M3, this is the document you need. If you are using the SVN trunk, please refer to [Qpid Management Framework](#) for up-to-date information.

Introduction

This document describes the management features that are used in the QPID C++ broker as of the M3 milestone. These features do not appear in earlier milestones nor are they implemented in the Java broker.

This specification is **not** a standard and is not endorsed by the AMQP working group. When such a standard is adopted, the QPID implementation will be brought into compliance with that standard.

Links

- The schema is checked into [svn](#).

Design note for getting info in and out via JMX

[JMX WS-DM Gateway](#)

Management Requirements

- Must operate from a formally defined management schema.
- Must natively use the AMQP protocol and its type system.
- Must support the following operations
 - SET operation on configurable (persistent) aspects of objects
 - GET operation on all aspects of objects
 - METHOD invocation on schema-defined object-specific methods
 - Distribution of unsolicited periodic updates of instrumentation data
 - Data updates shall carry an accurate sample timestamp for rate calculation
 - Updates shall carry object create/delete timestamps.
 - Transient objects shall be fully accounted for via updates. Note that short-lived transient objects may come and go within a single update interval. All of the information pertaining to such an object must be captured and transmitted.
 - Distribution of unsolicited event and/or alert indications (schema defined)
- Role-based access control at object, operation, and method granularity
- End-to-end encryption and signing of management content
- Schema must be self-describing so the management client need not have prior knowledge of the management model of the system under management.
- Must be extensible to support the management of objects beyond the QPID component set. This allows AMQP to be used as a general-purpose management protocol.

Definition of Terms

| | |
|-----------|---|
| class | A type definition for a manageable object. |
| package | A grouping of class definitions that are related to a single software component. The package concept is used to extend the management schema beyond just the QPID software components. |
| object | Also "manageable object". An instantiation of a class. An object represents a physical or logical component in the core function of the system under management. |
| property | A typed member of a class which represents a configurable attribute of the class. In general, properties don't change frequently or may not change at all. |
| statistic | A typed member of a class which represents an instrumentation attribute of the class. Statistics are always read-only in nature and tend to change rapidly. |
| method | A member of a class which represents a callable procedure on an object of the class. Methods may have an arbitrary set of typed arguments and may supply a return code. Methods typically have side effects on the associated object. |

| | |
|-------------------|---|
| event | A member of a class which represents the occurrence of an event of interest within the system under management. |
| management broker | A software component built into the messaging broker that handles management traffic and distributes management data. |
| management agent | A software component that is separate from the messaging broker, connected to the management broker via an AMQP connection, which allows any software component to be managed remotely by QPID. |

Operational Scenarios: Basic vs. Extended

The extensibility requirement introduces complexity to the management protocol that is unnecessary and undesirable for the user/developer that wishes only to manage QPID message brokers. For this reason, the protocol is partitioned into two parts: The **basic protocol**, which contains only the capability to manage a single broker; and the **extended protocol**, which provides the hooks for managing an extended set of components. A management console can be implemented using only the basic protocol if the extended capabilities are not needed.

Architectural Framework

Architectural Framework

The Management Exchange

The management exchange (called "qpid.management" currently) is a special type of exchange used for remote management access to the Qpid broker. The management exchange is an extension of the standard "Topic" exchange. It behaves like a topic exchange with the following exceptions:

- When a queue is successfully bound to the exchange, a method is invoked on the broker's management agent to notify it of the presence of a new remote management client.
 - When messages arrive at the exchange for routing, the exchange examines the message's routing key and if the key represents a management command or method, it routes it directly to the management agent rather than routing it to queues using the topic algorithm.

The management exchange is used by the management agent to distribute unsolicited management data. Such data is classified by the routing key allowing management clients to register for only the data they need.

Routing Key Structure

As noted above, the structure of the binding and routing keys used on the management exchange is important to the function of the management architecture. The routing key of a management message determines:

1. The type of message (i.e. operation request or unsolicited update).
 2. The class of the object that the message pertains to.
 3. The specific operation or update type.
 4. The namespace in which the class belongs. This allows for plug-in expansion of the management schema for manageable objects that are outside of the broker itself.

Placing this information in the routing key provides the ability to enforce access control at class, operation, and method granularity. It also separates the command structure from the content of the management message (i.e. element values) allowing the content to be encrypted and signed end-to-end while still allowing access control at the message-transport level. This means that special access control code need not be written for the management agent.

There are two general types of routing/binding key:

- **Command** messages use the key: `agent.<bank#>` or `broker`
 - **Unsolicited** keys have the structure: `mgmt.<agent>.<type>.<package>.<class>.<severity>` where
 - `<agent>` is the uid of the originating management agent,
 - `<type>` is one of "schema", "prop", "stat", or "event",
 - `<package>` is the namespace in which the `<class>` name is valid, and
 - `<class>` is the name of the class as defined in the schema.
 - `<severity>` is relevant for events only. It is one of "critical", "error", "warning", or "info".

In both cases, the content of the message (i.e. method arguments, element values, etc.) is carried in the body segment of the message.

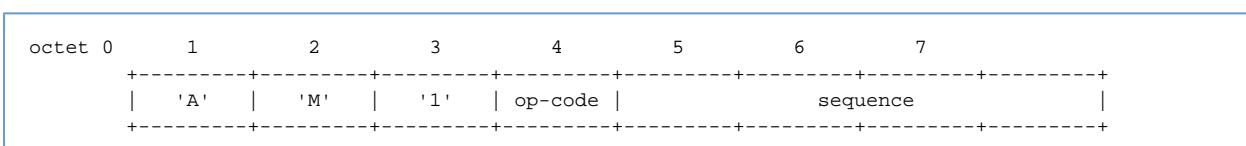
The `<package>` namespace allows this management framework to be extended with the addition of other software packages.

The Protocol

Protocol Header

The body segments of management messages are composed of sequences of binary-encoded data fields, in a manner consistent with the 0-10 version of the AMQP specification.

All management messages begin with a message header:



The first three octets contain the protocol **magic number** "AM1" which is used to identify the type and version of the message.

The **opcode** field identifies the operation represented by the message

Protocol Exchange Patterns

The following patterns are followed in the design of the protocol:

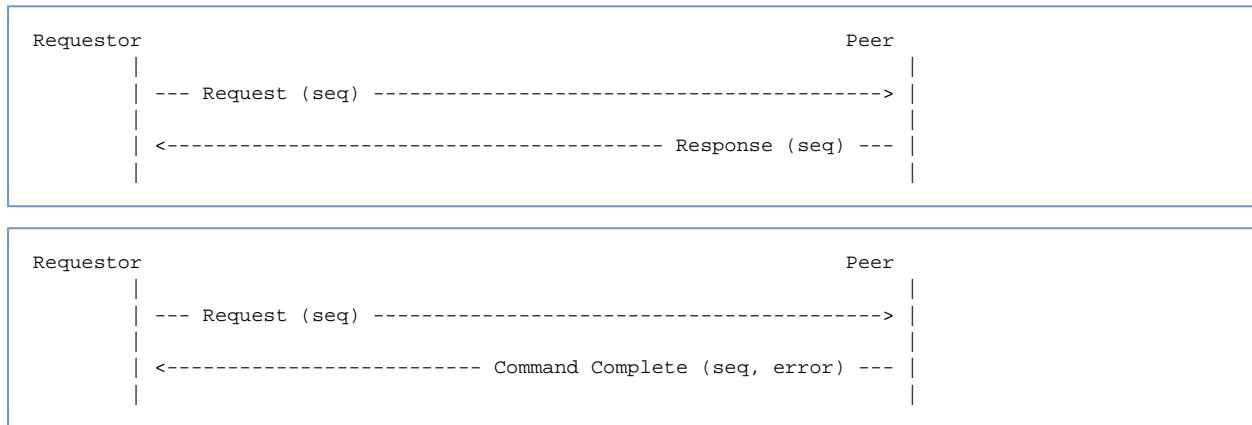
- Request-Response
- Query-Indication
- Unsolicited Indication

The Request-Response Pattern

In the request-response pattern, a requestor sends a **request** message to one of its peers. The peer then does one of two things: If the request can be successfully processed, a single **response** message is sent back to the requestor. This response contains the requested results and serves as the positive acknowledgement that the request was successfully completed.

If the request cannot be successfully completed, the peer sends a **command complete** message back to the requestor with an error code and error text describing what went wrong.

The sequence number in the **response** or **command complete** message is the same as the sequence number in the **request**.

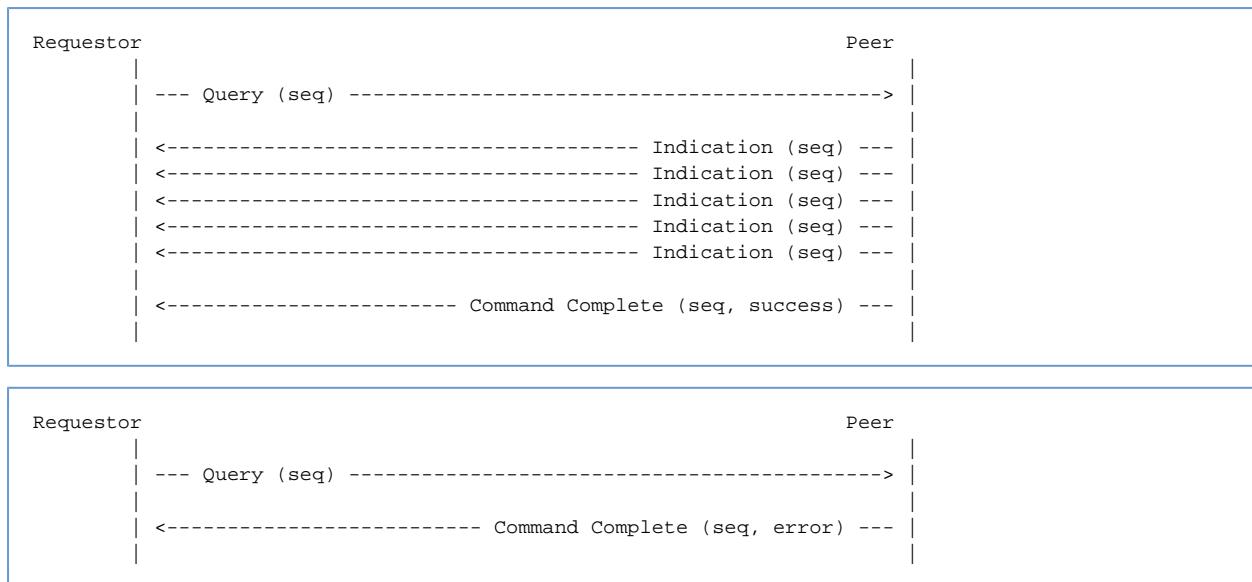


The Query-Indication Pattern

The query-indication pattern is used when there may be zero or more answers to a question. In this case, the requestor sends a **query** message to its peer. The peer processes the query, sending as many **indication** messages as needed back to the requestor (zero or more). Once the last **indication** has been sent, the peer then sends a **command complete** message with a success code indicating that the query is complete.

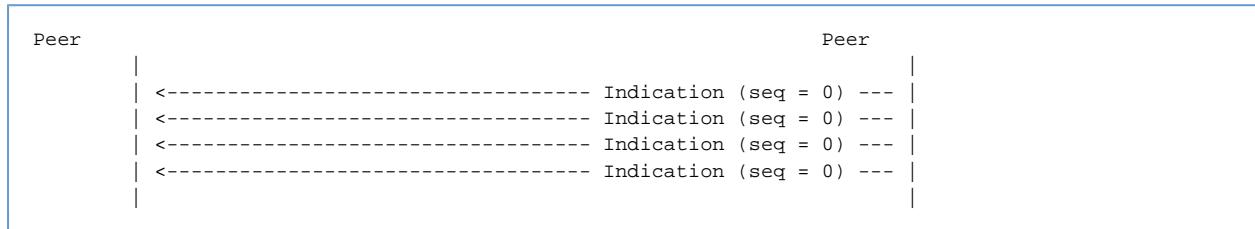
If there is an error in the **query**, the peer may reply with a **command complete** message containing an error code. In this case, no **indication** messages may be sent.

All **indication** and **command complete** messages shall have the same sequence number that appeared in the **query** message.



The Unsolicited-Indication Pattern

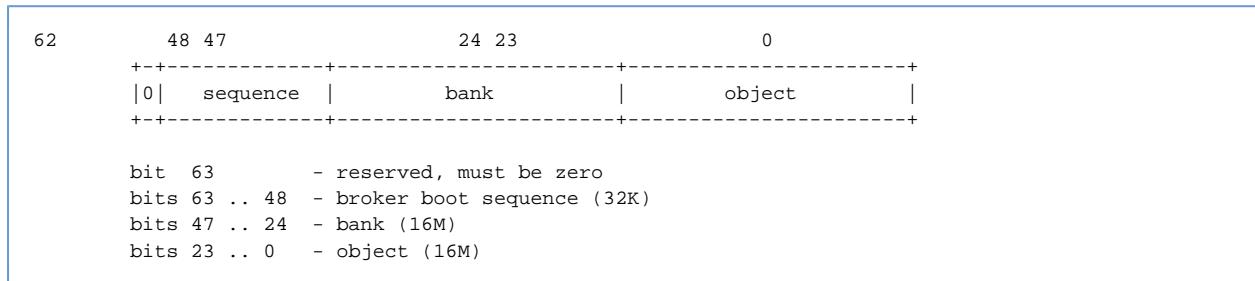
The unsolicited-indication pattern is used when one peer needs to send unsolicited information to another peer, or to broadcast information to multiple peers via a topic exchange. In this case, indication messages are sent with the sequence number field set to zero.



Object Identifiers

Manageable objects are tagged with a unique 64-bit object identifier. The object identifier space is owned and managed by the management broker. Objects managed by a single management broker shall have unique object identifiers. Objects managed by separate management brokers may have the same object identifier.

If a management console is designed to manage multiple management brokers, it must use the broker identifier as well as the object identifier to ensure global uniqueness.



- For persistent IDs, boot-sequence is zero
- For non-persistent IDs, boot sequence is a constant number which increments each time the management broker is restarted.
- Bank number:
 - 0 - reserved
 - 1 - broker-persistent objects
 - 2..4 - store-persistent objects
 - > 4 - transient objects

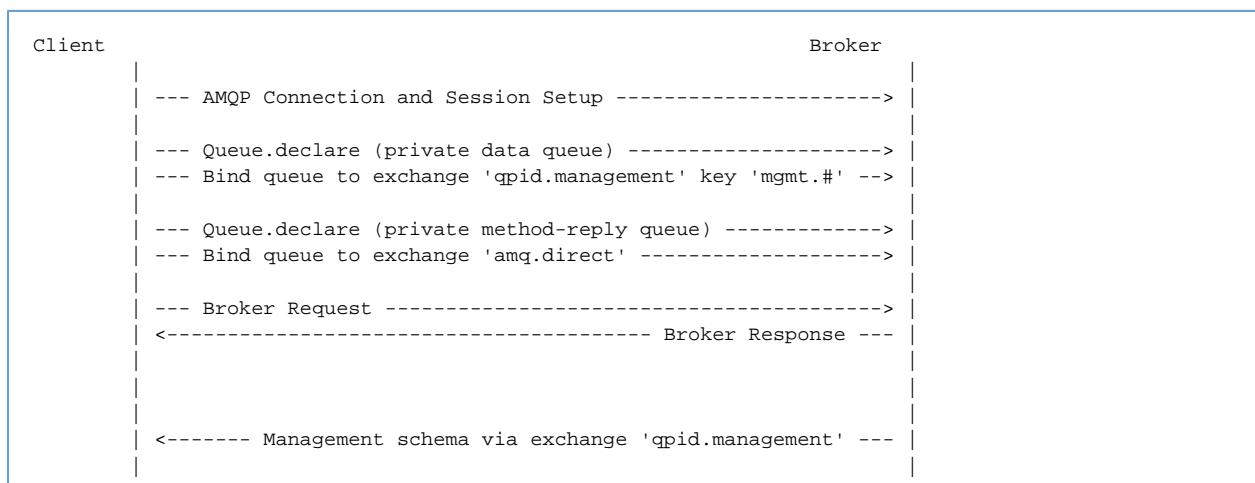
Establishing Communication Between Client and Agent

Communication is established between the management client and management agent using normal AMQP procedures. The client creates a connection to the broker and then establishes a session with its corresponding channel.

Two private queues are then declared (only one if method invocation is not needed). A management queue is declared and bound to the qpid.management exchange. If the binding key is "mgmt.#", all management-related messages sent to the exchange will be received by this client. A more specific binding key will result in a more restricted set of messages being received (see the section on Routing Key Structure below).

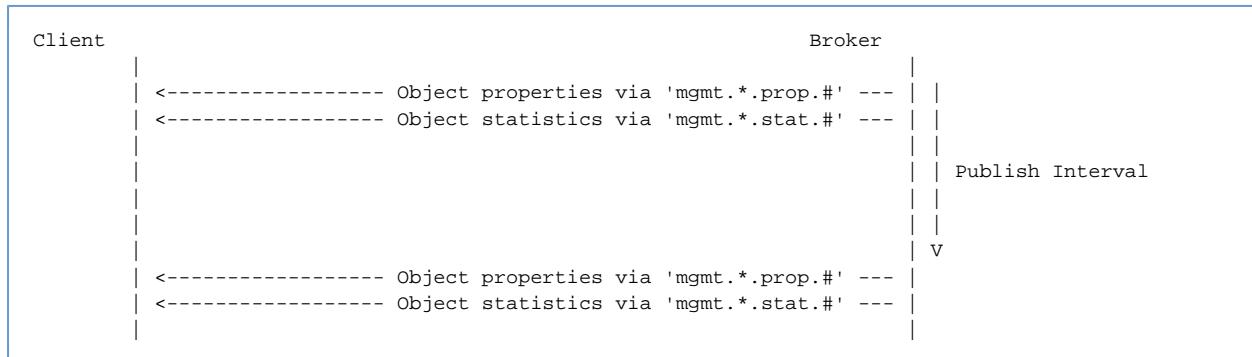
If methods are going to be invoked on managed objects, a second private queue must be declared so the client can receive method replies. This queue is bound to the amq.direct exchange using a routing key equal to the name of the queue.

When a client successfully binds to the qpid.management exchange, the management agent schedules a schema broadcast to be sent to the exchange. The agent will publish, via the exchange, a description of the schema for all manageable objects in its control.



Broadcast of Configuration and Instrumentation Updates

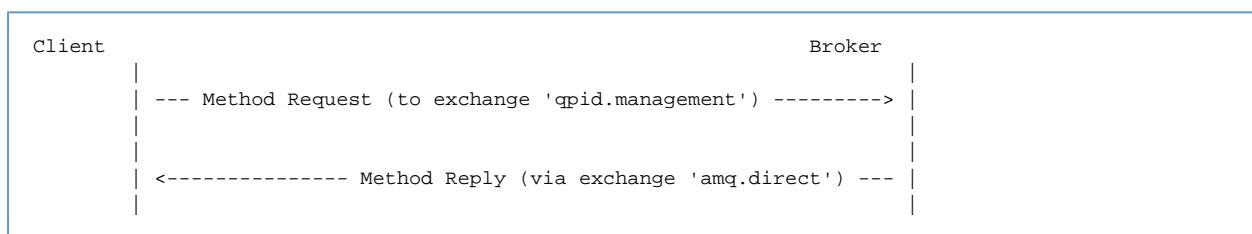
The management agent will periodically publish updates to the configuration and instrumentation of management objects under its control. Under normal circumstances, these updates are published only if they have changed since the last time they were published. Configuration updates are only published if configuration has changed and instrumentation updates are only published if instrumentation has changed. The exception to this rule is that after a management client binds to the qpid.management exchange, all configuration and instrumentation records are published as though they had changed whether or not they actually did.



Invoking a Method on a Managed Object

When the management client wishes to invoke a method on a managed object, it sends a method request message to the qpid.management exchange. The routing key contains the object class and method name (refer to Routing Key Structure below). The method request must have a header entry (reply-to) that contains the name of the method-reply queue so that the method response can be properly routed back to the requestor.

The method request contains a sequence number that is copied to the method reply. This number is opaque to the management agent and may be used by the management client to correlate the reply to the request. The asynchronous nature of requests and replies allows any number of methods to be in-flight at a time. Note that there is no guarantee that methods will be replied to in the order in which they were requested.



Messages for the Basic Scenario

The principals in a management exchange are the *management client* and the *management agent*. The management agent is integrated into the QPID broker and the management client is a remote entity. A management agent may be managed by zero or more management clients at any given time. Additionally, a management client may manage multiple management agents at the same time.

For authentication and access control, management relies on the mechanisms supplied by the AMQP protocol.

Basic Opcodes

| opcode | message | description |
|--------|--------------------|--|
| 'B' | Broker Request | This message contains a broker request, sent from the management console to the broker to initiate a management session. |
| 'b' | Broker Response | This message contains a broker response, sent from the broker in response to a broker request message. |
| 'z' | Command Completion | This message is sent to indicate the completion of a request. |
| 'Q' | Class Query | Class query messages are used by a management console to request a list of schema classes that are known by the management broker. |
| 'q' | Class Indication | Sent by the management broker, a class indication notifies the peer of the existence of a schema class. |
| 'S' | Schema Request | Schema request messages are used to request the full schema details for a class. |
| 's' | Schema Response | Schema response message contain a full description of the schema for a class. |

| | | |
|---------------|----------------------|---|
| 'h' | Heartbeat Indication | This message is published once per publish-interval. It can be used by a client to positively determine which objects did not change during the interval (since updates are not published for objects with no changes). |
| 'c', 'i', 'g' | Content Indication | This message contains a content record. Content records contain the values of all properties or statistics in an object. Such records are broadcast on a periodic interval if 1) a change has been made in the value of one of the elements, or 2) if a new management client has bound a queue to the management exchange. |
| 'G' | Get Query | Sent by a management console, a get query requests that the management broker provide content indications for all objects that match the query criteria. |
| 'M' | Method Request | This message contains a method request. |
| 'm' | Method Response | This message contains a method result. |

Broker Request Message

When a management client first establishes contact with the broker, it sends a Hello message to initiate the exchange.

```
+-----+-----+-----+-----+
| 'A' | 'M' | '1' | 'B' |      0      |
+-----+-----+-----+-----+
```

The Broker Request message has no payload.

Broker Response Message

When the broker receives a Broker Request message, it responds with a Broker Response message. This message contains an identifier unique to the broker.

```
+-----+-----+-----+-----+
| 'A' | 'M' | '1' | 'b' |      0      |
+-----+-----+-----+-----+
| brokerId (uuid)           |
+-----+-----+-----+
```

Command Completion Message

```
+-----+-----+-----+-----+
| 'A' | 'M' | '1' | 'z' |      seq      |
+-----+-----+-----+-----+
| Completion Code           |
+-----+-----+-----+
| Completion Text           |
+-----+-----+-----+
```

Class Query

```
+-----+-----+-----+-----+
| 'A' | 'M' | '1' | 'Q' |      seq      |
+-----+-----+-----+-----+
| package name (str8)       |
+-----+-----+-----+
```

Class Indication

```
+-----+-----+-----+-----+
| 'A' | 'M' | '1' | 'q' |      seq      |
+-----+-----+-----+-----+
| package name (str8)       |
+-----+-----+-----+
| class name (str8)         |
+-----+-----+-----+
| schema hash (bin128)      |
+-----+-----+-----+
```

Schema Request

```
+-----+-----+-----+-----+
| 'A' | 'M' | '1' | 'S' |      seq      |
+-----+-----+-----+-----+
|          packageName (str8)           |
+-----+-----+-----+-----+
|          className (str8)            |
+-----+-----+-----+-----+
|          schema-hash (bin128)        |
+-----+-----+-----+-----+
```

Schema Response

```
+-----+-----+-----+-----+
| 'A' | 'M' | '1' | 's' |      seq      |
+-----+-----+-----+-----+
|          packageName (str8)           |
+-----+-----+-----+-----+
|          className (str8)            |
+-----+-----+-----+-----+
|          schema-hash (bin128)        |
+-----+-----+-----+-----+
| propCnt | statCnt | methodCnt | eventCnt |
+-----+-----+-----+-----+
| propCnt property records           |
+-----+-----+-----+-----+
| statCnt statistic records          |
+-----+-----+-----+-----+
| methodCnt method records          |
+-----+-----+-----+-----+
| eventCnt event records            |
+-----+-----+-----+-----+
```

Each **property** record is an AMQP map with the following fields. Optional fields may optionally be omitted from the map.

| field name | optional | description |
|------------|----------|---|
| name | no | Name of the property |
| type | no | Type code for the property |
| access | no | Access code for the property |
| index | no | 1 = index element, 0 = not an index element |
| optional | no | 1 = optional element (may be not present), 0 = mandatory (always present) |
| unit | yes | Units for numeric values (i.e. seconds, bytes, etc.) |
| min | yes | Minimum value for numerics |
| max | yes | Maximum value for numerics |
| maxlen | yes | Maximum length for strings |
| desc | yes | Description of the property |

Each **statistic** record is an AMQP map with the following fields:

| field name | optional | description |
|------------|----------|--|
| name | no | Name of the statistic |
| type | no | Type code for the statistic |
| unit | yes | Units for numeric values (i.e. seconds, bytes, etc.) |
| desc | yes | Description of the statistic |

method and **event** records contain a main map that describes the method or header followed by zero or more maps describing arguments. The main map contains the following fields:

| field name | optional | description |
|------------|----------|-------------|
| | | |

| | | |
|----------|-----|--------------------------------------|
| name | no | Name of the method or event |
| argCount | no | Number of argument records to follow |
| desc | yes | Description of the method or event |

Argument maps contain the following fields:

| field name | method | event | optional | description |
|------------|--------|-------|----------|--|
| name | yes | yes | no | Argument name |
| type | yes | yes | no | Type code for the argument |
| dir | yes | no | yes | Direction code for method arguments |
| unit | yes | yes | yes | Units for numeric values (i.e. seconds, bytes, etc.) |
| min | yes | no | yes | Minimum value for numerics |
| max | yes | no | yes | Maximum value for numerics |
| maxlen | yes | no | yes | Maximum length for strings |
| desc | yes | yes | yes | Description of the argument |
| default | yes | no | yes | Default value for the argument |

type codes are numerics with the following values:

| value | type |
|-------|-------------------------|
| 1 | uint8 |
| 2 | uint16 |
| 3 | uint32 |
| 4 | uint64 |
| 6 | str8 |
| 7 | str16 |
| 8 | absTime(uint64) |
| 9 | deltaTime(uint64) |
| 10 | objectReference(uint64) |
| 11 | boolean(uint8) |
| 12 | float |
| 13 | double |
| 14 | uuid |
| 15 | map |
| 16 | int8 |
| 17 | int16 |
| 18 | int32 |
| 19 | int64 |

access codes are numerics with the following values:

| value | access |
|-------|--------------------|
| 1 | Read-Create access |
| 2 | Read-Write access |
| 3 | Read-Only access |

direction codes are numerics with the following values:

| value | direction |
|-------|--------------------------------|
| 1 | Input (from client to broker) |
| 2 | Output (from broker to client) |
| 3 | IO (bidirectional) |

Heartbeat Indication

```
+-----+-----+-----+-----+
| 'A' | 'M' | '1' | 'h' |          0      |
+-----+-----+-----+-----+
| timestamp of current interval (datetime)   |
+-----+-----+
```

Configuration and Instrumentation Content Messages

Content messages are published when changes are made to the values of properties or statistics or when new management clients bind a queue to the management exchange.

```
+-----+-----+-----+-----+
| 'A' | 'M' | '1' | 'g/c/i' |      seq      |
+-----+-----+-----+-----+
| packageName (str8)                         |
+-----+-----+
| className (str8)                          |
+-----+-----+
| class hash (bin128)                      |
+-----+-----+
| timestamp of current sample (datetime)    |
+-----+-----+
| time object was created (datetime)        |
+-----+-----+
| time object was deleted (datetime)        |
+-----+-----+
| objectId (uint64)                         |
+-----+-----+
| presence bitmasks (0 or more uint8 fields)|
+-----+-----+
| config/inst values (in schema order)      |
+-----+-----+
```

All timestamps are uint64 values representing nanoseconds since the epoch (January 1, 1970). The objectId is a uint64 value that uniquely identifies this object instance.

If any of the properties in the object are defined as optional, there will be 1 or more "presence bitmask" octets. There are as many octets as are needed to provide one bit per optional property. The bits are assigned to the optional properties in schema order (first octet first, lowest order bit first).

For example: If there are two optional properties in the schema called "option1" and "option2" (defined in that order), there will be one presence bitmask octet and the bits will be assigned as bit 0 controls option1 and bit 1 controls option2.

If the bit for a particular optional property is set (1), the property will be encoded normally in the "values" portion of the message. If the bit is clear (0), the property will be omitted from the list of encoded values and will be considered "NULL" or "not present".

The element values are encoded by their type into the message in the order in which they appeared in the schema message.

Get Query Message

A Get Request may be sent by the management console to cause a management agent to immediately send content information for objects of a class.

```
+-----+-----+-----+-----+
| 'A' | 'M' | '1' | 'G' |      seq      |
+-----+-----+-----+-----+
| Get request field table                  |
+-----+-----+
```

The content of a get request is a field table that specifies what objects are being requested. Most of the fields are optional and are available for use in more extensive deployments.

| Field Key | Mandatory | Type | Description |
|------------|-----------|--------------|---|
| "_class" | yes | short-string | The name of the class of objects being requested. |
| "_package" | no | short-string | The name of the extension package the class belongs to. If omitted, the package defaults to "qpid" for access to objects in the connected broker. |
| "_agent" | no | uuid | The management agent that is the target of the request. If omitted, agent defaults to the connected broker. |

When the management agent receives a get request, it sends content messages describing the requested objects. Once the last content message is sent, it then sends a Command Completion message with the same sequence number supplied in the request to indicate to the requestor that there are no more messages coming.

Method Request

Method request messages have the following structure. The sequence number is opaque to the management agent. It is returned unchanged in the method reply so the calling client can correctly associate the reply to the request. The objectId is the unique ID of the object on which the method is to be executed.

```
+-----+-----+-----+-----+
| 'A' | 'M' | '1' | 'M' |      seq      |
+-----+-----+-----+-----+
| objectId (uint64)           |
+-----+-----+
| methodName (str8)          |
+-----+-----+
| input and bidirectional argument values (in schema order) |
+-----+-----+
```

Method Response

Method reply messages have the following structure. The sequence number is identical to that supplied in the method request. The status code (and text) indicate whether or not the method was successful and if not, what the error was. Output and bidirectional arguments are only included if the status code was 0 (STATUS_OK).

```
+-----+-----+-----+-----+
| 'A' | 'M' | '1' | 'm' |      seq      |
+-----+-----+-----+-----+
| status code           |
+-----+-----+
| status text (str8)    |
+-----+-----+
| output and bidirectional argument values (in schema order) |
+-----+-----+
```

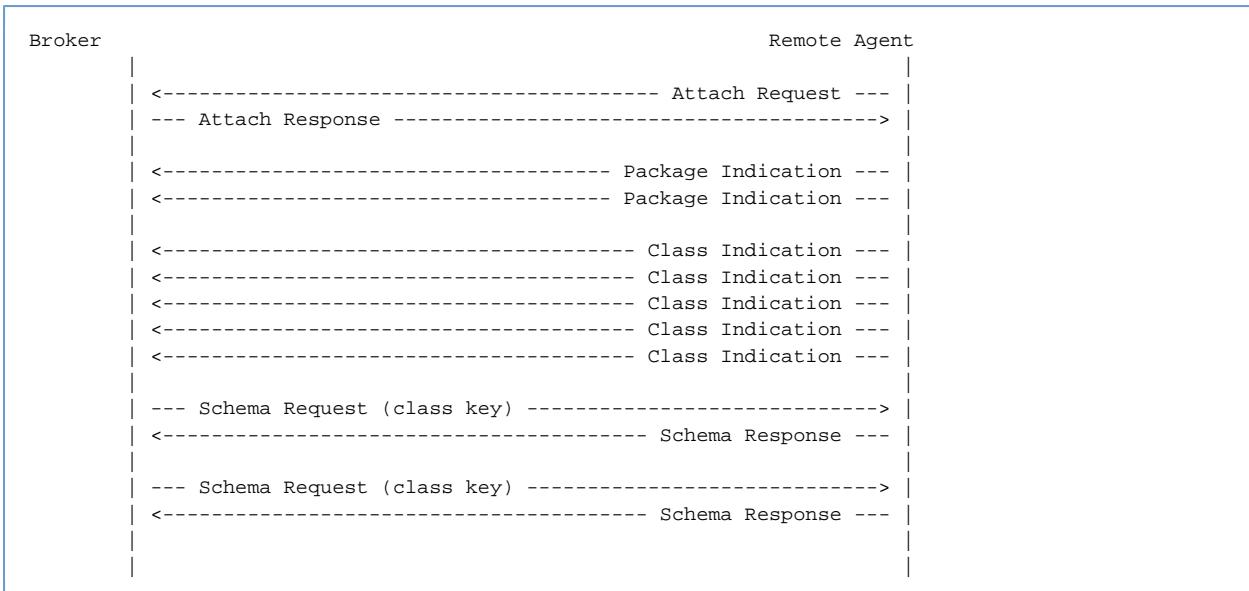
status code values are:

| value | description |
|-------|--|
| 0 | STATUS_OK - successful completion |
| 1 | STATUS_UNKNOWN_OBJECT - objectId not found in the agent |
| 2 | STATUS_UNKNOWN_METHOD - method is not known by the object type |
| 3 | STATUS_NOT_IMPLEMENTED - method is not currently implemented |

Messages for Extended Scenario

Extended Management Protocol

Qpid supports management extensions that allow the management broker to be a central point for the management of multiple external entities with their own management schemas.



Extended Opcodes

| opcode | message | description |
|--------|--------------------------|--|
| 'P' | Package Query | This message contains a schema package query request, requesting that the broker dump the list of known packages |
| 'p' | Package Indication | This message contains a schema package indication, identifying a package known by the broker |
| 'A' | Agent Attach Request | This message is sent by a remote agent when it wishes to attach to a management broker |
| 'a' | Agent Attach Response | The management broker sends this response if an attaching remote agent is permitted to join |
| 'x' | Console Added Indication | This message is sent to all remote agents by the management broker when a new console binds to the management exchange |

Package Query

```

+-----+-----+-----+-----+
| 'A' | 'M' | '1' | 'P' |      seq      |
+-----+-----+-----+-----+

```

Package Indication

```

+-----+-----+-----+-----+
| 'A' | 'M' | '1' | 'p' |      seq      |
+-----+-----+-----+-----+
| package name (str8)           |
+-----+

```

Attach Request

```

+-----+-----+-----+-----+
| 'A' | 'M' | '1' | 'A' |      seq      |
+-----+-----+-----+-----+
| label (str8)                   |
+-----+
| system-id (uuid)              |
+-----+
| requested objId bank          |
+-----+

```

Attach Response (success)

| | | | | | |
|----------------------|-----|-----|-----|-----|--|
| 'A' | 'M' | '1' | 'a' | seq | |
| assigned broker bank | | | | | |
| assigned objId bank | | | | | |

Console Added Indication

| | | | | | |
|-----|-----|-----|-----|-----|--|
| 'A' | 'M' | '1' | 'x' | seq | |
|-----|-----|-----|-----|-----|--|

JMX Gateway

Qpid Management and JMX

Currently the C++ broker supports the AMQP-mgmt protocol and makes all its management information available to any language client. Each object that is managed has a schema and MD5 sum so that version-ing can be handled.

To play with this [look here](#)

To see the current [object schema](#)

The Java Broker already has JMX exposed – schema to be link here...

What all the pieces are...

Consuming the mgmt events from JMX

We have a GSoC project done by Rahul, that will take the schema from AMQP-mgmt and then dynamically expose those over JMX and WS-DM. This means that any of the management data from the broker can be managed by any JMX console or WS-DM console. I believe that Andrea will be helping with the WS-DM piece.

[Notes on this project](#)

Putting JMX events across the AMQP-mgmt pipe

In this case we want to be able to take any JMX objects and map them into the AMQP-mgmt pipe, as an agent. This makes it possible for any process including our current Java broker to place all its events onto the infrastructure. I believe Andrea is doing this for us.

Command line tools for JMX

The ability to hit the JMX interfaces from cmd line tool for scripting on JMX. This is a GSoC project, Lahiru is doing this for us.

Mapping the schema

Brought up by Rob, basically we need to work through all the management commands and instrumentation data that the two brokers have and make sure the full set is represented into the [schema](#)

Finally we need to work out if our users want us to provide a translation bridge between the two schema.

qmf_architecture

Architectural Framework

Components

Management Broker

The management broker has the following responsibilities:

1. Manage the object-id space used to uniquely identify all manageable objects.
2. Route agent commands from a console to the appropriate agent.
3. Cache schema information provided by agents for the use of consoles.

Management Agent

The management agent consists of two parts. The first part is a component of the Qpid Management Framework and provides an API interface for the second part. The second part is specific to the software system being managed and is developed by the same team that

develops the target system.

The management agent is responsible for the following:

1. Defining and owning the management schema for the target system.
2. Maintaining manageable objects that are associated with physical or logical objects in the target system.
3. Executing schema-defined method requests on object under its care.

Management Console

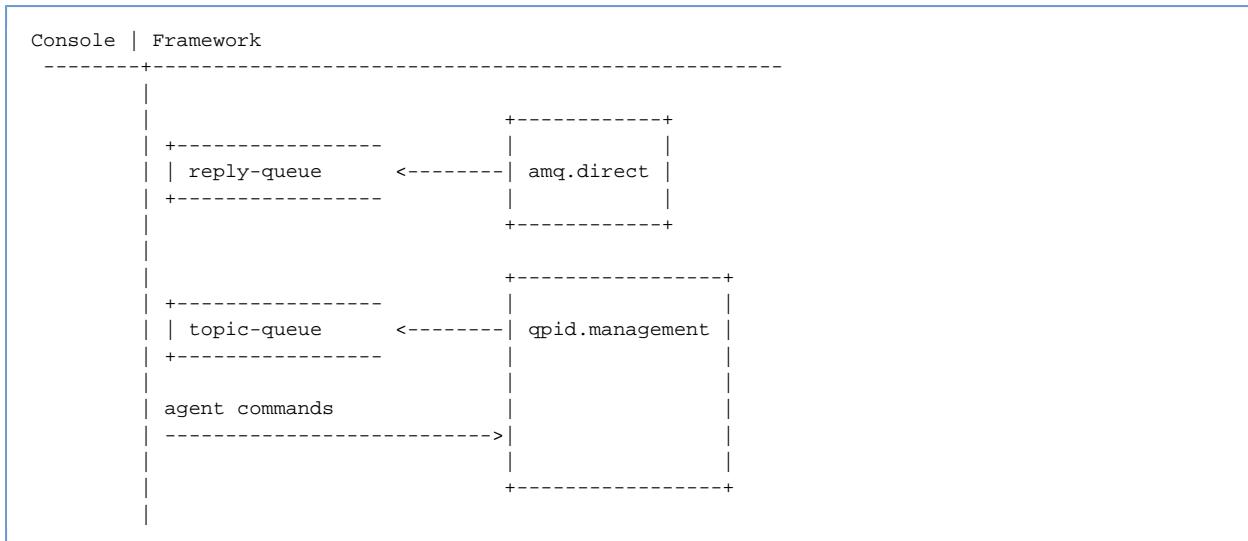
The management console consists of two main parts: The framework-supplied access API and the user-defined application built on the API. The user application is typically a CLI utility or a Graphical/Browser user interface but can take any other form as well.

Other examples of console applications are:

- Event and audit storage applications
- Event correlation applications
- Two-tiered management servers (for web-based UIs)
- Bridges to other management protocols
- Automated monitoring and control applications that react to changes in the managed infrastructure
- Test harnesses
- Custom-built applications for any purpose

Interfaces

Console Interface



Agent Interface

Features

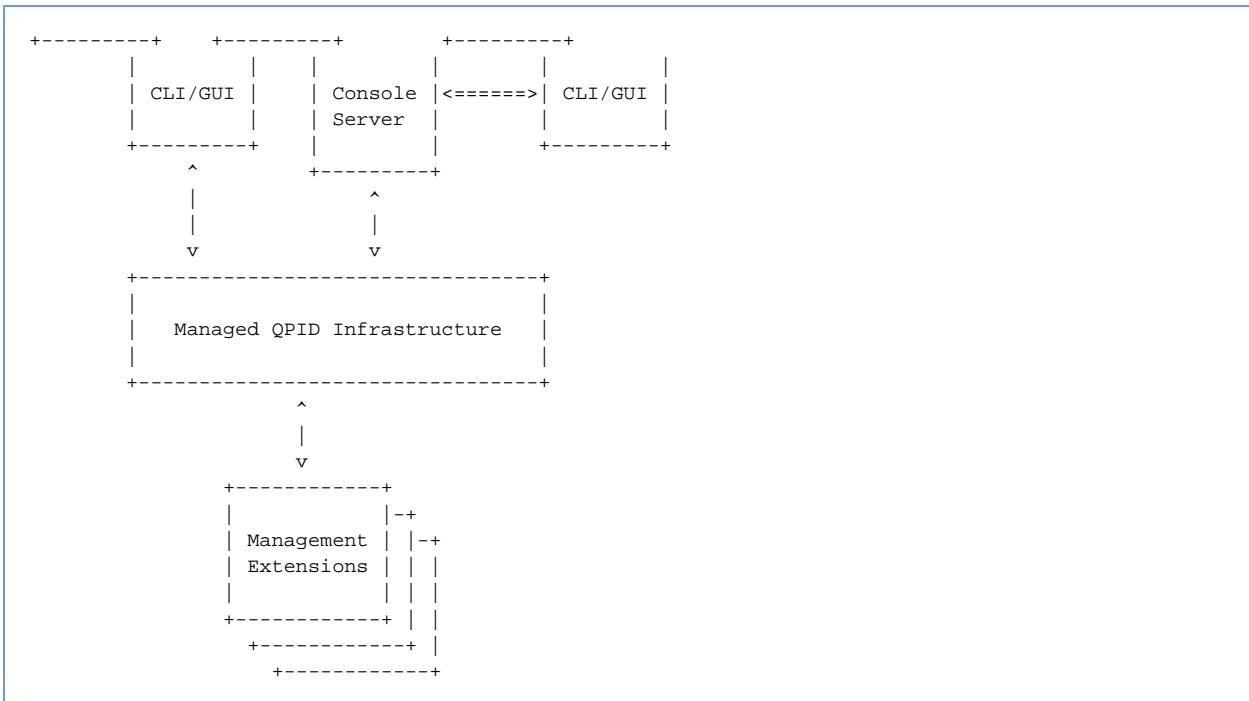
Redundant components and paths

Single point of entry

Old Text

There are two primary interfaces defined in the management architecture:

1. The Management Console Interface is used by management clients (CLIs, GUIs, console servers, etc.) to remotely access management data.
2. The Extension Interface is used by software components (not necessarily related to the QPID infrastructure) to provide access to their managed objects.



Both management interfaces are based on the AMQP protocol and its type system.

Management Tools

Current Management Tools

This is a list of the current management tools available for the Qpid Java Broker.

- JMX Management Console
- JConsole
- MessageStore Tool

JConsole

JConsole

JConsole is a management tool that comes with the Java Runtime Environment (6+) or Java Development Kit (5+) and provides a very simple view of managed beans. It requires no special configuration to be used with Qpid.

You can run JConsole with the command 'jconsole' assuming you have Java installed and configured to be available in your PATH.

Recent versions of the broker can make use of SSL to encrypt their RMI based JMX connections. If the broker being connected to is making use of this ability then additional configuration may be required, particularly when using self-signed certificates, in order to provide JConsole with access to an SSL truststore capable of validating the certificate received from the broker. If you don't do this JConsole will fail to connect, although it will not emit a clear indication of why.

As example, in order to connect to the broker using the test SSL resources within the Qpid subversion repository, in the trunk/qpid/java directory the following command could be used to start jconsole:

```
jconsole -J-Djavax.net.ssl.trustStore=test-profiles/test_resources/ssl/certstore.jks
-J-Djavax.net.ssl.trustStorePassword=password
```

To attach to a (remotely) running broker, simply enter the host, port, and login details in the JConsole connect dialog. Once you are connected expand the tree nodes marked "org.apache.qpid" to gain access to the Qpid related MBeans.

For further details see Sun's JConsole [guide](#)

MessageStore Tool

MessageStore Tool

We have a number of implementations of the Qpid MessageStore interface. This tool allows the interrogation of these stores while the broker is offline.

MessageStore Implementations

- [BDBMessageStore \(3rd Party\)](#)
- [JDBCStore](#)
- [MemoryMessageStore](#)

Introduction

Each of the MessageStore implementations provide different back end storage for their messages and so would need a different tool to be able to interrogate their contents at the back end.

What this tool does is to utilise the Java broker code base to access the contents of the storage providing the user with a consistent means to inspect the storage contents in broker memory. The tool allows the current messages in the store to be inspected and copied/moved between queues. The tool uses the message instance in memory for all its access paths, but changes made will be reflected in the physical store (if one exists).

Usage

The tools-distribution currently includes a unix shell command 'msTool.sh' this script will launch the java tool.

The tool loads \$QPID_HOME/etc/config.xml by default. If an alternative broker configuration is required this should be provided on the command line as would be done for the broker.

```
msTool.sh -c <path to different config.xml>
```

On startup the user is present with a command prompt

```
$ msTool.sh
MessageStoreTool - for examining Persistent Qpid Broker MessageStore instances
bdb$
```

Available Commands

The available commands in the tool can be seen through the use of the 'help' command.

```
bdb$ help
+-----+
|           Available Commands           |
+-----+
| Command | Description                |
+-----+
| quit   | Quit the tool.               |
| list   | list available items.          |
| dump   | Dump selected message content. Default: show=content |
| load   | Loads specified broker configuration file. |
| clear  | Clears any selection.          |
| show   | Shows the messages headers.    |
| select  | Perform a selection.          |
| help   | Provides detailed help on commands. |
+-----+
bdb$
```

A brief description is displayed and further usage information is shown with 'help <command>'

```
bdb$ help list
list available items.
Usage:list queues [<exchange>] | exchanges | bindings [<exchange>] | all
bdb$
```

Future Work

Currently the tool only works whilst the broker is offline i.e. it is up, but not accepting AMQP connections. This requires a stop/start of the broker. If this functionality was incorporated into the broker then a telnet functionality could be provided allowing online management.

Qpid JMX Management Console

Qpid JMX Management Console

Overview

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

- Configuring Management Users
- Configuring Qpid JMX Management Console
 - Management Console Security
- Qpid JMX Management Console FAQ
- Qpid JMX Management Console User Guide
- Qpid Management Features

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.

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: Add New Users](#)
2. Grant the new user permission to the JMX Console

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.

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.

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
```

Configuring Qpid JMX Management Console

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 [here](#).

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 [Management Console Security](#) 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 [Management Console Security](#).

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).

Using the Qpid JMX Management Console

Please see [Qpid JMX Management Console User Guide](#) for details on using this Eclipse RCP application.

Using JConsole

See [JConsole](#)

Using HermesJMS

HermesJMS also offers integration with the Qpid management interfaces. You can get instructions and more information from <http://wiki.apache.org/qpid/HermesJMS>.

Using MC4J

MC4J is an alternative management tool. It provide a richer "dashboard" that can customise the raw MBeans.

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

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 port changes.

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.

Management Console Security

Management Console Security

- SSL encrypted RMI (0.5 and above)
- JMXMP (M4 and previous)
- User Accounts & Access Rights

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).

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 using the example 'create-example-ssl-stores' script in the brokers bin/ directory to generate 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 ${conf}/qpid.keystore -->
    <keyStorePath>${prefix}/../test_resources/ssl/keystore.jks</keyStorePath>
    <keyStorePassword>password</keyStorePassword>
  </ssl>
</management>
```

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 [User Guide](#) for more information on this configuration process.

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 [JConsole](#) for further details on configuration.

Additional Information

More information on Java's handling of SSL certificate verification and customizing the keystores can be found in the [JSSE Reference Guide](#).

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.

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 [here](#). 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.

Broker Configuration

To enable 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"
```

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.

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 [Configuring Management Users](#)

Qpid JMX Management Console FAQ

Errors

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

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

Qpid JMX Management Console User Guide

Qpid JMX Management Console User Guide

The guide can be found below in wiki form, or downloaded as a file: ([DOC](#)) ([PDF](#))

Introduction
Startup & Configuration
Startup
SSL configuration
JMXMP configuration
Managing Server Connections
Main Toolbar
Connecting to a new server
Reconnecting to a server
Disconnecting from a server
Removing a server
Navigating a connected server
ConfigurationManagement MBean
LoggingManagement MBean
Runtime Options
ConfigurationFile Options
ServerInformation MBean
UserManagement MBean
VirtualHostManager MBean
Notifications
Managing Queues
Managing Exchanges
Managing Connections

Introduction

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.

Startup & Configuration

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.

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.

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.

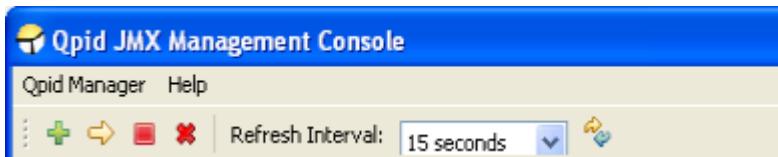
You can download the JMX Remote API 1.0.1_04 Reference Implementation from the Sun website [here](#). The included *jmxremote-1_0_1-bin/lib/jmxremote_optional.jar* file must be added to the *plugins/jmxremote.sasl_1.0.1* folder of the console release (again, in Mac OS X you will need to select 'Show 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.

Managing Server Connections

Main Toolbar

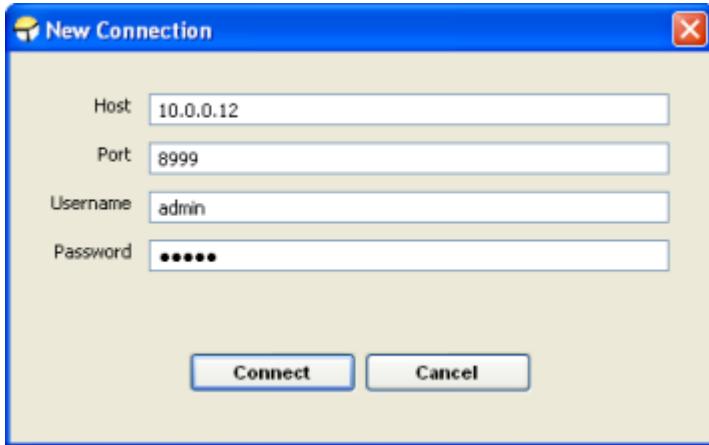
The main toolbar of the console can be seen in the image below. The left most buttons respectively allow for adding a new server connection, reconnecting to an existing server selected in the connection tree, disconnecting the selected server connection, and removing the server from the connection tree.



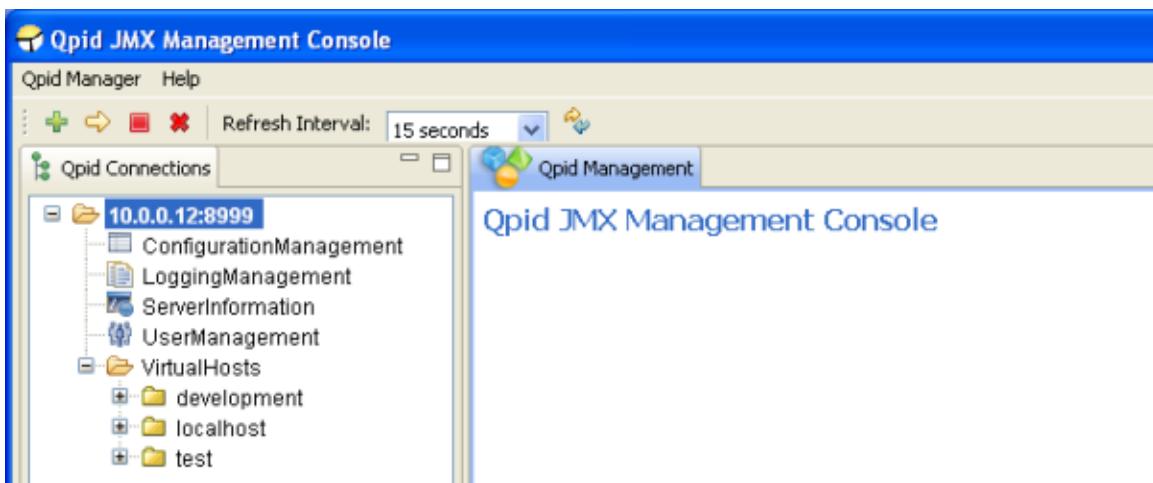
Beside these buttons is a combo for selecting the refresh interval; that is, how often the console requests updated information to display for the currently open area in the main view. Finally, the right-most button enables an immediate update.

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:



Once all the required details are entered, pressing Connect will initiate a connection attempt to the server. If 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.

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.

Disconnecting from a server

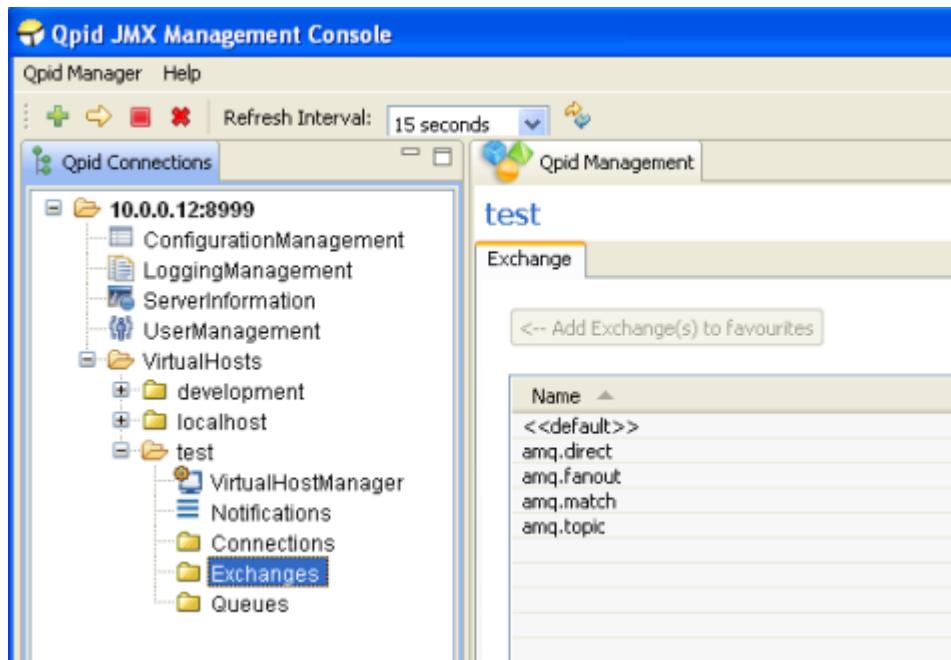
To disconnect from a server, select the connection tree node for the server and press the *Disconnect* toolbar button, or use the *Qpid Manager -> Disconnect* menu option.

Removing a server

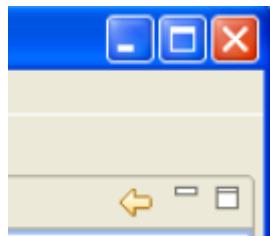
To remove a server from the connection list, select the connection tree node for the server and press the *Remove* toolbar button, or use the *Qpid Manager -> Remove Connection* menu option.

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.



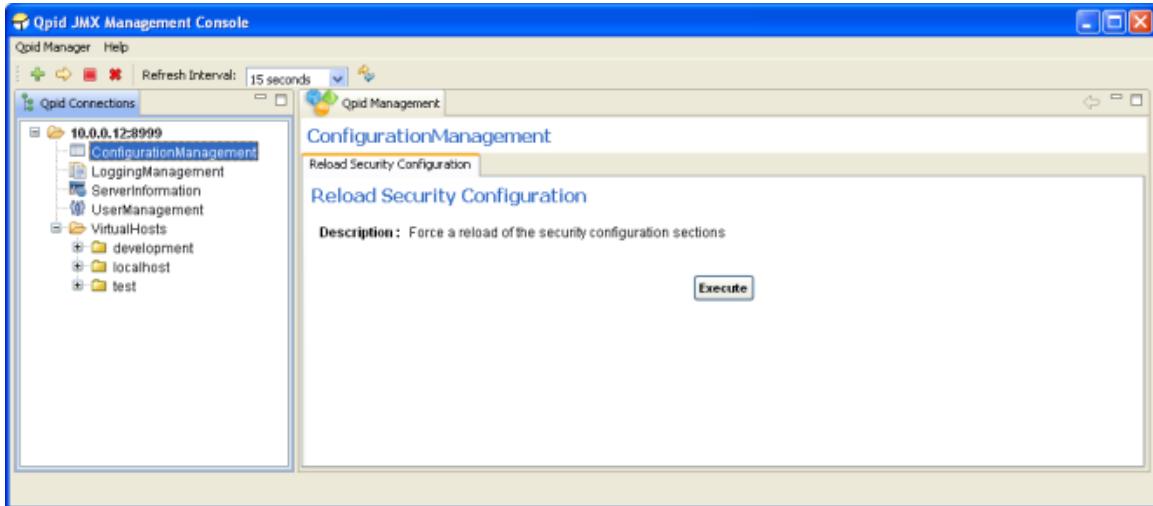
As there may be vast numbers of Queues, Connections, and Exchanges on the server these MBeans are not automatically added to the tree along with the general administration MBeans. Instead, dedicated selection areas are provided to allow users to select which Queue/Connection/Exchange they wish to view 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.

ConfigurationManagement MBean

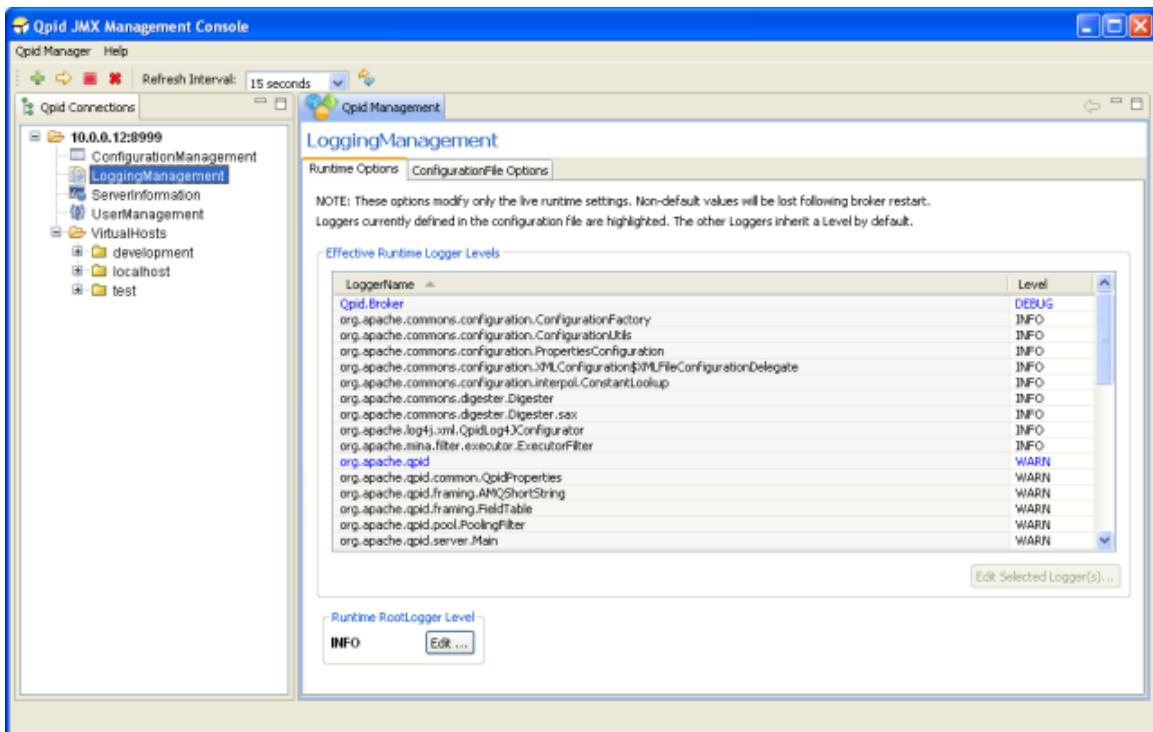
The ConfigurationManagement MBean is available on newer servers, to users with admin level management rights. It offers the ability to perform a live reload of the *Security* sections defined in the main server configuration file (e.g. defaults to: *etc/config.xml*). This is mainly to allow updating 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.



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.

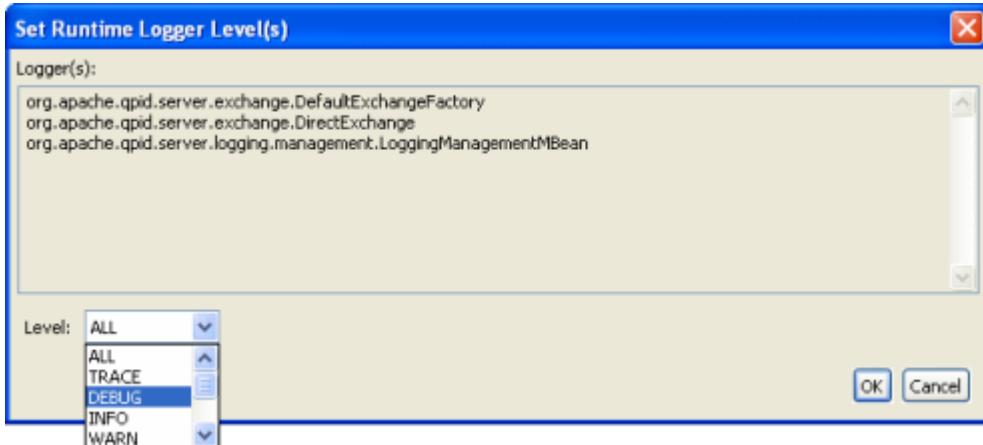
Runtime Options



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 set, to inheriting that of its parent by setting it to an INHERITED level that removes any previously set Level of its own.



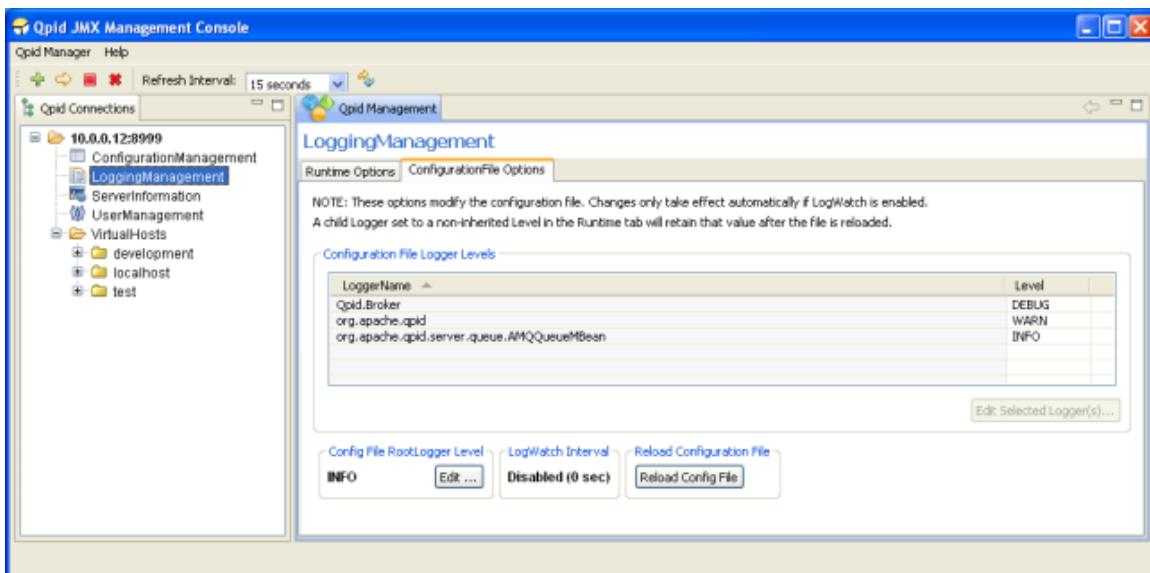
In order to set one of more Loggers to a new Level, they should be selected in the table (or double click an individual Logger to modify it) and the *Edit Selected Logger(s)* button pressed to load the dialog shown above. At this point, any of the available Levels supported by the server can be applied 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.

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.

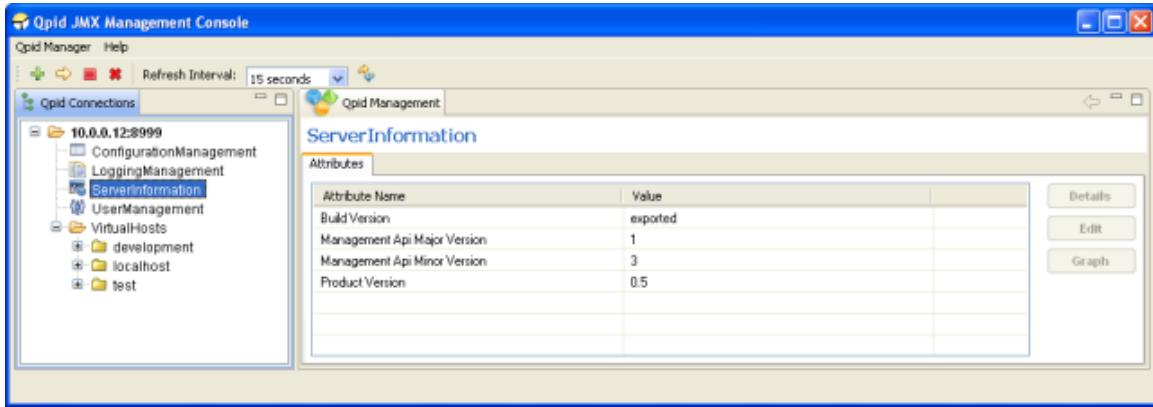
Manipulating the Levels is as on the Runtime Options tab, either double-click an individual Logger entry or select multiple Loggers and use the button to load the dialog to set the new Level.



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.

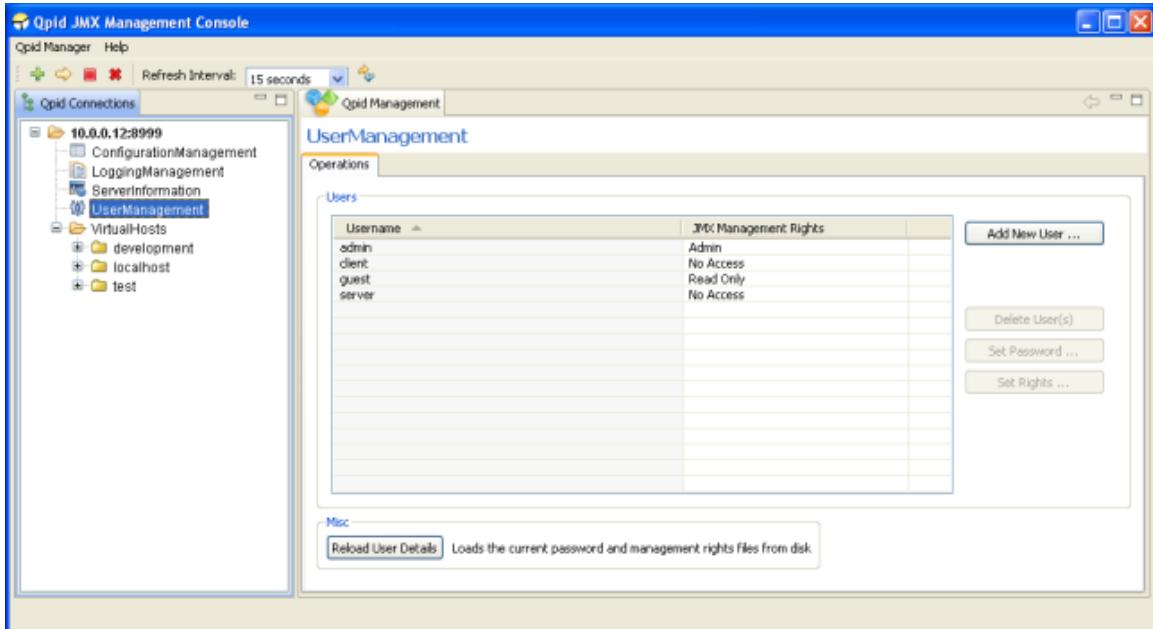
ServerInformation MBean



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.

UserManagement MBean

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



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



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.

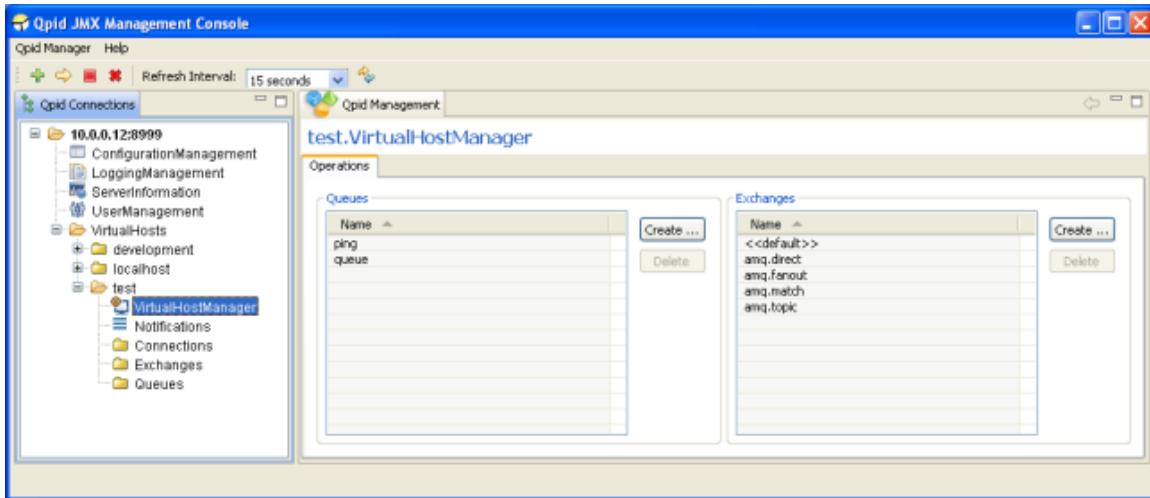
The server caches the user details in memory to aid performance. If may sometimes be necessary to externally modify the password and access right files on disk. In order for these changes to be known to the server without a restart, it must be instructed to reload the file contents. This can be done 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.

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.

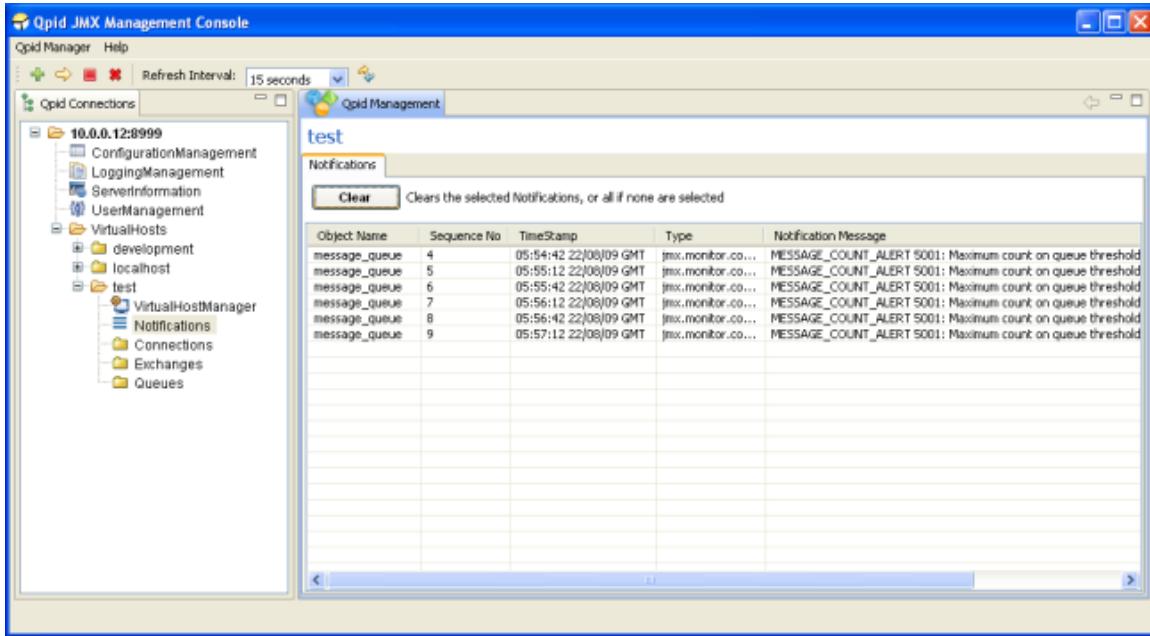
One or more Exchanges may be deleted by selecting them in the table and clicking the *Delete* button. This will unregister all the related channels and Queue bindings then delete the Exchange(s). The console will prompt for confirmation before undertaking the operation.

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

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.

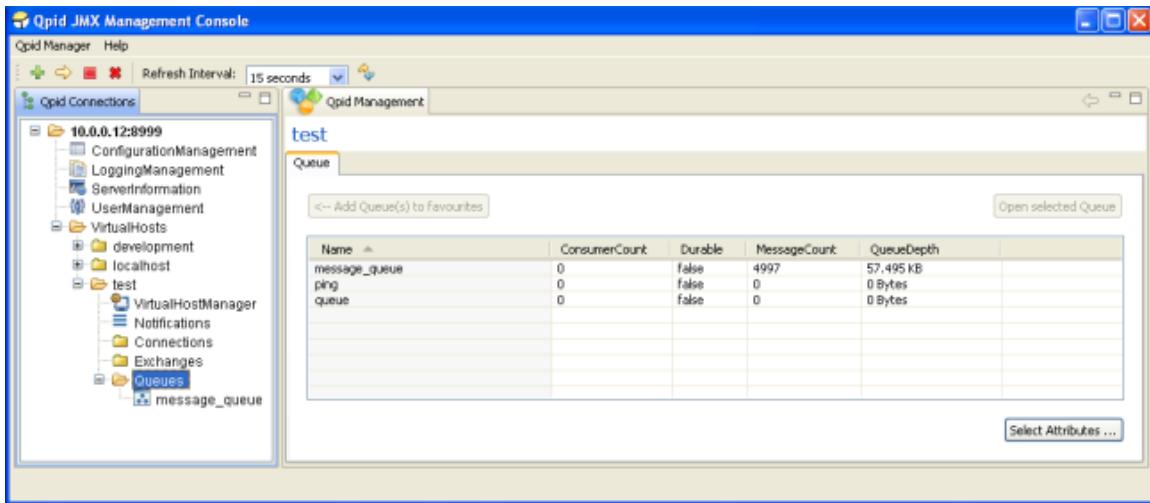


All received Notifications will be displayed until such time as the user removes them, either in this aggregated view, or in the Notifications area of the individual MBean that generated the Notification.

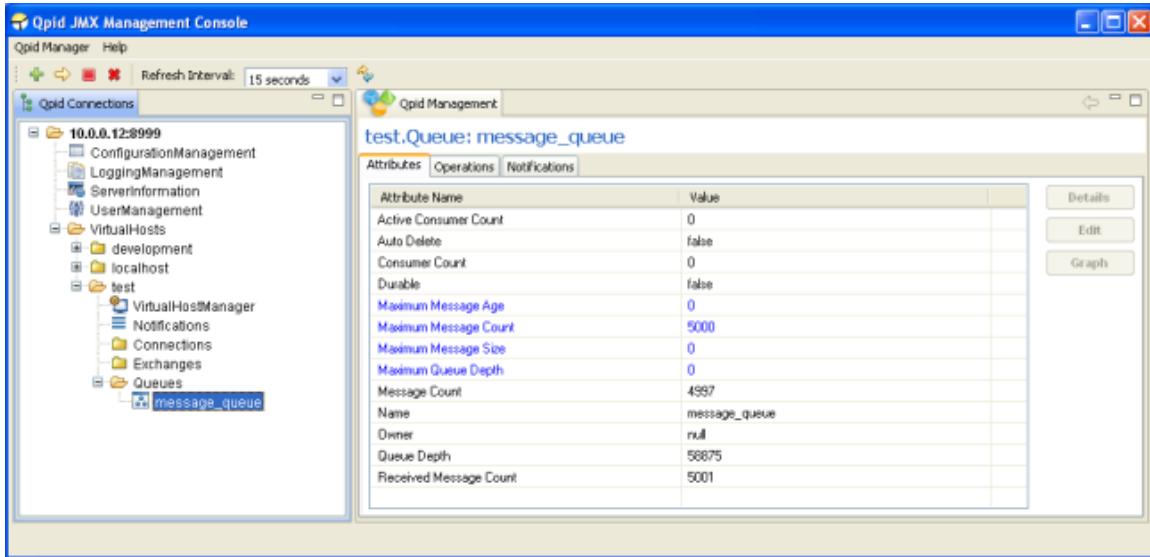
They may be cleared selectively or all at once. To clear particular Notifications, they should be selected in the table before pressing the *Clear* button. To clear all Notifications, simply press the *Clear* button without anything selected in the table, at which point the console will request confirmation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

Managing Queues

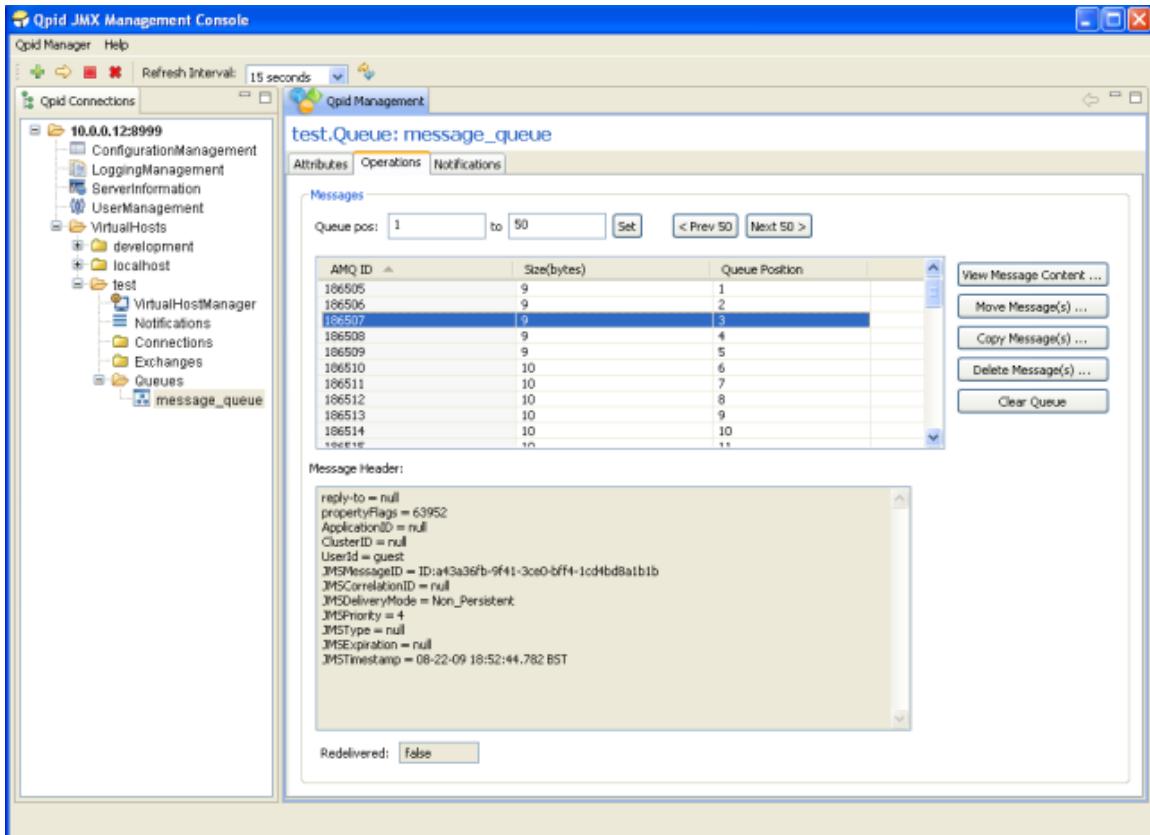
As mentioned in earlier discussion of Navigation, Queue MBeans can be opened either by double clicking an entry in the Queues selection area, or adding a queue to the tree as a favourite and clicking on its tree node. Unique to the Queue selection screen is the ability to view additional attributes beyond 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).



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.



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).



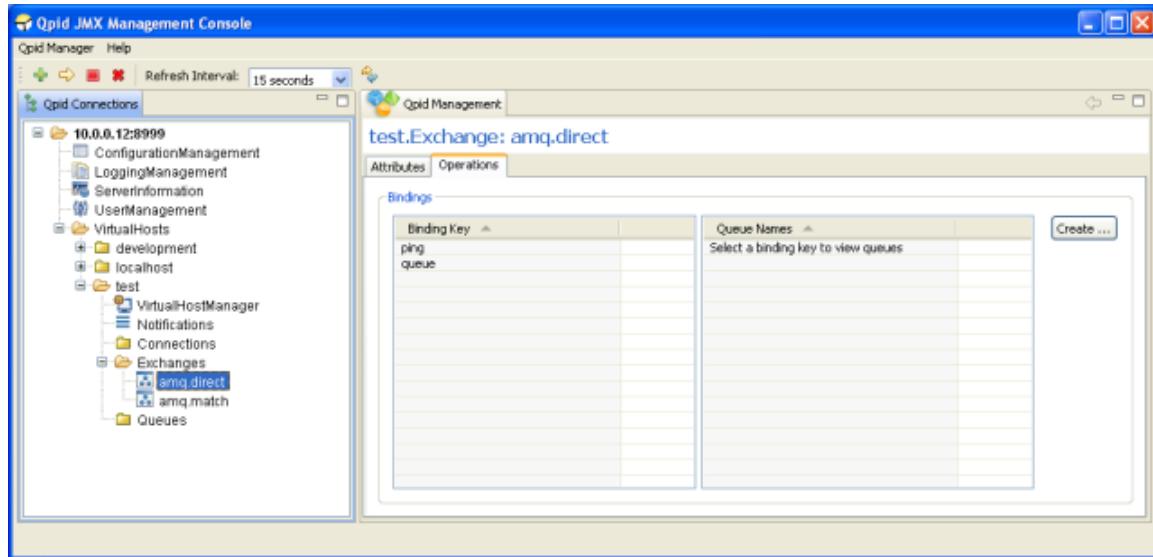
Upon selecting a message in the table, its header properties and redelivery status are updated in the area below the table. Double clicking a message in the table (or using the *View Message Content* button to its right) will open a dialog window displaying the contents of the message.

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.

Managing Exchanges

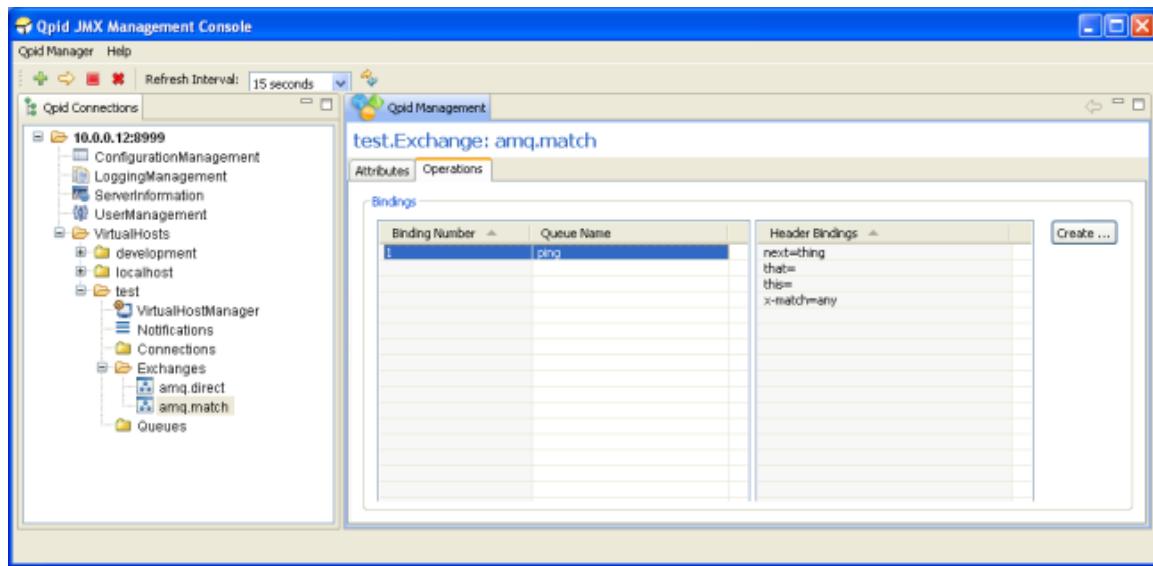
Exchange MBeans are opened for management operations in similar fashion as described for Queues, again showing an Attributes tab initially, with the Operations tab next:



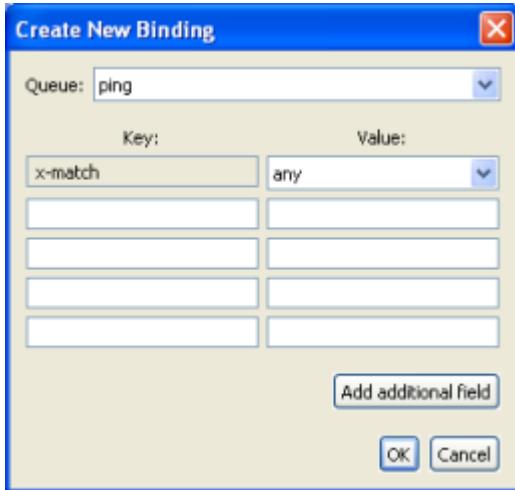
Of the four default Exchange Types (*direct*, *fanout*, *headers*, and *topic*) all but *headers* have their bindings presented in the format shown above. The left table provides the binding/routing keys present in the exchange. Selecting one of these entries in the table prompts the right table to display 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:



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.



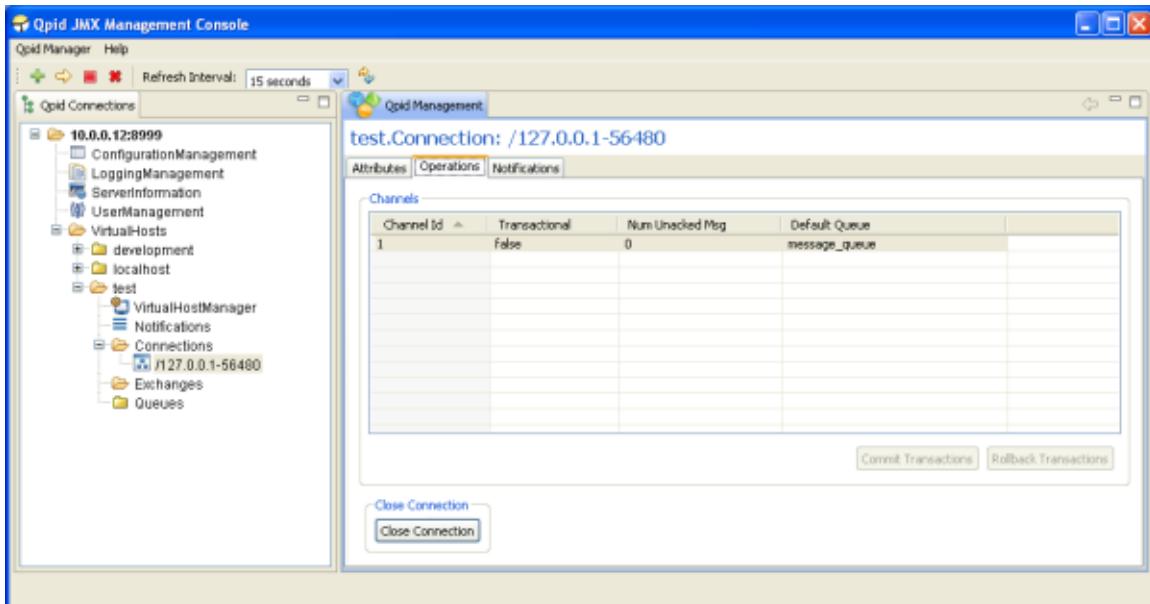
Pressing the *Create* button when managing a *headers* Exchange opens a dialog allowing creation of a new binding, associating an existing Queue with a particular set of header keys and values. The *x-match* key is required, and instructs the server whether to match the binding with incoming messages 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.

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.



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 Committing 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.

Qpid Management Features

Management tool: See our [Management Console](#) page 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 handled?]

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 **Network IO Interface** 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.

Multiple AMQP Version Support

- [Multiple-AMQP Version Support in Qpid](#)
 - 1. Current Generator Status
 - 2. Generator Description
 - 2.1. Overview
 - 2.2. AMQP version model
 - 3. Code Generation
 - 3.1. Difference Modes
 - 3.2 Java Generation
 - 3.3. C++ Generation

Multiple-AMQP Version Support in Qpid

This page describes an effort to allow multiple AMQP versions to be supported in the broker. This implies:

- that a broker will be able to accept a connection from clients requesting a variety of versions of the AMQ protocol;
- The versions to be supported in this manner are determined at compile time;
- A code generator generates the framing classes directly from the XML specification file(s), allowing generated classes to support any

of the supported versions.

- Each of these classes need only the major and minor version numbers at instantiation to represent a frame from that protocol version.

The thinking behind the following generator description is described in [AMQPVersion.1](#). Option 3 (Intelligent Generation) was selected for this implementation.

1. Current Generator Status

The Java generator is more-or-less complete and has been checked into subversion under the gentools directory for initial review. It has **not** been integrated into the Qpid project as yet; I would like to complete the C++ generation first. However, while the C++ work is in progress, the Java generator is available for review and comment. For instructions on installing and running, see the README file in the gentools directory.

2. Generator Description

2.1. Overview

The generator first reads in all the listed specification files and constructs from them a memory model (structure) of the specifications "superimposed" on top of each other so that the differences between them are easy to determine. A domain map (which maps all domain names to their simple domain types) is also constructed.

The generator then uses the model to perform code generation. This is achieved by using templates which contain the static parts of the code (which are simply reproduced) and in which are embedded tokens. These tokens, when encountered in the template, are passed on to the generator class, which then uses the context and model to generate specific the version-dependent sections of the code.

Both of these are discussed in more detail below.

2.2. AMQP verion model

The memory model has two parts - the **domain map** and the **model** (specification structure) itself.

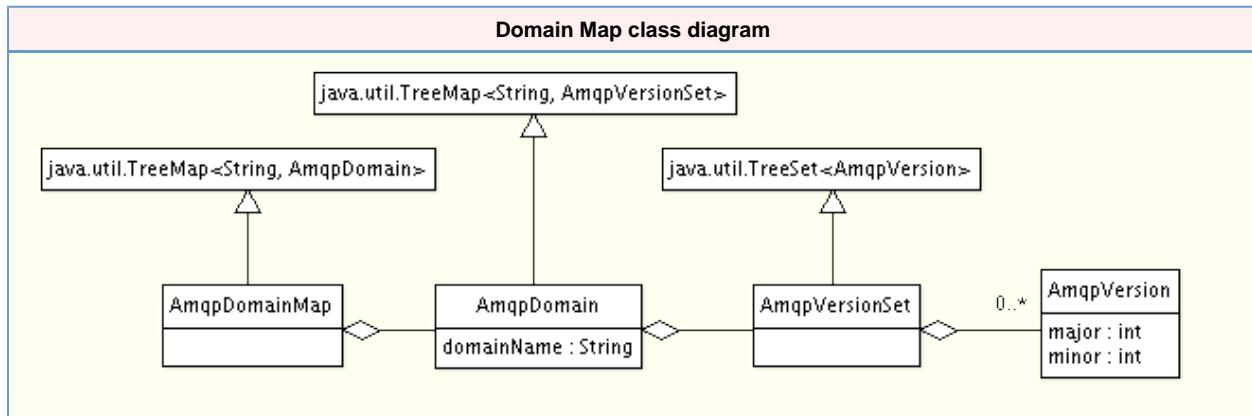
Domain Map

The domain map is a two-level map. The lowest level maps the simple domain types to the AMQP versions in which they are defined. The upper level maps the domain names to the simple domain type.

In the following hypothetical example, the **class-id** domain is changed from *short* in v.0.8 to *long* in v.0.9, then back to *short* in v.0.10. The **queue-type** domain was introduced in v.0.10, while the **redirected** was removed in v0.9.

```
access-ticket --- shortstr --- v[0.8, 0.9, 0.10]
class-id ----- short ----- v[0.8, 0.10]
           +----- long ----- v[0.9]
queue-type ----- shortstr --- v[0.10]
redirected ----- bit ----- v[0.8]
```

A simplified version of the object model is as follows:



Specification Model

The specification model consists of a series of embedded maps in the same logical structure as the XML specification elements themselves: the model contains a map of class maps; class maps contain field and method maps; method maps contain field maps. At the lowest level, there is a map to a set of AMQP versions.

The following illustrates a small portion of a model.

The **Access** class has an index of 30 for versions 0.8 - 0.10. The **request** method has index 10 in v.0.8 and 0.9, but was changed to 20 in

v.0.10. This method has a **active** field in ordinal 1 and a **realm** field in ordinal 0 for all versions. The **realm** field is of domain **path** for version 0.8, but was changed to **domain shortstr** in versions 0.9 and 0.10. **NOTE:** The domains in this model are the domain names, not the domain types. The Domain Map above is used to look up the domain type.

```

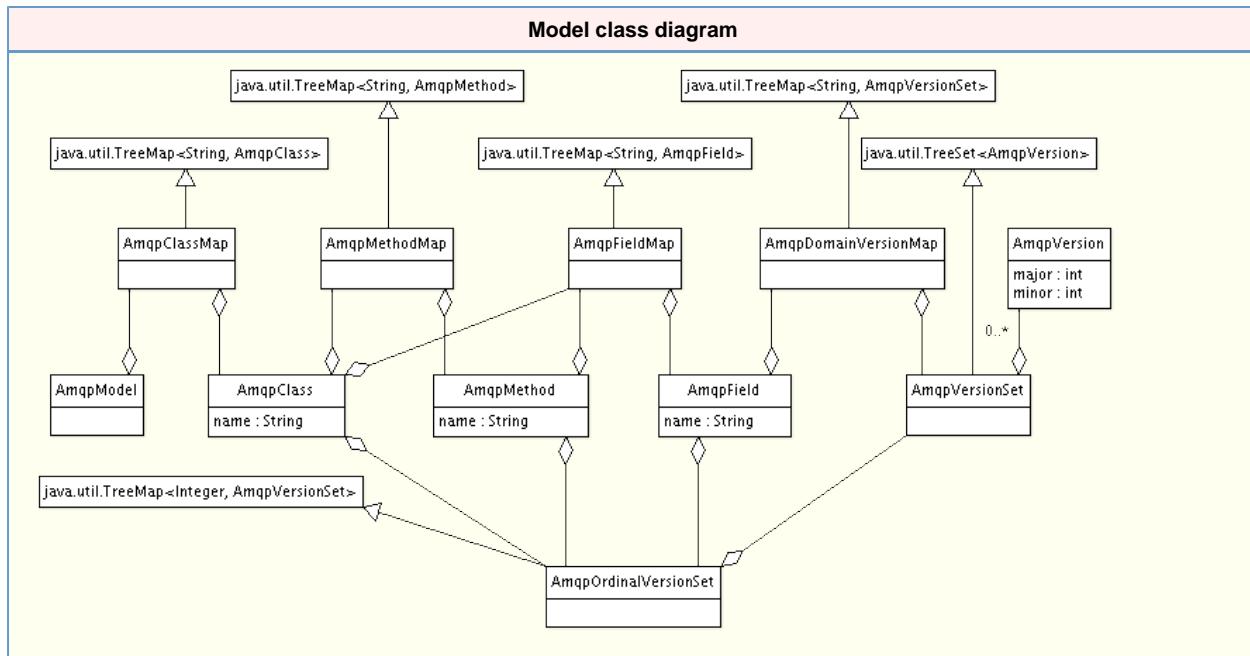
C Access -+--- I 30 ----- V[0.8, 0.9, 0.10]
      +--- M request -+--- I 10 ----- V[0.8, 0.9]
          | +- I 20 ----- V[0.10]
          +-+ F active --- O 1 ----- V[0.8, 0.9, 0.10]
              |           +- D bit ----- V[0.8, 0.9, 0.10]
              +- F realm --- O 0 ----- V[0.8, 0.9, 0.10]
                  +-+ D path ----- V[0.8]
                      +- D shortstr --- V[0.9, 0.10]

```

C = class; M = method; F = field; D = domain name; I = index; O = ordinal; V = version(s)

An **ordinal** is the index number of a field implied by its relative position in the XML specification file. The first field in a class or method has ordinal 0, the second ordinal 1, etc.

A simplified version of the object model is as follows:



Generation

The Generator itself consists of a template parser and large number of code-generating methods for handling the various tokens that are embedded in the templates.

Templates contain three types of tokens:

1. Filename tokens, which determine the name of the file to be generated;
2. Simple Class/Method/Field replacement tokens in which the name of these elements are used to replace the token (e.g. "\${CLASS}\${METHOD}Body" becomes "BasicConsumeBody");
3. List tokens, in which a code snippet is generated once for each item in the list. A second token on the same line determines the code snippet that will be generated. The list tokens cannot be combined or embedded within each other. There are four list tokens:
 - a. %{VLIST} which generates once per version;
 - b. %{CLIST} which generates once per class;
 - c. %{MLIST} which generates once per method;
 - d. %{FLIST} which generates once per field.

3. Code Generation

3.1. Difference Modes

The following changes may take place between one version and the next:

- Addition of classes, methods, fields or domains;
- Deletion of classes, methods, fields or domains;
- Modification of field domains or domain types;
- Modification of the ordinal position of fields;
- Modification of the index of classes or methods;

3.2 Java Generation

3.2.1. *MethodBody classes*

MethodBody classes here.

3.2.2. *PropertyContentHeader classes*

PropertyContentHeader classes here.

3.2.3. *Registry classes*

Registry classes here.

3.3. C++ Generation

Watch this space...

AMQPVersion.1

Back to [Multiple AMQP Version Support](#)

Approach to adding support for multiple AMQP versions into the broker

I had intended to follow approach 2 below, however after some brainstorming, a better and more efficient approach has been proposed. The following is a summary of the possible courses of action.

Kim van der Riet

1. Present status: No AMQP version discrimination.

ADVANTAGES:

- a. Nothing changes in the code, and everyone keeps doing what they are already doing... 😊

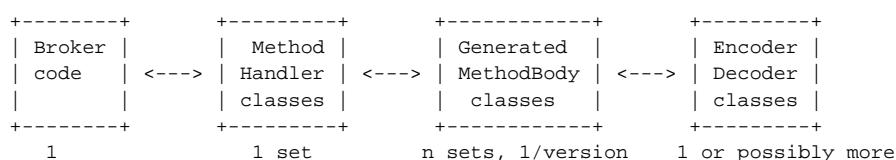
DISADVANTAGES:

- a. There is no easy way of hosting more than one version of AMQP at the same time in the broker.
b. The code that touches the version-specific generated method body classes is scattered all over the place. This makes it difficult to maintain the code if there *is* a change to the AMQP protocol. In order to help conceive the scope of possible protocol changes (outside of any policy the AMQP WG may have in this regard) consider the following possible change types/scopes:

1. A method call (class, method, fields, types) does not change, but the use or purpose of a field changes;
2. A method call changes the type of one or more fields;
3. A method call changes the number of fields (including by removing a field from one version to the next);
4. A method call changes the order of the fields;
5. A totally new method call is added;
6. A method call is removed;
7. A class is added;
8. A class is removed;
9. Any of 5-9 above in such a way as to alter the class/method numbers of unchanged elements (e.g. a class is removed, so class Basic changes from 60 to 50)

2. Initial approach to solving problem: Namespace discrimination.

- a. In this approach, each version of the protocol is generated as before, but into a namespace unique to that version.
b. The `ProtocolSession` objects are made version-aware, and carry the major and minor version numbers of the current session from the `ProtocolInitiation` objects used to start the session.
c. The points in the code where version-specific method body objects and/or calls are made are moved into a single layer - the method handlers. Static function calls are added where necessary to instantiate and/or process these objects. Since the method handlers are version-aware, they would contain the logic to select the correct namespace.
d. The generated code would contain additional methods which allow the fields within any method body to be accessed or set using a parametrized function call. This would aid the coding of the version-sensitive logic used in the method handlers.



ADVANTAGES:

- a. Allows multiple AMQP versions to be used simultaneously in the broker.
- b. Isolates the scope of version-specific code to the method handlers, where hand-coded logic specific to the needs of individual AMQP versions may reside. This aids maintainability in the event of additional versions being added/versions being changed.
- c. Forces the code to choose a specific AMQP version to use the `MethodBody` classes.
- d. Works using existing code-generation system (XSLT).

DISADVANTAGES

- a. Code duplication: Since in most AMQP version changes, 90+% of the specification remains unchanged, the generated classes will contain mostly exact duplicates from version to version.
- b. The ideal system of version fallback (in which default behavior is to use the latest class and to hand-code logic only where differences exist) does not operate. Each version must be handled separately whether identical or not.
- c. The code using the parametrized function calls is more complex and not as easy to understand as direct calls.

3. Revised approach: Intelligent generation

- a. In this approach, the limitations of the existing XSLT code generation are removed. We assume that we can compare one version of the AMQP specification to another, and generate code for the latest version and code to handle only the differences between the latest and each earlier version. To achieve this, we conceptually refactor the XML specification file so that the version information is contained inside the elements instead of outside them. Then each version of the spec is added cumulatively to create a complete map of the specification. For example, assume the following simple example:

```
AMQP 1.0:  
Basic.Consume(Ticket ticket, Queue queue, ConsumerTag consumer_tag, NoLocal no_local, NoAck  
no_ack, bool exclusive, bool nowait);  
AMQP 1.2:  
Basic.Consume(Ticket ticket, Queue queue, ConsumerTag consumer_tag, int no_ack, bool exclusive,  
bool nowait, NoLocal no_local, int priority);
```

would be refactored to something like (abbreviated):

```

<AMQP>
<class name="Basic">
    <version major="1" minor="0" num="60"/>
    <version major="1" minor="2" num="50"/>
    <method name="Consume">
        <version major="1" minor="0" num="20"/>
        <version major="1" minor="2" num="20"/>
        <field name="ticket" type="Ticket">
            <version major="1" minor="0" num="0"/>
            <version major="1" minor="2" num="0"/>
        </field>
        <field name="queue" type="Queue">
            <version major="1" minor="0" num="1"/>
            <version major="1" minor="2" num="1"/>
        </field>
        <field name="consumer_tag" type="ConsumerTag">
            <version major="1" minor="0" num="2"/>
            <version major="1" minor="2" num="2"/>
        </field>
        <field name="no_local" type="NoLocal">
            <version major="1" minor="0" num="3"/>
            <version major="1" minor="2" num="6"/>
        </field>
        <field name="no_ack" type="NoAck">
            <version major="1" minor="0" num="4"/>
        </field>
        <field name="no_ack" type="int">
            <version major="1" minor="2" num="3"/>
        </field>
        <field name="exclusive" type="bool">
            <version major="1" minor="0" num="5"/>
            <version major="1" minor="2" num="4"/>
        </field>
        <field name="nowait" type="bool">
            <version major="1" minor="0" num="6"/>
            <version major="1" minor="2" num="5"/>
        </field>
        <field name="priority" type="int">
            <version major="1" minor="2" num="7"/>
        </field>
    </method>
</class>
</AMQP>

```

This would result in the generation of only a single MethodBody class, and since all instances of this class are version-aware, each knows how to handle calls for a given version (pseudo-java):

```

class BasicConsumeBody extends AMQPMethodBody
{
    public int _major, _minor;
    public BasicConsumeBody(int major, int minor) {_major = major; _minor = minor;}
    public int getAMQPClass(){if (major == 1 && minor == 0) return 60; else return 50;} // the
    class number has changed!
    public int getAMQPMethod(){return 20;}

    public Ticket getTicket() {...}
    public Queue getQueue() {...}
    public ConsumerTag getConsumerTag() {...}
    public NoLocal getNoLocal() {...}
    public T getNoAck(Class<T>) {if (major == 1 && minor == 0)...} // NoAck changes type
    public boolean getExclusive() {...}
    public boolean getNoWait() {...}
    public int getPriority() throws Exception {if (major != 1 || minor != 2) throw...} // Priority
    field is in 1.2 only
    // also for setXXX() methods...
}

```

Static functions would require major and minor to be passed to be able to operate.

a.#2 As before, the ProtocolSession objects are made version-aware, and carry the major and minor version numbers of the current session from the ProtocolInitiation objects used to start the session.

- b. As before, the points in the code where version-specific method body objects and/or calls are made are moved into a single layer - the method handlers.
- c. If different versions of a particular method have additional and/or different parameters, the generated class may be thought of as a cumulative collection of all these parameters/types. Since the object is AMQP version-aware, it is possible for the handlers to know which fields to use and/or how to initialize/handle the unused fields for any call which addresses less than all the available fields. Possible courses of action include ignoring the unused field, or initializing it with a meaningful default. Making the field members of the class public would afford this flexibility, and the get/set methods could simply be viewed as convenience methods. Where the type of a field changes, the different fields would require some kind of name-mangling to indicate their type.

ADVANTAGES

- a. Only a single class for each method body
- b. Single get/set method for each field
- c. Common code is simple and remains unchanged from version to version
- d. Where a field type changes, a `Class<T>` type method is used; this will break all areas in the code which used to use this method, and conveniently indicate where hand-coded logic is required.
- e. Where there is a field that is unique to a subset of versions, auto-generated code will throw an exception if it is invoked from a session of the wrong version. The addition of the exception will break the compilation (i.e. must be caught), and will indicate places in the code where hand-coded logic will be required.
- f. The existing handler code can remain substantially unchanged.

DISADVANTAGES

- a. Complexity...
- b. This generation is beyond the capabilities of XSLT (or would be inordinately difficult), and we would need to switch to either Python (or Jython) or a purpose-written Java generator to do the job.
- c. It is not clear from the interface what field is valid in what version. There has been some discussion on name-mangling to solve this, but the general dislike for name-mangling in general and the complexities it brings seem to outweigh any advantages...

Qpid Java FAQ

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.

Contents

- Purpose
- Contents
- Getting Started
 - What is Qpid
 - Why am I getting a ConfigurationException at broker startup
 - InvocationTargetException
 - Cannot locate configuration source null/virtualhosts.xml
 - How do I run the Qpid broker
 - How can I create a connection using a URL
 - How do I represent a JMS Destination string with QPID
 - Queues
 - Topics
 - How do I connect to the broker using JNDI
 - I'm using Spring and Weblogic - can you help me with the configuration for moving over to Qpid
 - How do I configure the logging level for Qpid
 - How can I configure my application to use Qpid client logging
 - How can I configure the broker
 - What ports does the broker use?
 - How can I change the port the broker uses at runtime
 - What command line options can I pass into the qpid-server script
 - Command Line Options
 - How do I authenticate with the broker and What user id & password should I use
 - How do I create queues that will always be instantiated at broker startup
 - How do I create queues at runtime
 - How do I tune the broker
 - Where do undeliverable messages end up
 - Can I configure the name of the Qpid broker log file at runtime
 - Setting a prefix or suffix
 - Including the PID
 - My client application appears to have hung
 - How do I contact the Qpid team
 - How can I change a user's password while the broker is up
 - How do I know if there is a consumer for a message I am going to send
 - How do I use an InVM Broker for my own tests
 - How can I inspect the contents of my MessageStore
 - Why are my transient messages slower than expected
 - Why does my producer fill up the broker with messages
 - The broker keeps throwing an OutOfMemory exception
 - Why am I getting a broker side exception when I try to publish to a queue or a topic

- Why is there a lot of AnonymousIoService threads
- "unable to certify the provided SSL certificate using the current SSL trust store" when connecting the Management Console to the broker.
- Client keeps throwing 'Server did not respond in a timely fashion' [error code 408: Request Timeout].
- Can a use TCP_KEEPMALIVE or AMQP heartbeating to keep my connection open

Getting Started

- Can you help me get started with Qpid?
- How do I install the Qpid broker ?
- Where can I find more information?
- Qpid JMX Management Console FAQ

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 [3rd Party Libraries](#) 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

Why am I getting a ConfigurationException at broker startup

InvocationTargetException

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

```
Error configuring message broker: org.apache.commons.configuration.ConfigurationException:  
java.lang.reflect.InvocationTargetException  
2008-09-26 15:14:56,529 ERROR [main] server.Main (Main.java:206) - Error configuring message  
broker: org.apache.commons.configuration.ConfigurationException:  
java.lang.reflect.InvocationTargetException  
org.apache.commons.configuration.ConfigurationException:  
java.lang.reflect.InvocationTargetException  
at  
org.apache.qpid.server.security.auth.database.ConfigurationFilePrincipalDatabaseManager.initialisePri  
  
at  
org.apache.qpid.server.security.auth.database.ConfigurationFilePrincipalDatabaseManager.initialisePri  
  
at  
org.apache.qpid.server.security.auth.database.ConfigurationFilePrincipalDatabaseManager.<init>(Config  
  
at  
org.apache.qpid.server.registry.ConfigurationFileApplicationRegistry.initialise(ConfigurationFileAppl  
  
at org.apache.qpid.server.registry.ApplicationRegistry.initialise(ApplicationRegistry.java:79)  
at org.apache.qpid.server.registry.ApplicationRegistry.initialise(ApplicationRegistry.java:67)  
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.java:25)  
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.java:25)  
at java.lang.reflect.Method.invoke(Method.java:597)  
at  
org.apache.qpid.server.security.auth.database.ConfigurationFilePrincipalDatabaseManager.initialisePri
```

.. 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>
```

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 QPID-431

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

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 parameters 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 [Getting Started](#) page.

How can I create a connection using a URL

Please see the [Connection URL Format](#) documentation.

How do I represent a JMS Destination string with QPID

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 [BindingURLFormat](#) is described in more detail on its own page.

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.

How do I connect to the broker using JNDI

see [How to Use JNDI](#)

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](#) which shows the config for Qpid side by side with Weblogic. HtH !

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.

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.

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 [Qpid Design - Configuration](#)

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.

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 [this FAQ entry](#).

For more detailed information on configuration, please see [Qpid Design - Configuration](#)

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

The following command line options are available:

Command Line Options

The following options are available:

| 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 | import | 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 |

How do I authenticate with the broker and What user id & password should I use

You should login as user guest with password guest

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 this 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

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.

How do I tune the broker

There are a number of tuning options available, please see the [How to Tune M3 Java Broker Performance](#) page for more information.

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.

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 archive 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.

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) it could look something like qpidMyApp.log

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**

cloak: Missing opening cloak.

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 to complete. 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 [QPID-429](#)

How do I contact the Qpid team

For general questions, please subscribe to the users@qpid.apache.org mailing list ([archive](#)).

For development questions, please subscribe to the dev@qpid.apache.org mailing list ([archive](#)).

More details on these lists are available on our [Mailing Lists](#) page.

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

You can do this via the [Qpid JMX Management Console](#). 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 [Qpid JMX Management Console User Guide](#)

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.

How do I use an InVM Broker for my own tests

I would take a look at the `testPassiveTTL` in [TimeToLiveTest](#)

The `setUp` and `tearDown` methods show how to correctly start up a broker for InVM testing. If you write your tests using a file for the JNDI you can then very easily swap between running your tests InVM and against a real broker.

See our [JNDI How to page](#) on how to configure it

Basically though you just need to set two System Properties:

```
java.naming.factory.initial = org.apache.qpid.jndi.PropertiesFileInitialContextFactory  
java.naming.provider.url = <your JNDI file>
```

and call `getInitialContext()` in your code.

You will of course need to have the broker libraries on your class path for this to run.

How can I inspect the contents of my MessageStore

There are two possibilities here:

1) The management console can be used to interrogate an active broker and browse the contents of a queue. See the [Qpid JMX Management Console](#) page for further details.

2) The [MessageStore Tool](#) can be used to inspect the contents of a persistent message store. Note: this can currently only be used when the broker is offline.

Why are my transient messages slower than expected

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.

Why does my producer fill up the broker with messages

The Java broker does not currently implement producer flow control. Publishes are currently asynchronous, so there is no ability to rate limit this automatically. While this is something which will be addressed in the future, it is currently up to applications to ensure that they do not publish faster than the messages are being consumed for significant periods of time.

The broker keeps throwing an OutOfMemory exception

The broker can no longer store any more messages in memory. This is particularly 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 [Java Environment Variables](#) for details of changing the memory settings.

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 tyo'd the exchange type in your queue or topic declaration. Open your virtualhosts.xml and check that the

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

is set to amq.direct for the <queue/> element you're trying to publish to.

```
2009-01-12 15:26:27,957 ERROR [pool-11-thread-2] protocol.AMQMinaProtocolSession  
(AMQMinaProtocolSession.java:365) - Unexpected exception while processing frame. Closing  
connection.  
java.lang.NullPointerException  
    at  
org.apache.qpid.server.security.access.PrincipalPermissions.authorise(PrincipalPermissions.java:398)  
    at org.apache.qpid.server.security.access.plugins.SimpleXML.authorise(SimpleXML.java:302)  
    at  
org.apache.qpid.server.handler.QueueBindHandler.methodReceived(QueueBindHandler.java:111)  
    at  
org.apache.qpid.server.handler.ServerMethodDispatcherImpl.dispatchQueueBind(ServerMethodDispatcherImpl  
    at org.apache.qpid.framing.amqp_8_0.QueueBindBodyImpl.execute(QueueBindBodyImpl.java:167)  
    at org.apache.qpid.server.state.AMQStateManager.methodReceived(AMQStateManager.java:204)  
    at  
org.apache.qpid.server.protocol.AMQMinaProtocolSession.methodFrameReceived(AMQMinaProtocolSession.jav  
    at org.apache.qpid.framing.AMQMethodBodyImpl.handle(AMQMethodBodyImpl.java:93)  
    at  
org.apache.qpid.server.protocol.AMQMinaProtocolSession.frameReceived(AMQMinaProtocolSession.java:235)  
    at  
org.apache.qpid.server.protocol.AMQMinaProtocolSession.dataBlockReceived(AMQMinaProtocolSession.java:  
    at  
org.apache.qpid.server.protocol.AMQPFastProtocolHandler.messageReceived(AMQPFastProtocolHandler.java:  
    at  
org.apache.mina.common.support.AbstractIoFilterChain$TailFilter.messageReceived(AbstractIoFilterChain  
    at  
org.apache.mina.common.support.AbstractIoFilterChain.callNextMessageReceived(AbstractIoFilterChain.ja  
    at  
org.apache.mina.common.support.AbstractIoFilterChain.access$1200(AbstractIoFilterChain.java:54)  
    at  
org.apache.mina.common.support.AbstractIoFilterChain$EntryImpl$1.messageReceived(AbstractIoFilterChai  
    at org.apache.qpid.pool.PoolingFilter.messageReceived(PoolingFilter.java:371)  
    at  
org.apache.mina.filter.ReferenceCountingIoFilter.messageReceived(ReferenceCountingIoFilter.java:96)  
    at  
org.apache.mina.common.support.AbstractIoFilterChain.callNextMessageReceived(AbstractIoFilterChain.ja  
    at  
org.apache.mina.common.support.AbstractIoFilterChain.access$1200(AbstractIoFilterChain.java:54)  
    at  
org.apache.mina.common.support.AbstractIoFilterChain$EntryImpl$1.messageReceived(AbstractIoFilterChai  
    at  
org.apache.mina.filter.codec.support.SimpleProtocolDecoderOutput.flush(SimpleProtocolDecoderOutput.ja  
    at  
org.apache.mina.filter.codec.QpidProtocolCodecFilter.messageReceived(QpidProtocolCodecFilter.java:174  
    at  
org.apache.mina.common.support.AbstractIoFilterChain.callNextMessageReceived(AbstractIoFilterChain.ja  
    at  
org.apache.mina.common.support.AbstractIoFilterChain.access$1200(AbstractIoFilterChain.java:54)  
    at  
org.apache.mina.common.support.AbstractIoFilterChain$EntryImpl$1.messageReceived(AbstractIoFilterChai  
    at org.apache.qpid.pool.Event$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$Worker.runTask(ThreadPoolExecutor.java:885)  
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:907)  
    at java.lang.Thread.run(Thread.java:619)
```

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.

"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 [Management Console Security](#) for more details.

Client keeps throwing 'Server did not respond in a timely fashion' [error code 408: Request Timeout].

Certain operations wait for a response from the Server. One such operations is commit. If the server does not respond to the commit request within a set time a Request Timeout [error code: 408] exception is thrown (Server did not respond in a timely fashion). This is to ensure that a server that has hung does not cause the client process to be come unresponsive.

However, it is possible that the server just needs a long time to process a give request. For example, sending a large persistent message when using a persistent store will take some time to a) Transfer accross the network and b) to be fully written to disk.

These situations require that the default timeout value be increased. A client [System Properties](#) 'amqj.default_syncwrite_timeout' can be set on the client to increase the wait time. The default in 0.5 is 30000 (30s).

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

See [Configure Broker and Client Heartbeating](#)

Qpid Java How To

Collection of How Tos

- Add New Users
- Configure ACLs
- Configure Broker and Client Heartbeating
- Configure Java Qpid to use a SSL connection.
- Configure Log4j CompositeRolling Appender
- Configure Operational Status Logging
- Configure the Broker via config.xml
- Configure the Virtual Hosts via virtualhosts.xml
- Debug using log4j
- Firewall Configuration
- How to Tune M3 Java Broker Performance
- How to Use JNDI
- Interact with a JMX MBean
- Qpid Java Build How To
- Split configuration files
- Tune Broker and Client Memory Usage
- Use Last Value Queues (LVQ)
- Use Priority Queues
- Use Producer Flow Control

Add New Users

The Qpid Java Broker has a single reference source ([PrincipalDatabase](#)) 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.

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.

- Plain
- Base64MD5

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.

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.PlainPasswordFilePrincipalDatabase</class>
        <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>
```

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:X03M01qnZdYdgfeuILPmQ==

```

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.

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 [htpasswd](#) 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 [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 [Qpid JMX Management Console User Guide](#)).

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 [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.

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 a client that understands CRAM-MD5 will work correctly with a Plain and Base64MD5 PrincipalDatabase.

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

For details of SASL support see [Qpid Interoperability Documentation](#)

Configure ACLs

Configure ACLs

Specification

- Version 1
- Version 2

C++ Broker

The C++ broker supports Version 2 of the ACLs

Java Broker

- Configuration Guide for Version 1 ACLs
- Support for Version 2 specification is in progress.

Java XML ACLs

Java XML ACLs

This page documents version 1 of Qpid ACLs that was implemented only in the Java broker.

- Java XML ACLs
 - Specification
 - XML Format
 - User Guide (SimpleXML)
 - Permission Limitations
 - Enabling XML ACLs
 - ACL Configuration
 - Background
 - ACCESS_CONTROL_LIST Section
 - PUBLISH Section
 - CONSUME Section
 - CREATE Section
 - ACCESS Section (since Qpid 0.6)
 - Durable topic subscriptions
 - Known Issues
 - Granting temporary queue and named queue consume rights

Specification

The XML ACL focus was to take a business style focus to access rather than the individual AMQP method level. As a result we have the following permissions:

- CONSUME
- PUBLISH
- CREATE
- ACCESS

- BIND
- UNBIND
- DELETE
- PURGE

XML Format

DTD TBC

User Guide (*SimpleXML*)

The XML ACLs have been implemented as per the [ACLPlugin] design, *SimpleXML*. Currently this class is only configurable via the main broker configuration file, this means that all the ACL configuration must be included in the main configuration file.

Permission Limitations

Only the first four permissions, CONSUME, PUBLISH, CREATE and ACCESS (since Qpid 0.6) have been implemented. An oversight in the original design resulted in the inability to specify negative permissions. As a result permission can only be granted to users and not taken away.

Enabling XML ACLs

To enable the ACLs the security access class in the main broker configuration needs to be updated as follows:

```
...
<security>
    <access>
        <class>org.apache.qpid.server.security.access.plugins.SimpleXML</class>
    </access>
...

```

This tells the broker that it should use the *SimpleXML* class to perform access control. When the broker starts up the *SimpleXML* class will look in the the `<security>` subsection for each virtualhost for the required ACLs.

ACL Configuration

Background

The configuration is described in reference to an example configuration used in a request/response application. In this example the 'client' creates a temporary queue and sends a request to a known queue which the 'server' application is processing. The 'server' then sends a response to the specified temporary queue which the 'client' can read. The ACLs have been configured such that the 'server' cannot create additional queues other than its process queue and the 'client' is only allowed to create temporary queues.

ACCESS_CONTROL_LIST Section

The ACL configuration lives inside the `<access_control_list>` section, inside the `<security>` subsection of each virtualhost configuration.

```
...
<security>
    <access_control_list>
        <!-- This section grants virtualhost-level access to the specified users, giving
            giving them full permissions to all artifacts in the containing virtualhost -->
        <access>...</access>

        <!-- This section grants publish rights to an exchange + routing key pair -->
        <publish>...</publish>

        <!-- This section grants users the ability to consume from the broker -->
        <consume>...</consume>

        <!-- This section grants clients the ability to create queues and exchanges -->
        <create>...</create>
    </access_control_list>
...

```

This gives the basic structure for the configuration the contents of each section naturally depend on what permissions are needed.

PUBLISH Section

This section allows the granting of permission for *Publishers* to send messages. Controls have been implemented to allow the publication of messages limited by Exchange to:

- specified routing keys.
- partial matching routing keys. Using * to match the end of a routing key.

Here the 'client' users is only give rights to PUBLISH messages using the key 'example.RequestQueue'. The 'server' user is allowed to publish to 'tmp_*' and 'TempQueue*' keys. The reason there are two values here is due to changes in the naming of temporary queues during the example's development. However, what occurs here is that the 'server' is granted permission to publish messages to any routing key that begins with 'tmp_' or 'TempQueue', the '*' matching is only completed at the end of the key so entries such as 'Special*Key' are not allowed.

Whilst not shown here multiple `<user>` values can be specified in the `<users>` section.

Remember that the `routing_key` value in the Java broker is the same as the queue name (correct at release of M4) for the amq.direct exchange. For topic exchanges the `routing_key` is the topic name that a *Publisher* uses to send messages.

```

<publish>
  <exchanges>
    <exchange>
      <!-- This is the name of the exchange to limit publication to. -->
      <name>amq.direct</name>
      <routing_keys>

        <!-- Allow clients to publish requests -->
        <routing_key>
          <value>example.RequestQueue</value>
          <users>
            <user>client</user>
          </users>
        </routing_key>

        <!-- Allow the processor to respond to a client on their Temporary Topic -->
        <routing_key>
          <value>tmp_*</value>
          <users>
            <user>server</user>
          </users>
        </routing_key>
        <routing_key>
          <value>TempQueue*</value>
          <users>
            <user>server</user>
          </users>
        </routing_key>
      </routing_keys>

    </exchange>
  </exchanges>
</publish>

```

CONSUME Section

This section allows the granting of permissions to *Consumers*. There are two formats the `<queue>` entry can take:

- Users can be granted permission to a named queue by the use of the `<name>` field.
- Users can be granted permission to *ALL* temporary queues with the addition of the `<temporary/>` key.

These two formats can be combined to allow the consumption from a named queue and temporary queues. However, care must be taken if using multiple `<queue>` entries as access to temporary queues will be defined by the last `<queue>` definition. **This is a known issue.**

```

<!-- This section grants users the ability to consume from the broker -->
<consume>
    <queues>

        <!-- Allow the clients to consume from their temporary queues-->
        <queue>
            <temporary/>
            <users>
                <user>client</user>
            </users>
        </queue>

        <!-- Only allow the server to consume from the Request Queue-->
        <queue>
            <name>example.RequestQueue</name>
            <users>
                <user>server</user>
            </users>
        </queue>

    </queues>
</consume>

```

CREATE Section

This section allows the granting of permissions to create new queues as used by *Consumers*. When a consumer is created it makes a request to create the queue for the consumer. This means that all your consumers must also be allowed to create the queue they are going to consume from or they will fail to create.

The `<create>` section contains a number of fields that can be present:

```

<temporary/>
<name>
<users>
<exchanges>

```

The first three behave as in `<consume>` limiting the list of users to a named queue or all temporary queues. The additional `<exchanges>` element contains a number of `<exchange>` entries, this entry contains a list of users that are limited to using only that exchange for the given queue. This is used in the example below to limit the user 'client' to only be able to create temporary queues on the 'amq.direct' exchange.

NOTE: This section also suffers from the same issue as `<consume>` with regard to the `<temporary/>` keyword. See the known [issue](#) for more details.

```

<!-- This section grants clients the ability to create queues and exchanges -->
<create>
    <queues>
        <!-- Allow clients to create temporary queues-->
        <queue>
            <temporary/>
            <exchanges>
                <exchange>
                    <name>amq.direct</name>
                    <users>
                        <user>client</user>
                    </users>
                </exchange>
            </exchanges>
        </queue>
        <!-- Allow the server to create the Request Queue-->
        <queue>
            <name>example.RequestQueue</name>
            <users>
                <user>server</user>
            </users>
        </queue>

    </queues>
</create>

```

ACCESS Section (since Qpid 0.6)

This section allows granting virtualhost-level access permissions to specific users, giving them full permissions to all artifacts within the virtualhost irrespective of any rights assigned in the CREATE, CONSUME, and PUBLISH sections outlined above. This allows granting only certain users full access to certain virtualhosts.

The `<access>` section contains a `<users>` subsection, with a list of individual `<user>` elements:

```
<!-- This section grants virtualhost-level access to the specified users, giving
     giving them full permissions to all artifacts in the containing virtualhost -->
<access>
  <users>
    <user>admin</user>
  </users>
</access>
```

Durable topic subscriptions

Qpid implements durable topic subscriptions as a persistent queue bound to the topic exchange. This queue is named `<clientid>:<subscriptionname>`. To allow a JMS durable topic subscription it's necessary to allow queue creation and consumption for the user. eg:

```
<consume>
  <queues>
    <queue>
      <name>clientid:subscriptionName</name>
      <users>
        <user>testuser</user>
      </users>
    </queue>
  </queues>
</consume>

<create>
  <queues>
    <queue>
      <name>clientid:subscriptionName</name>
      <users>
        <user>testuser</user>
      </users>
    </queue>
  </queues>
</create>
```

Known Issues

Granting temporary queue and named queue consume rights

When defining a `<queue>` entry the existence of the `<temporary>` key grants access to temporary queues. However, the lack of the key denies access to temporary queues. As a result if there are multiple `<queue>` entries the last entry will specify the value for access to temporary queues. i.e. In this example it is expected that 'client' can consume from temporary queues and named queue 'exampleQueue2'. Infact what will happen is that the user will only have access to 'exampleQueue2'.

```
<queue>
  <temporary/>
  <users>
    <user>client</user>
  </users>
</queue>
<queue>
  <name>exampleQueue2</name>
  <users>
    <user>client</user>
  </users>
</queue>
```

To work around this issue the correct definition would be:

```

<queue>
    <name>exampleQueue2</name>
    <users>
        <user>client</user>
    </users>
</queue>

<queue>
    <temporary/>
    <users>
        <user>client</user>
    </users>
</queue>

```

Configure Broker and Client Heartbeating

AMQP Heartbeating

Heartbeating at the AMQP protocol level works by sending a small "heartbeat" frame whenever the (half-)connection is idle... That is each peer is responsible for sending a heartbeat frame if it has not sent other data for a given period of time.

Qpid allows for the heartbeat interval to be set on a per client basis, and set to broker configured default if a client does not explicitly set the heartbeat interval.

The successful receipt of a heartbeat message from a broker/client does not imply that it is able to send or receive messages - it is merely indicative that the TCP connection is alive and the broker/client is minimally operational. Higher level application heartbeats are required to test the full functional operability of the client/broker.

To set the heartbeat interval on the client, the system property "amqj.heartbeat.delay" needs to be set to an integer number representing the desired interval in seconds (for example by passing -Damqj.heartbeat.delay=30 on the JVM command-line to get a 30 second interval).

If the heartbeat interval is not explicitly set on the client then the default interval from the broker is taken, this is set in the config.xml file, e.g. in the default config:

```

<heartbeat>
<delay>0</delay>
...
</heartbeat>

```

A <delay> of zero means "no heartbeating".

The client or broker will detect a connectivity problem when it has received no heartbeat, or other data, from its peer for delay * timeoutFactor seconds. The timeoutFactor is set separately for client and broker - the client does not default to using the broker value if it is not set. The timeoutFactor is set on the client by setting the system property "amqj.heartbeat.timeoutFactor" to a valid floating point number. The client and broker default timeoutFactors are (independently) set at 2.0.

If heartbeating is activated, and a client does not receive a heartbeat from the server for delay * (client)timeoutFactor seconds then it closes the TCP connection and activates the failover logic.

If heartbeating is activated, and a broker does not receive a heartbeat from the server for delay * (server)timeoutFactor seconds then it logs this event but does not take any further action (i.e. the connection is not closed, and processing of incoming data/sending outgoing data still occurs.

TCP SO_KEEPALIVE

Neither the client nor the broker (to version 0.5) set SO_KEEPALIVE on the TCP connection, nor is there a way to request them to do so using configuration.

Configure Java Qpid to use a SSL connection.

Using SSL connection with Qpid Java.

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

Setup

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.

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.

Performing the connection.

Configure Log4j CompositeRolling Appender

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

    <layout class="org.apache.log4j.PatternLayout">
        <param name="ConversionPattern" value="%d %-5p [%t] %C{2} (%F:%L) - %m%n"/>
    </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/qpidd.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}qpidd${logsuffix}.log"/>
<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 renames at log
rollover -->
<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"/>
```

Configure Operational Status Logging

How to Configure Operational Status Logging

New in Apache Qpid 0.6 Java Broker is Operational Status Logging. The design overview can be found [here](#) which details all the proposed new logging features.

The Status Logging allows for a range of new log statements which provide details about the various state changes that occur within the broker.

Enabling Status Updates

The new status updates are controlled by the following new configuration entry.

```
<broker>
  ...
  <status-updates>ON</status-updates>
  ...
</broker>
```

If the 'status-updates' entry is missing then Apache Qpid Java broker will default logging on. The value of 'on' is not case sensitive but any other string will disable updates.

Broker Locale

The addition of the new logging format also provided the opportunity to allow localisation of the log messages. Currently we have only completed the mapping US English, which therefore is the default.

As the broker starts up a number of standard messages are logged. These messages will be logged in the VM's default locale, if a mapping is available. Once the broker configuration file is read then any locale specified in the configuration file will be enabled and adjust the future log statements.

```
<broker>
  ...
  <advanced>
    ...
      <locale>en_US</locale>
  </advanced>
  ...
</broker>
```

New Log Messages

There are a number of new log messages generated when status logging is enabled they are broken down into 10 categories. Each of the messages are detailed below in the [Message Section](#).

Log Format

Currently the messages are logged as part of the default log4j configuration. The default broker log4j configuration will produce messages in this format.

```
2009-08-13 12:40:35,192 INFO [qpid.message] MESSAGE [Broker] BRK-1002 : Starting : Listening on
TCP port 5672
```

The message is composed in the following way:

```
<date-time> INFO [qpid.message] MESSAGE <Actor> [<subject>] <MessageID> : <Message>
```

The display of the first three entries '<date-time> INFO [qpid.message]' depend on your particular log4j configuration however you will always get the final message section:

```
MESSAGE <Actor> [<Subject>] <MessageID> : <Message>
```

Actor

There are a number of Actors that can perform logging, each has a different format which gives additional information about the thread that is performing the logging.

Broker

Actor format:

```
[Broker]
```

Used:

On broker startup and shutdown messages logged about the state of the broker will use the Broker actor.

Management

Actor format:

```
[mng:1(169.24.29.116)]
```

Used:

When an operation is performed via the JMX interfaces the connection Actor will provide details of the connection that performed the action.

Queue

Actor format:

```
[vh(/test)/qu(example.queue)]
```

Used:

This is used when the queue is processing the messages on the queue. Currently only SUB-1003 messages will be logged by this actor

Subscription

Actor format:

```
[sub:6(vh(test)/qu(example.queue))]
```

Used:

When a subscription is acting on the broker then it will log messages. Currently SUB-1003 suspend and SUB-1002 messages are the only one that this actor will provide

Channel\Connection

Actor format:

```
[con:1(/127.0.0.1:59556)]
[con:1(guest@/127.0.0.1:59556/test)]
[con:1(guest@/127.0.0.1:59556/test)/ch:1]
```

Used:

There are a number of formats that this actor will present depending on the information available. On initial connection open only the remote ip is available. After authentication the username and vhost are available.

Most logging will be of the latter type were the channel id is also present.

Subject

Binding

Subject format:

```
[vh(/test)/ex(direct/<>/<>)/qu(testQueue)/rk(testQueue)]
```

Channel\Connection

Subject format:

```
[con:1(/127.0.0.1:59556)]
[con:1(guest@/127.0.0.1:59556/test)]
[con:1(guest@/127.0.0.1:59556/test)/ch:1]
```

Exchange

Subject format:

```
[vh(/test)/ex(direct/testName)]
```

MessageStore

Subject format:

```
[vh(localhost)/ms(DerbyMessageStore)]
```

Queue

Subject format:

```
[vh(/test)/qu(testQueue)]
```

Subscription

Subject format:

```
[sub:0(qu(testQueue))]
```

Message List

The definitive list of messages is the property file found [here](#). For readability the list as been reproduced here with additional detail about the various parameterised values, shown like this '<value>', and optional values, shown like '[optional]'.

Broker

```
BRK-1001 : Startup : Version: <Version> Build: <Build>
BRK-1002 : Starting : Listening on <Transport: TCP|TCP/SSL> port <Port>
BRK-1003 : Shuting down : <Transport: TCP|TCP/SSL> port <Port>
BRK-1004 : Ready
BRK-1005 : Stopped
BRK-1006 : Using configuration : <path>
BRK-1007 : Using logging configuration : <path>
```

JMX Management

```
MNG-1001 : Startup
MNG-1002 : Starting : <service> : Listening on port <Port>
MNG-1003 : Shuting down : <service> : port <Port>
MNG-1004 : Ready
MNG-1005 : Stopped
MNG-1006 : Using SSL Keystore : <path>
MNG-1007 : Open : User <username>
MNG-1008 : Close
```

VirtualHost

```
VHT-1001 : Created : <name>
VHT-1002 : Closed
```

DerbyMessageStore/MemoryMessageStore

```
MST-1001 : Created : <full classname>
MST-1002 : Store location : <path>
MST-1003 : Closed
MST-1004 : Recovery Start [: <queue.name>]
MST-1005 : Recovered <count> messages for queue <queue.name>
MST-1006 : Recovery Complete [: <queue.name>]
```

Connections

```
CON-1001 : Open : Client ID <id> : Protocol Version : <version>
CON-1002 : Close
```

AMQChannel

```
CHN-1001 : Create : Prefetch <count>
CHN-1002 : Flow <value: Started|Stopped>
CHN-1003 : Close
CHN-1004 : Prefetch Size (bytes) <bytes> : Count <message count>
```

Queue

```
QUE-1001 : Create : [AutoDelete] [Durable|Transient] [Priority:<levels>] Owner:<name>
QUE-1002 : Deleted
```

Exchange

```
EXH-1001 : Create : [Durable] Type:<value> Name:<value>
EXH-1002 : Deleted
```

Bindings

```
BND-1001 : Create [: Arguments : <key=value>]
BND-1002 : Deleted
```

Subscription

```
SUB-1001 : Create[ : Durable][ : Arguments : {0}]
SUB-1002 : Close
SUB-1003 : State : <state: ACTIVE|SUSPENDED>
```

Configure the Broker via config.xml

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 [Qpid Design - Configuration](#).

Qpid Version

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

M2 - config.xml
M2.1 - config.xml

M2.1 - config.xml

M2.1 Broker config.xml details

Qpid Upgrade steps from M2

Here are the manual changes required to config.xml for M2.1:

1. Remove use of old password format

- Replace line '<class>org.apache.qpid.server.security.auth.database.PlainPasswordVHostFilePrincipalDatabase</class>'
- With '<class>org.apache.qpid.server.security.auth.database.PlainPasswordFilePrincipalDatabase</class>'
- Change format of the password file '\${conf}/passwdVhost' to be username:password
- Rename file on disk '\${conf}/passwdVhost' to '\${conf}/passwd'
- Replace config line '<value>\${conf}/passwdVhost</value>' with '<value>\${conf}/passwd</value>'
- For details on how to configure the new ACLs to restore the per VirtualHost Access rights see [Configure ACLs](#)

2. Update package of AllowAll

- Replace line '<class>org.apache.qpid.server.security.access.AllowAll</class>'
- With '<class>org.apache.qpid.server.security.access.plugins.AllowAll</class>'

3. Remove all Security sections from virtualhosts

Changes from M2 configuration

There are four sections with changes that have occurred since M2. Taking them in order as they appear in the file the first change is in the *connector* section. The *protectio* feature is new its purpose is to limit the underlying send and receive buffers so that they do not grow unbounded. Testing has shown this feature to affect performance so further work is required to fully understand the impact.

The *advanced* section now includes a boolean *enableJMSXUserID* which causes the broker to stamp every message with the UserID of the producing connection. This has an impact on performance so will be improved in a later release with client side setting of JMSXUserID and broker side verification, which is a low overhead.

The *security* section has had a couple of changes. The *PlainPasswordVHostFilePrincipalDatabase* was an early attempt to show how ACLs could be performed. The introduction of a more comprehensive ACL package now removes the need for that class and so the use of *PlainPasswordFilePrincipalDatabase* would be recommended instead. The change to ACLs also included the repackaging of the *AllowAll* ACL class to be in a *pluings* package.

The *virtualhost* sections now have new security sections based on the type of ACL being used. The documentation of which will occur on the a different page.

File Format

This is an overview of the top level of the config file. Description of each section is embedded below. Each section is then described in detail in their own section. Each section that has changes from M2 is highlighted.

```

<broker>
    <connector>
        <!-- Type of connections and properties -->                                <!-- Additional features in M2.1
    -->
    <management>
        <!-- Enablement of management functionality -->
    <advanced>
        <!-- Various advanced flags -->                                         <!-- Additional features in M2.1
    -->
    <security>
        <!-- Definition of available security options -->                      <!-- M2 Incompatible changes in
    M2.1 -->
    <virtualhosts>
        <!-- Definition of available virtual hosts -->                         <!-- M2 Incompatible changes in
    M2.1 -->
    <heartbeat>
        <!-- Heartbeat configuration -->
    <queue>
        <!-- General queue configuration options-->
    <virtualhosts>
        <!-- Configuration of various virtual hosts. -->
</broker>

```

Configuration Sections - Detailed Information

The following sections provide an element by element overview of the config.xml.

Broker

The setting of the prefixes for QPID_HOME and QPID_WORK allows environment variables to be used throughout the config.xml and removes the need for hard coding of paths in this file.

See the [Environment Variables](#) section of the [Getting Started Guide](#) for more information on these variables.

```

<broker>
    <prefix>${QPID_HOME}</prefix>
    <work>${QPID_WORK}</work>
    <conf>${prefix}/etc</conf>

```

Connector

The connector section allows configuration of SSL and related keystore settings. By default this section is commented out and thus SSL is not enabled.

```

<connector>
    <!-- Uncomment out this block and edit the keystorePath and keystorePassword
         to enable SSL support
    <ssl>
        <enabled>true</enabled>
        <sslOnly>true</sslOnly>
        <keystorePath>/path/to/keystore.ks</keystorePath>
        <keystorePassword>keystorepass</keystorePassword>
    </ssl>-->
    <qpiddnio>false</qpiddnio>
    <protectio>
        <enabled>false</enabled>
        <readBufferLimitSize>262144</readBufferLimitSize>
        <writeBufferLimitSize>262144</writeBufferLimitSize>
    </protectio>
    <transport>nio</transport>
    <port>5672</port>
    <sslport>8672</sslport>
    <socketReceiveBuffer>32768</socketReceiveBuffer>
    <socketSendBuffer>32768</socketSendBuffer>
</connector>

```

Management

This element allows the user to switch the connectivity of the management console on/off i.e. if the enabled tag is set to false you will not be able to connect a management console to this broker instance. The JMX Management port is set to 8999 by default but it can be changed here in the XML or on the [command line](#). The management console has the ability to utilise some additional Sun Binary Code License code to improve the security of JMX Connections. Full details of this can be found [here](#).

```
<management>
    <enabled>true</enabled>
    <jmxport>8999</jmxport>
    <security-enabled>false</security-enabled>
</management>
```

Advanced

The elements in this section are used under the covers in the broker. At present, we do not recommend any changes to these settings.

```
<advanced>
    <filterchain enableExecutorPool="true" />
    <enablePooledAllocator>false</enablePooledAllocator>
    <enableDirectBuffers>false</enableDirectBuffers>
    <framesize>65535</framesize>
    <compressBufferOnQueue>false</compressBufferOnQueue>
    <enableJMSXUserID>false</enableJMSXUserID>           <!-- Additional features in M2.1 -->
</advanced>
```

Security

This section lists all the principal databases that are available for authentication and the default access control. The databases understand what SASL mechanisms can be used against their data and so are responsible for registering these SASL mechanisms. Currently we do not provide means of limiting these mechanisms.

```
<security>
    <principal-databases>
        <principal-database>
            <!-- A name for referencing this database-->
            <name>passwordfile</name>
            <!-- The type of principal database -->

            <class>org.apache.qpid.server.security.auth.database.PlainPasswordFilePrincipalDatabase</class>
            <!-- Any attributes associated with the database. Here it is a password file to load.
-->
            <attributes>
                <attribute>
                    <name>passwordFile</name>
                    <value>${conf}/passwd</value>
                </attribute>
            </attributes>
        </principal-database>
    </principal-databases>
    <!-- This access value can be any access manager. The built in defaults are AllowAll and
DenyAll -->
    <access>
        <class>org.apache.qpid.server.security.access.plugin.AllowAll</class>      <!-- NOTE class
change in M2.1 -->
    </access>
    <!-- Properties required when running the JMX Management console. -->
    <jmx>
        <!-- Access file that allows users rights to access the management console. -->
        <access>${conf}/jmxremote.access</access>
        <!-- The principal database to use to authenticate users. -->
        <principal-database>passwordfile</principal-database>
    </jmx>
</security>
```

Virtualhosts

This section allows you to define the set of virtual hosts which will be contained in your broker instance, and the message store & location for each. NB: The commented out section referencing `BDBMessageStore` should be used for all applications wishing to use persistence to disk.

If you are using transient messaging you can use the MemoryMessageStore, with the caveat that scalability for transient use is limited by heap size.

In our example config.xml, we define three virtual hosts which we commonly use for development (development), system testing (test) and integration testing (localhost). In the config.xml the per virtual host sections define both the Message Store in use (MemoryMessageStore for non-persistent applications or BDBMessageStore for persistent application usage) and the security for each virtual host. The security settings are under currently development so subject to changes.

The default virtual host for connections which do not specify a host on the url is 'test' in the example config.xml.

```
<virtualhost>
    <name>localhost</name>
    <localhost>
        <store>
            <!-- <class>org.apache.qpid.server.store.berkeleydb.BDBMessageStore</class>
            <environment-path>${work}/localhost-store</environment-path> -->

            <class>org.apache.qpid.server.store.MemoryMessageStore</class>
        </store>
    </localhost>
</virtualhost>
```

Heartbeat

The Qpid broker sends an internal (only) heartbeat. This element allows configuration of the frequency of this heartbeat. At present, we recommend that you leave this section unchanged !

```
<heartbeat>
    <delay>0</delay>
    <timeoutFactor>2.0</timeoutFactor>
</heartbeat>
```

Queue

This should NOT be changed lightly as it sets the broker up to automatically bind queues to exchanges.

It could theoretically be used to prevent users creating new queues at runtime, assuming that you have created all queues/topics etc at broker startup. However, best advice is to leave unchanged for now.

```
<queue>
    <auto_register>true</auto_register>
</queue>
```

Virtualhosts

This element allows you to specify a location for the virtualhosts.xml file that you wish to use. If you are not using a subdirectory under \$QPID_HOME you can provide a fully qualified path instead. For more information on the content of the virtualhosts.xml file please see [Configure the Virtual Hosts via virtualhosts.xml](#)

```
<virtualhosts>${conf}/virtualhosts.xml</virtualhosts>
```

M2 - config.xml

M2 Broker config.xml details

Changes from M1 configuration

File Format

This is an overview of the top level of the config file. Description of each section is embedded below. Each section is then described in detail in their own section.

```

<broker>
  <!-- Various initial global definitions -->
  <connector>
  <!-- Various connection information about the type connections the broker should listen for-->
  <management>
  <!-- Enablement of management functionality -->
  <advanced>
  <!-- Various advanced flags -->
  <security>
  <!-- Definition of available security options -->
  <virtualhosts>
  <!-- Definition of available virtual hosts -->
  <heartbeat>
  <!-- Heartbeat configuration -->
  <queue>
  <!-- General queue configuration options-->
  <virtualhosts>
  <!-- Configuration of various virtual hosts. -->
</broker>

```

Configuration Sections - Detailed Information

The following sections provide an element by element overview of the config.xml.

Broker

The setting of the prefixes for QPID_HOME and QPID_WORK allows environment variables to be used throughout the config.xml and removes the need for hard coding of paths in this file.

See the [Getting Started Guide](#) for more information on these variables.

```

<broker>
  <prefix>${QPID_HOME}</prefix>
  <work>${QPID_WORK}</work>
  <conf>${prefix}/etc</conf>

```

Connector

The connector section allows configuration of SSL and related keystore settings. By default this section is commented out and thus SSL is not enabled.

```

<connector>
  <!-- Uncomment out this block and edit the keystorePath and keystorePassword
       to enable SSL support
  <ssl>
    <enabled>true</enabled>
    <sslOnly>true</sslOnly>
    <keystorePath>/path/to/keystore.ks</keystorePath>
    <keystorePassword>keystorepass</keystorePassword>
  </ssl>-->
  <qpiddnio>true</qpiddnio>
  <transport>nio</transport>
  <port>5672</port>
  <sslport>8672</sslport>
  <socketReceiveBuffer>32768</socketReceiveBuffer>
  <socketSendBuffer>32768</socketSendBuffer>
</connector>

```

Management

This element allows the user to switch the connectivity of the management console on/off i.e. if the enabled tag is set to false you will not be able to connect a management console to this broker instance.

```

<management>
  <enabled>true</enabled>
</management>

```

Advanced

The elements in this section are used under the covers in the broker. At present, we do not recommend any changes to these settings.

```
<advanced>
<filterchain enableExecutorPool="true"/>
<enablePooledAllocator>false</enablePooledAllocator>
<enableDirectBuffers>false</enableDirectBuffers>
<framesize>65535</framesize>
<compressBufferOnQueue>false</compressBufferOnQueue>
</advanced>
```

Security

This section lists all the principal databases that are available for authentication and the default access control. The databases understand what SASL mechanisms can be used against their data and so are responsible for registering these SASL mechanisms. Currently we do not provide means of limiting these mechanisms.

```
<security>
  <principal-databases>
    <principal-database>
      <!-- A name for referencing this database-->
      <name>passwordfile</name>
      <!-- The type of principal database -->

      <class>org.apache.qpid.server.security.auth.database.PlainPasswordVhostFilePrincipalDatabase</class>
      <!-- Any attributes associated with the database. Here it is a password file to load. -->
      <attributes>
        <attribute>
          <name>passwordFile</name>
          <value>${conf}/passwdVhost</value>
        </attribute>
      </attributes>
    </principal-database>
  </principal-databases>
  <!-- This access value can be any access manager. The built in defaults are AllowAll and
DenyAll -->
  <access>
    <class>org.apache.qpid.server.security.access.AllowAll</class>
  </access>
  <!-- Properties required when running the JMX Management console. -->
  <jmx>
    <!-- Access file that allows users rights to access the management console. -->
    <access>${conf}/jmxremote.access</access>
    <!-- The principal database to use to authenticate users. -->
    <principal-database>passwordfile</principal-database>
  </jmx>
</security>
```

Virtualhosts

This section allows you to define the set of virtual hosts which will be contained in your broker instance, and the message store & location for each. NB: The commented out section referencing BDBMessageStore should be used for all applications wishing to use persistence to disk.

If you are using transient messaging you can use the MemoryMessageStore, with the caveat that scalability for transient use is limited by heap size.

In our example config.xml, we define three virtual hosts which we commonly use for development (development), system testing (test) and integration testing (localhost). In the config.xml the per virtual host sections define both the Message Store in use (MemoryMessageStore for non-persistent applications or BDBMessageStore for persistent application usage) and the security for each virtual host. The security settings are under currently development so subject to changes.

The default virtual host for connections which do not specify a host on the url is 'test' in the example config.xml.

```

<virtualhost>
    <name>localhost</name>
    <localhost>
        <store>
            <!-- <class>org.apache.qpid.server.store.berkeleydb.BDBMessageStore</class>
            <environment-path>${work}/localhost-store</environment-path> -->

            <class>org.apache.qpid.server.store.MemoryMessageStore</class>
        </store>

        <security>
            <!-- Need protocol changes to allow this-->
            <authentication>
                <name>passwordfile</name>
                <!-- Currently this can't be used as Vhost isn't specified at connection
start only connection open -->
                <mechanism>PLAIN</mechanism>
            </authentication>
            <access>

                <class>org.apache.qpid.server.security.access.PrincipalDatabaseAccessManager</class>
                <attributes>
                    <attribute>
                        <name>principalDatabase</name>
                        <value>passwordfile</value>
                    </attribute>
                    <attribute>
                        <name>defaultAccessManager</name>
                        <value>DenyAll</value>
                    </attribute>
                </attributes>
            </access>
        </security>
    </localhost>
</virtualhost>

```

Heartbeat

The Qpid broker sends an internal (only) heartbeat. This element allows configuration of the frequency of this heartbeat. At present, we recommend that you leave this section unchanged !

```

<heartbeat>
    <delay>0</delay>
    <timeoutFactor>2.0</timeoutFactor>
</heartbeat>

```

Queue

This should NOT be changed lightly as it sets the broker up to automatically bind queues to exchanges.

It could theoretically be used to prevent users creating new queues at runtime, assuming that you have created all queues/topics etc at broker startup. However, best advice is to leave unchanged for now.

```

<queue>
    <auto_register>true</auto_register>
</queue>

```

Virtualhosts

This element allows you to specify a location for the virtualhosts.xml file that you wish to use. If you are not using a subdirectory under \$QPID_HOME you can provide a fully qualified path instead. For more information on the content of the virtualhosts.xml file please see [Configure the Virtual Hosts via virtualhosts.xml](#)

```
<virtualhosts>${conf}/virtualhosts.xml</virtualhosts>
```

Configure the Virtual Hosts via virtualhosts.xml

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.

XML Format with Comments

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

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

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

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

            <!-- Define the set of queues to be created at broker startup -->
            <queues>
                <!-- The properties configured here will be applied as defaults to all -->
                <!-- 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>
```

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.

Debug using log4j

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.

Logging Connection State *Deprecated*

deprecation notice Version 0.6 of the Java broker includes [Operational Status Logging](#) functionality which improves upon these messages and as such enabling status logging would be more beneficial.

The configuration file has been left here for assistance 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.

[application-connections.xml](#)

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

[application-debug.xml](#)

Firewall Configuration

Configuration

The access restrictions apply either to the server as a whole or too a particular virtualhost. Rules are evaluated in the virtualhost first, then the server as a whole (most-specific to least-specific). This allows whole netblocks to be restricted from all but one virtualhost. A <firewall> element would appear in either the <broker><security> section or inside the equivalent <virtualhost><security> element.

Elements inside <firewall> would be <rule> or <xml fileName="[path"]/> which can be used to include further rules at that point in the rule chain.

<rule> must have action and either hostname or network attributes. The action attribute must be either allow or deny. Host contains a comma seperated list of [regexp](#)s against which it would match the reverse dns lookup of the connecting IP. Network contains a comma seperated list of CIDR networks against which the IP would be matched.

The first <rule> which matched the connection would apply. If no rules applied, the default-action would apply.

For example, the following could appear in config.xml:

```

<broker>
  <security>
    <firewall default-action="deny">
      <rule access="allow" hostname="*.qpid.apache.org"/>
      <xml fileName="/path/to/file" />
      <rule access="allow" network="192.168.1.0/24" />
      <rule access="allow" network="10.0.0.0/8" />
    </firewall >
  </security>
</broker>

[...]
<virtualhosts>
  <virtualhost>
    <name>prod</name>
    <prod>
      <security>
        <firewall>
          <rule access="deny" network="192.168.1.0/24"/>
        </firewall>
      </security>
    </prod>
  </virtualhost>
</virtualhosts>

```

Any machine in the 192.168.1.0/24 network would be allowed access to any virtualhost other than prod

Any machine in the qpid.apache.org domain would be allowed access to any virtualhost

Any machine in the 10.0.0.0/8 network would be allowed access to any virtual host

Any other machine would be denied access.

Changes would be possible while broker was running via commons-configuration magic when the file is edited. Existing connections would be unaffected by a new rule.

Examples

Denying everybody but foo.bar.com:

```

<firewall default-action="deny">
  <rule access="allow" hostname="foo.bar.com"/>
</firewall>

```

Denying everybody outside of bar.com:

```

<firewall default-action="deny">
  <rule access="allow" hostname=".bar.com"/>
</firewall>

```

Allowing everybody except Baxcorp:

```

<firewall default-action="allow">
  <rule access="deny" hostname=".baxcorp.*"/>
</firewall>

```

Deny everybody except one machine:

```

<firewall default-action="deny">
  <rule access="allow" network="192.168.1.2"/>
</firewall>

```

Allow everybody except one machine:

```
<firewall default-action="allow">
  <rule access="deny" network="192.168.1.2"/>
</firewall>
```

Deny everybody except machines in the range 192.168.1.0-192.168.1.255

```
<firewall default-action="deny">
  <rule access="allow" network="192.168.1.0/24"/>
</firewall>
```

Allow everybody except machines in the range 192.168.1.0-192.168.1.255

```
<firewall default-action="allow">
  <rule access="deny" network="192.168.1.0/24"/>
</firewall>
```

Allow everybody except machines in the range 192.168.0.0-192.168.255.255 unless it's 192.168.1.2, has the magic word in the hostname or is in the IP range 192.168.23.0-192.168.23.255

```
<firewall default-action="allow">
  <rule access="allow" network="192.168.1.2"/>
  <rule access="allow" hostname=".*please.*"/>
  <rule access="allow" network="192.168.23.0/24"/>
  <rule access="deny" network="192.168.0.0/16"/>
</firewall>
```

Complete example configuration files are attached to this page:

| Name | Size | Creator | Creation Date | Comment |
|--|------|---------------|--------------------|---------|
| firewall-test-4-allow-ip-deny-defau... | 3 kB | Aidan Skinner | Apr 22, 2009 07:11 | |
| firewall-test-5-deny-ip-allow-defau... | 3 kB | Aidan Skinner | Apr 22, 2009 07:11 | |
| firewall-test-3-deny-hostname-allow... | 3 kB | Aidan Skinner | Apr 22, 2009 07:11 | |
| firewall-test-2-allow-client-deny-d... | 3 kB | Aidan Skinner | Apr 22, 2009 07:11 | |
| firewall-test-1-no-restrictions.xml | 3 kB | Aidan Skinner | Apr 22, 2009 07:11 | |
| firewall-test-7-deny-cidr-allow-def... | 3 kB | Aidan Skinner | Apr 22, 2009 07:12 | |
| firewall-test-6-allow-cidr-deny-def... | 3 kB | Aidan Skinner | Apr 22, 2009 07:12 | |

How to Tune M3 Java Broker Performance

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 producer (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**.

Successful Tuning Options

The key scenario being targeted 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:

| Strategy | Result |
|--|--|
| Separate connections to one broker for separate streams of messages. | Messages processed successfully, no problems experienced |
| Separate 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.

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.

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 improvement.

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.

How to Use JNDI

How to use the PropertiesFileInitialContextFactory

This ContextFactory uses a java properties formatted file to setup initial values.

JNDI Property setup

By setting the JNDI Initial Context Factory and URL as below it is possible to load any File from the locally mounted file system to use for JNDI purposes. The format of the file is described in the next section.

```
java.naming.factory.initial = org.apache.qpid.jndi.PropertiesFileInitialContextFactory  
java.naming.provider.url = <path to JNDI File>
```

By simply setting these two system properties you can jump straight to the InitialContext creation in your code.

Example properties file

This is the example properties file.

```

# register some connection factories
# connectionfactory.[jndiname] = [ConnectionURL]
connectionfactory.local = amqp://guest:guest@clientid/testpath?brokerlist='vm://:1'

# register some queues in JNDI using the form
# queue.[jndiName] = [physicalName]
queue.MyQueue = example.MyQueue

# register some topics in JNDI using the form
# topic.[jndiName] = [physicalName]
topic.ibmStocks = stocks.nyse.ibm

# Register an AMQP destination in JNDI
#   NOTE: Qpid currently only supports direct,topics and headers
# destination.[jniName] = [BindingURL]
destination.direct = direct://amq.direct//directQueue

```

The property file allows a number of queues to be defined that can then be discovered via JNDI. There are four properties used by the PFICFactory.

`connectionfactory.<jndiname>` this is the Connection URL that the connection factory will use to perform connections.

`queue.<jndiname>` this defines a jms queue or in amqp a amq.direct exchange

`topic.<jndiname>` this defines a jms topic or in amqp a amq.topic exchange

`destination.<jndiname>` this takes a Binding URL and so can be used for defining all amq destinations, queues, topics and header matching.

In all of these properties the `<jndiname>` is the string value that would be given when performing a lookup.

NOTE: This does not create the queue on the broker. You should ensure that you have created the queue before publishing to it. Queues can be declared in the virtualhosts.xml file so that they are created on broker startup, or created dynamically by consuming clients. Topics and other destinations that use temporary queues cannot be created in this way, so a consumer must be created first before publishing messages with mandatory routing.

Example lookup code

The `bindingValue` is the String that would be placed in `<jndiname>` above.

Simple JNDI lookup using files

```

//Ensure you have your system properties set
final String INITIAL_CONTEXT_FACTORY = "org.apache.qpid.jndi.PropertiesFileInitialContextFactory";
System.setProperty(Context.INITIAL_CONTEXT_FACTORY, INITIAL_CONTEXT_FACTORY);
System.setProperty(Context.PROVIDER_URL, _JNDIFile);

// Create the initial context
Context ctx = new InitialContext();

// Perform the binds
object = ctx.lookup(bindingValue);

// Close the context when we're done
ctx.close();

```

Simple JNDI lookup using properties

```
final String INITIAL_CONTEXT_FACTORY = "org.apache.qpid.jndi.PropertiesFileInitialContextFactory";

final String CONNECTION_JNDI_NAME = "local";
final String CONNECTION_NAME = "amqp://guest:guest@clientid/testpath?brokerlist='vm://:1'";

final String QUEUE_JNDI_NAME = "queue";
final String QUEUE_NAME = "example.MyQueue";

// Set the properties ...
Properties properties = new Properties();
properties.put(Context.INITIAL_CONTEXT_FACTORY, INITIAL_CONTEXT_FACTORY);
properties.put("connectionfactory." + CONNECTION_JNDI_NAME, CONNECTION_NAME);
properties.put("queue." + QUEUE_JNDI_NAME, QUEUE_NAME);

// Create the initial context
Context ctx = new InitialContext(properties);

// Perform the lookups
ConnectionFactory factory = (ConnectionFactory)ctx.lookup(CONNECTION_JNDI_NAME);
Queue queue = (Queue)ctx.lookup(QUEUE_JNDI_NAME);

// Close the context when we're done
ctx.close();
```

Using Qpid with other JNDI Providers

Using Qpid with other JNDI Providers

How to use a JNDI Provider

Qpid will work with any JNDI provider capable of storing Java objects. We have a task to add our own initial context factory, but until that's available

First you must select a JNDI provider to use. If you aren't already using an application server (i.e. Tomcat ?) which provides JNDI support you could consider using either:

- Apache's [Directory](#) which provides an LDAP JNDI implementation
- OR the SUN JNDI SPI for the FileSystem which can be downloaded from <http://java.sun.com/products/jndi/downloads/index.html>
- Click : Download JNDI 1.2.1 & More button
- Download: File System Service Provider, 1.2 Beta 3
- and then add the two jars in the lib dir to your class path.

There are two steps to using JNDI objects.

- Bind : Which stores a reference to a JMS Object in the provider.
- Lookup : Which tries to retrieve the reference and create the JMS Object.

There are two objects that would normally be stored in JNDI.

- A ConnectionFactory
- A Destination (Queue or Topic)

Binding

Then you need to setup the values that the JNDI provider will used to bind your references, something like this:

Setup JNDI

```
Hashtable env = new Hashtable(11);
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.fscontext.RefFSContextFactory");
env.put(Context.PROVIDER_URL, LOCAL_FILE_PATH_FOR_STORING_BINDS_PATH_MUST_EXIST);
```

These values are then used to create a context to bind your references.

Perform Binding of ConnectionFactory

```
try
{
    Context ctx = new InitialContext(env);

    // Create the object to be bound in this case a ConnectionFactory
    ConnectionFactory factory = null;

    try
    {
        factory = new AMQConnectionFactory(CONNECTION_URL);
        try
        {
            ctx.bind(binding, factory);
        }
        catch (NamingException e)
        {
            //Handle problems with binding. Such as the binding already exists.
        }
    }
    catch (URLSyntaxException amqe)
    {
        //Handle any exception with creating ConnnectionFactory
    }
}
catch (NamingException e)
{
    //Handle problem creating the Context.
}
```

To bind a queue instead simply create a AMQQueue object and use that in the binding call.

Bind a AMQQueue

```
AMQQueue queue = new AMQQueue(QUEUE_URL);
ctx.bind(binding, queue);
```

Lookup

You can then get a queue connection factory from the JNDI context.

Perform Binding of ConnectionFactory

```
ConnectionFactory factory;
try
{
    factory= (ConnectionFactory)ctx.lookup(binding);
}
catch (NamingException e)
{
    //Handle problems with lookup. Such as binding does not exist.
}
```

Note that you need not cast the bound object back to an AMQConnectionFactory so all your current JMS apps that use JNDI can start using Qpid straight away.

How to create a TopicConnectionFactory and QueueConnectionFactory

AMQConnectionFactory implements TopicConnectionFactory and QueueConnectionFactory as well as the ConnectionFactory.

Interact with a JMX MBean

Interact with a JMX MBean

In order to call a method on an JMX MBean you must know its ObjectName on the server. This can be identified in advance and the exact ObjectName hardcoded/generated in your application, however any evolution of the MBeans ObjectName on the server in future versions would cause such implementation to fail when the ObjectNames no longer exactly match. This will not be identified until an operation is

attempted, as the code does not check that the MBean defined by the ObjectName actually exists before using the ObjectName and would only fail when calling a method and the target is not found by the MbeanServer.

Instead, the suggested route would be to use an ObjectName Pattern along with the queryNames method from the MBeanServerConnection class. You can then define a pattern of key-value pairs that the server will use to find all matching MBean ObjectNames and return these in a Set. By using property name-value pairs that uniquely identify an MBean, it is thus possible to locate the exact ObjectName of the MBean you wish to use without knowing every property name and value in its ObjectName in advance. In doing so you also validate the MBeans existence before using them via an MBean Proxy.

To create an ObjectName pattern for a Qpid JMX MBean, you define the ObjectName with the domain org.apache.qpid, and specify any name-value pairs that identify the MBean(s) you wish to gather the ObjectNames for, followed by ",*" That would define that the server return the ObjectNames of any MBean posessing the specific properties names and values you specify as well as 0 or more other properties of any value.

For example, to identify the VirtualHostManager MBean for a given VirtualHost (VHOST-NAME), you would construct an ObjectName which uniquely identified it by including the MBean type and the containing VirtualHost as below:

```
ObjectName hostManagerObjectName =  
    new ObjectName("org.apache.qpid:type=VirtualHost.VirtualHostManager,VirtualHost=VHOST-NAME,*")
```

This pattern will find the MBean we are interested in, regardless whether it has additional properties or not. Using this pattern, you would query the MBeanServer for all names that match, using the queryNames method from the MBeanServerConnection instance (mbsc) for your JMX connection to the server:

```
Set<ObjectName> objectInstances = mbsc.queryNames(hostManagerObjectName, null);
```

Checking that the returned set is of size 1 will ensure you have matched the specific MBean you wished, otherwise it either did not exist (size 0) or the properties specified were not enough to uniquely identify it and multiple mbeans matched your query (this is unlikely unless actually desired, as the VirtualHost, Exchange, Queue etc properties makes it straightforward to uniquely identify Qpid JMX objects)

The ObjectName(s) returned can then be retrieved from the Set and used as the target ObjectName for invoking operations via the MBeanServerConnection. One way of doing so is to use the ObjectName and the Interface for the MBean in question to create an MBean Proxy object, which can be used as if it were a normal local java object:

```
Java 6+:  
  
ManagedBroker hostManagerProxy = JMX.newMBeanProxy(mbsc, objectName, ManagedBroker.class);  
  
Java5+:  
  
ManagedBroker hostManagerProxy = (ManagedBroker)  
    MBeanServerInvocationHandler.newProxyInstance(mbsc, objectName, ManagedBroker.class, false);
```

You may then call methods on the proxy object as if it were a local object:

```
hostManagerProxy.createNewQueue(queueName, null, true);
```

Qpid Java Build How To

Build Instructions - General

Check out the source

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

<https://svn.apache.org/repos/asf/qpid/trunk>

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.

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.

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:

| 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:

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

Build targets

The main build targets you are probably interested in are:

| 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

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

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

Dependencies

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

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

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

broker

- common
- management/common

client

- Common

systest

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

perftests

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

management/eclipse-plugin

- broker
- common
- management/common

management/console

- common
- client

management/agent

- common
- client

management/tools/qpid-cli

- common
- management/common

management/client

- common
- client

integrationtests

- systests
- client
- common
- junit-toolkit

testkit

- client
- broker
- common

tools

- client
- common

client/examples

- common
- client

broker-plugins

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

What next ?

If you want to run your built Qpid package, see our [Getting Started Guide](#) 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.

Building

Java Building Pages

Developer Information

- Build How To
- CruiseControl
- Qpid Developer Documentation
- Coding Standards
- AMQP Version Handling
- URL format for Connections and Binding
- Creating Java unit tests with InVM broker

Performance Testing

- IBM JMS Performance Test Results

Management Interfaces

- Management Features
- Management Console

C++ Building Pages

Developer Information

- Build How To get it going
- The Python test suite
- C++ Coding Standard, Design notes etc

Performance Testing

- Performance Testing for C++

Management Interfaces

- Running & Administration and getting started for the C++ Broker
- Managing the C++ Broker

Python and Ruby obviously don't need building.

.NET Building Pages

.NET can be built under MONO, or on a Windows .NET platform

CruiseControl

Prerequisites

Check out the source

see <http://cwiki.apache.org/qpid/building.html>

Install CruiseControl

Download CruiseControl from: <http://cruisecontrol.sourceforge.net/>

- Unzip the release to a directory, for example ~/cruisecontrol-bin-2.7.2
- Check that the scripts cruisecontrol-bin-2.7.2/cruisecontrol.sh and cruisecontrol-bin-2.7.2/apache-ant-1.7.0/bin/ant have execution permission.
- Make sure your directory ~/.ant/lib contains the following jars:
 - The ant jar files that can be found in cruisecontrol-bin-2.7.2/apache-ant-1.7.0/lib/
 - xalan-2.7.0.jar

Set system variables

Prior to use CruiseControl you'll need to set two system variables:

| Variable | Value |
|---------------|---|
| CC_HOME | path to your qpid project, for example /home/foo/projects/qpid |
| CPPSTORE_HOME | path to your C++ store, for example /home/foo/projects/bdbstore-cpp |

Note: the cpp store can be checked out from: <https://svn.jboss.org/repos/rhmessaging/store/trunk/cpp>

Notes

- Only unix scripts are currently provided

Running CruiseControl

Run cruisecontrol-bin-2.7.2/cruisecontrol.sh from CC_HOME/cc

Projects

| Project | Description |
|--------------------------|---|
| qpid-cpp-trunk | Builds and tests the C++ broker |
| qpid-cpp-trunk-perftests | Runs the C++ performance tests |
| qpid-java-trunk | Builds and runs the Java tests with an 0.8 inVM broker, a c++ broker without prefetch and a c++ broker with pre-fetch |
| bdbstore-cpp-trunk | Builds the C++ store (required for the Java tests) |
| example-automation | Runs all the example combinations for python, C++ and java |

Performance Testing for C++

How to measure performance of my hardware

Overview

Brief page on how to get perf data for your configuration. Note that per data is affected greatly by hardware, and OS tuning. So tune your OS and baseline your hardware. You should be able to get perfest 'below' to within 5%-8% of the max for the baseline of a well setup config. If you can't - welcome to mail the list.

Basic tuning

Tuning will increase your throughput and increase your determinism. The simple things are turn off cpuspeed, irqbalance etc and set timer resolution for the processors you use.... All the standard stuff.

Sample Data

Some sample results running on the out of date lump in the corner, running on current hardware you should easily beat these.

1K block Size

| clients | pubs/sec | subs/sec | transfers/sec | Mbytes/sec |
|---------|----------|----------|---------------|------------|
| | | | | |

..

more data. to illustrate point TODO

...

Note that there will be two limits, one is the size of the pipe, the other will be how many IO's per-second the hardware can do. On small messages 64bytes or less you will hit the IO limit, on larger messages you will hit the network bandwidth, then scale with more NICs. You can get more 'batch' if you like to get the number up when you hit the IO's per second limit with AMQP-0-10 sync points etc... play with perfest + baseline tools like netperf.

The following tool is including, or can be located in 'cpp/src/tests/' Run with --help to check the options on the latest version.

```
$ ./perftest --help
N4qpid7Options9ExceptionE: Test Options:
-h [ --host ] HOST (localhost)      Broker host to connect to
-b [ --broker ] HOST (localhost)    Broker host to connect to
-p [ --port ] PORT (5672)          Broker port to connect to
-v [ --virtualhost ] VHOST         virtual host
-n [ --clientname ] ID (cpp)       unique client identifier
--username USER (guest)           user name for broker log in.
--password USER (guest)           password for broker log in.
--help                            print this usage statement
--setup                           Create shared queues.
--control                          Run test, print report.
```

```

--publish          Publish messages.
--subscribe       Subscribe for messages.
--mode shared|fanout|topic (shared)
                  Test mode.
                  shared: --qt queues, --npubs publishers
                  and --nsubs subscribers per queue.

                  fanout: --npubs publishers, --nsubs subs
                  criers, fanout exchange.
                  topic: --qt topics, --npubs publishers a
                  nd --nsubs subscribers per topic.

--npubs N (1)      Create N publishers.
--count N (500000)  Each publisher sends N messages.
--size BYTES (1024) Size of messages in bytes.
--pub-confirm yes|no (1) Publisher use confirm-mode.
--durable yes|no (0) Publish messages as durable.
--unique-data yes|no (0) Make data for each message unique.
--nsubs N (1)       Create N subscribers.
--sub-ack N (0)     N>0: Subscriber acks batches of N.
                    N==0: Subscriber uses unconfirmed mode
                    Create N queues or topics.
--qt N (1)          Desired number of iterations of the test
--iterations N (1)

-s [ --summary ]   Summary output: pubs/sec subs/sec transf
                  ers/sec Mbytes/sec
--queue_max_count N (0) queue policy: count to trigger 'flow to
                         disk'
--queue_max_size N (0) queue policy: accumulated size to trigge
                         r 'flow to disk'
--interval_sub ms (0) >=0 delay between msg consume
--interval_pub ms (0) >=0 delay between msg publish

Logging options:
--log-output FILE (stderr) Send log output to FILE. FILE can be a file name
                           or one of the special values:
                           stderr, stdout, syslog
-t [ --trace ]           Enables all logging
--log-enable RULE (error+) Enables logging for selected levels and component
                           s. RULE is in the form 'LEVEL[+][:PATTERN]'
                           Levels are one of:
                           trace debug info notice warning error critical
                           For example:
                           '--log-enable warning+' logs all warning, error
                           and critical messages.
                           '--log-enable debug:framing' logs debug messages
                           from the framing namespace. This option can be
                           used multiple times
--log-time yes|no (1)    Include time in log messages
--log-level yes|no (1)   Include severity level in log messages
--log-source yes|no (0)  Include source file:line in log messages
--log-thread yes|no (0)  Include thread ID in log messages
--log-function yes|no (0) Include function signature in log messages

```

There are two ways to use perftest: single process or multi-process.

If none of the --setup, --publish, --subscribe or --control options are given perftest will run a single-process test.

For a multi-process test first run:

```
perftest --setup <other options>
```

and wait **for** it to complete. The remaining process should run concurrently::

Run --npubs times: perftest --publish <other options>

Run --nsubs times: perftest --subscribe <other options>

```
Run once:           perf test --control <other options>
Note the <other options> must be identical for all processes.
```

Qpid Cpp Build How To

Note: for building on windows, see [QpidCppWindowsBuild](#)

Qpid SVN Trunk Build Instructions

Prerequisites

Some of the source is auto-generated from the AMQP spec file. This generator is written in Ruby. Auto-generation is performed by running make (see below); no special steps are required.

Ensure you have the latest devel versions of the following packages installed:

- boost <<http://www.boost.org>> (>=1.33) *
- e2fsprogs <<http://e2fsprogs.sourceforge.net/>>

* There is a patch to get v.1.32 working in the svn tree though that is only recommended as a last resort.

To build directly from the SVN repository you will need all of the above plus the following development tools:

- pkgconfig <<http://pkgconfig.freedesktop.org/wiki/>>
- gcc <<http://gcc.gnu.org/>>
- GNU make <<http://www.gnu.org/software/make/>>
- autoconf <<http://www.gnu.org/software/autoconf/>>
- automake <<http://www.gnu.org/software/automake/>>
- help2man <<http://www.gnu.org/software/help2man/>>
- libtool <<http://www.gnu.org/software/libtool/>>
- doxygen <<ftp://ftp.stack.nl/pub/users/dimitri/>>
- graphviz <<http://www.graphviz.org/>>
- ruby <<http://www.ruby-lang.org>>

★ Hint: To check and install all of the above, use (as root):

```
yum install boost-devel e2fsprogs-devel pkgconfig gcc-c++ make autoconf automake help2man libtool
doxygen graphviz ruby
```

Optional cluster functionality requires:

- openais <<http://openais.org/>>

Optional XML exchange requires:

- xqilla <<http://xqilla.sourceforge.net/HomePage>>
- xerces-c <<http://xerces.apache.org/xerces-c/>>

Optional SSL support requires:

- nss <<http://www.mozilla.org/projects/security/pki/nss/>>
- nspr <<http://www.mozilla.org/projects/nspr/>>

Check out the source

Check the source for Qpid java out of our subversion repository: <https://svn.apache.org/repos/asf/qpid/trunk/qpid>

```
svn co https://svn.apache.org/repos/asf/qpid/trunk/qpid
```

Automake vs CMake

Currently we have 2 parallel build systems one using automake the other using cmake.

We are moving towards cmake and will eventually get rid of automake, but for the moment the cmake system is not complete and we have to live with both.

That means if you add a file to the build, you must update **both** Makefile.am and CMakeLists.txt and check that you can build with both systems as described below.

Build with automake

First you need to initialize the autotools:

```
./bootstrap
```

To build Qpid, run

```
./configure  
make
```

By default, configure will enable the options for which it finds the installed packages. However, to override this behavior, use parameters with configure to disable unwanted options.

 Hint: To see all the configure options, run

```
./configure --help
```

Finally, to make sure all the test pass both C++ and Python run

```
make check
```

As a convenient shortcut you can do all of the above steps in one command with:

```
./bootstrap -build
```

Build with cmake

```
mkdir builddir  
cd builddir  
cmake <path to checkout>/cpp/CMakeLists.txt  
make # Build  
make test # Run tests
```

You can also use the interactive ccmake on linux and the GUI on windows to modify the configuration.

QpidCppWindowsBuild

These notes were contributed by Vince Seavello and include details of getting a build from the svn repository to work on:

Windows:

Windows Vista (64 bit)
Windows Server 2008 (64 bit).
Windows XP (32 bit)

Visual Studio:

Visual Studio 2008 Express (32 bit C++ package)
Visual Studio Team Suite 2008 (32 bit and 64 bit C++ packages)
Visual Studio 2008 Professional (32 bit and 64 bit C++ packages)

Note: Step 2.5.1 (nmake /f protocol_gen.mak) is not needed for the M4-release build. It's only needed when checking sources out of svn.

1) Introduction

The following are notes for building and testing QPID M4 on Windows.

This document is split into 4 sections:

- setup for building QPID M4 on Windows. This includes obtaining sources, supporting technologies, installation and configuration of the build environment.
- building QPID M4 on Windows.
- running the Linux test suites against the Windows QPID broker.
- building the tests on a Windows system.

These notes are intended to highlight issues discovered during the build and testing of QPID on Windows. It's a work in progress.

Comments welcome.

2) Setup of QPID M4 on Windows

2.1) Obtaining the sources

M4 C++ broker and client source archive is available on the QPID web site.

<http://www.apache.org/dist/qpid/M4/qpid-cpp-M4.tar.gz>

The sources contained here will build a Static_Debug version of the broker and client libraries. To be able to build a Static_Release version, replace the file cpp/src/qpid/InlineAllocator.h with the M5 version of the file:

<https://svn.apache.org/repos/asf/qpid/trunk/qpid/cpp/src/qpid/InlineAllocator.h>

2.2) Setting up for the build

Builds have been taking place using a variety of platforms and versions of Visual Studio. Platforms include:

Windows:

Windows Vista (64 bit)
Windows Server 2008 (64 bit).
Windows XP (32 bit)

Visual Studio:

Visual Studio 2008 Express (32 bit C++ package)
Visual Studio Team Suite 2008 (32 bit and 64 bit C++ packages)
Visual Studio 2008 Professional (32 bit and 64 bit C++ packages)

2.3) Technologies Dependencies

2.3.1) boost

The stated requirement is boost 1.35.0. 1.36.0 is also being tested in our labs. There are only a few version compatibility issues detected.

You can find install images for boost at:

http://www.boostpro.com/boost_1_35_0_setup.exe http://www.boostpro.com/boost_1_36_0_setup.exe

Once you've installed one of these, set your environment variables:

for 1.35.0:
BOOST_ROOT = C:\Program Files (x86)\boost\boost_1_35_0
BOOST_VERSION = 103500

for 1.36.0:
BOOST_ROOT = C:\Program Files (x86)\boost\boost_1_36_0
BOOST_VERSION = 103600

Note: "Program Files (x86)" is "Program Files" on the 32 bit system. The boost libs are 32 bit. When building on a 64 bit system the 32 bit boost libs are installed in "Program Files (x86)". 64 bit boost build is being looked into.

2.3.2) python

Use Python 2.6, which is the stated requirement for the QPID build.
Pick up a copy at:

<http://www.python.org/ftp/python/2.6/python-2.6.msi>

You must include the python directory in the environment variable PATH.
Typically, python is installed in:

C:\python26

test to see that python is installed correctly and that you have the correct version:

C:\>python --version
Python 2.6.1

2.3.3) ruby

Use Ruby 1.8.6, which is the required version. Pick up a copy at:

http://rubyforge.org/frs/?group_id=167

You must include the ruby directory in the environment variable PATH.
Typically, ruby is installed in:

C:\ruby\bin

test to see that ruby is installed correctly and that you have the correct version:

```
C:>ruby -v  
ruby 1.8.6 (2007-09-24 patchlevel 111) [i386-mswin32]
```

2.4) Build tools

The QPID build environment on Windows are centered around Visual Studio C++ Development tools. There is an effort to reduce the overhead of the QPID build so developers who are unfamiliar with the Visual Studio IDE can build and test on Windows with as minimal an effort as possible.

2.4.1) Visual Studio

Microsoft makes Visual Studio Express available at no charge. You can find it online at the following location:

<http://www.microsoft.com/downloads/details.aspx?FamilyID=3254c868-bcb9-412c-95c6-d100c872ec60&DisplayLang=en>

2.5) M4 preparation

2.5.1) Initial environment configuration

Most of the prep is in the installation of the helper apps, Visual Studio, source extraction and patching. When that's all done, there is one more step you still need to do before you can build anything.

Using the Visual Studio's Command Prompt, change directory to qpid/cpp/src and run:

```
nmake protocol_gen.mak
```

2.5.2) source changes

The described source fix needed to be able to get a clean Static_Release build from M4 is listed above. This fix wasn't available before M4 was finalized, so it was added in M5.

Other fixes may be available in M5 to address other issues, but haven't been tried in this environment.

Any changes made to source files during this investigation haven't been submitted. They are presented here for what they're worth. Diffs of source changes are appended at the end of this document.

2.5.3) Debug vs Release

When building QPID, you may want to change from building static debug to static release. There are several ways to do this. Look in the tool bar at the top of the Visual Studio window. Locate the pulldown selection box that contains the value "Debug" or "Release". Change the value to whichever active solution configuration you would like to produce. All built objects/executables will be found in the subdirectories Static_Debug or Static_Release, depending on your active solution configuration.

3) Building QPID M4 on Windows

3.1) QPID Project Files

Visual Studio uses special configuration files to identify the "solution" and "project" parameters. These are made available in .sln and .vcproj files. There are a smattering of these files in the M4 source tree, and more are being added to the M5 tree as time goes on.

For QPID, look for the following:

```
qpid/cpp/src/qpid.sln  
qpid/cpp/src/broker.vcproj  
qpid/cpp/src/client.vcproj  
qpid/cpp/src/common.vcproj  
qpid/cpp/src/MaxMethodBodySize.vcproj
```

When you open the solution, you can choose to build the entire solution, or just a project within. The executables and objects will be in Static_Release

or Static_Debug, depending on your build settings.

Select the solution in the solution explorer pane. Right mouse click and choose "Build Solution". When complete, you should have built the libraries and qpidbroker.exe.

3.2) Libraries

The Windows QPID M4 build will produce 2 libraries:

qpidclients.lib
qpidcommons.lib

The Linux build will also produce a broker library. The Windows build links the broker source objects directly into the broker, but doesn't add them to a library. A broker library will be needed for the building of some tests.

The build of the broker will have compiled all the source objects which are to live in the broker library. By creating a library by hand we will be able to link the tests that require the broker library. See libs.mk later in this document (section 5.4.1).

Libraries are placed in cpp/src.

Our efforts don't include building the qmfconsole library yet, so no status on that.

4) Testing AMQP from a Linux system

There isn't a "push button" build and test for Windows. In fact, in M4 there are relatively few tests that build for Windows. However, the tests do run on Linux. The Windows broker can be tested by running the test suites from a Linux platform.

4.1) localhost vs distributed

The test suites as they are built and run on Linux assume brokers and tests are running on the localhost. There are a few that try to use ssh to start and stop processes on a remote host. But, the majority of the tests start brokers locally, look for PID files and run the tests against the local brokers.

In order to use these tests against a remote Windows broker, some modifications will be necessary. These changes are outlined later in this document.

4.2) test harnesses

QPID makes use of the boost test framework for some of it's tests, but there are a number of different tools and programming methods used. These are:

make - Building and running the tests from the makefile.
Provides a convenient way to produce a push button QPID.
The makefiles allow selection of subsets of tests and provide some command parameters to the test framework.

boost - Boost provides libraries and header files for putting together test cases, test suites and overall test execution control.

/bin/sh shell - the "bridge" between make and the test cases is handled through shell scripts and other tools, like valgrind and libtool. There are makefile variables that can control how some of these tools are utilized.

python - There is a separate python directory with test cases and test suites. These tests are initiated from the makefile by calling a wrapper boost test case called python_tests.

C++ - The .cpp source files in the tests directory are the main consumers of the boost framework. They typically provide a common command line interface that every test shares, as well as additional options specific to the test itself. Many of these tests have multiple test cases within and are controlled, to some fashion, by the command line options.

At the top level is the make file. To initiate the tests, type the

following:

```
make check
```

This will build all the QPID sources and tests. After all has been built, the test suites are run. There are several options you can use when using make to narrow down the running of the tests.

You can select a specific list of test suites by setting the variable TESTS:

```
make check TESTS=perftest
```

These command line options have been helpful in isolating the various test suites.

The makefile uses the script run_test to initiate the indicated test. In most cases, the test being requested is front-ended by a shell script generated during the make process.

The run_test script can pass command line options through to the called program, but the makefile isn't set up to send much in the way of options. Many of the lower level test programs have an option that accepts a broker address on the command line. In some cases, the intermediate scripts have a broker option, too.

In order to use the Linux tests against the Windows broker, the tests have to be redirected to the remote broker address. Rather than modify the structure of the Makefile.am file used to generate the final makefile, the broker address can be passed to the tests in the execution environment. The modified command line invocation then looks like this:

```
BROKER="IP_Address" make check TESTS="fanout_perftest"
```

The run_test script can be modified to look for a broker address environment variable. Because run_test is used to start many of the test programs, sometimes the broker address should not to be passed through. In particular, the broker address is not necessary when starting and stopping localhost brokers. The run_test script determines when to pass the broker address by looking at the program name.

Alternatively, tests can be run directly. The fanout_perftest example can be run like this:

```
./perftest --summary --count 30000 --broker IP_Address \  
--mode fanout --npubs 16 --nsubs 16 --size 64
```

The python tests can be run by changing directory to the python test dir and using one of the following command lines:

```
./run-tests --skip-self-test -v -s 0-10-errata \  
-l cpp_failing_0-10.txt \  
-b 10.197.62.244:5672  
./run-tests -v -s 0-10-errata -l cpp_failing_0-10.txt \  
-b 10.197.62.244:5672  
./run-tests -v -s 0-10-errata -b 10.197.62.244:5672
```

4.3) Changes to Linux tests

Some of the tests, or scripts that initiate the tests, have the localhost address hardcoded, or look for a broker PID in a local file. A majority of the changes made to allow the tests to be run from a Linux system to a Wondows broker deal with this limitation.

The tests that were run successfully against the Windows broker from a Linux system are:

```
client_test  
quick_perftest  
quick_topictest  
python_tests  
run_federation_tests  
fanout_perftest  
shared_perftest  
multiq_perftest  
topic_perftest  
txtest
```

```
latencytest  
echotest  
benchmark
```

5) Building tests on Windows

The Linux test environment makes use of tools, scripts and methods that don't work on a Windows platform. These issues will have to be addressed before a "push button" test can be accomplished similar to that on the Linux system.

In the mean time, the test programs can be built on a Windows platform and run by hand. The following sections talk about this.

5.1) quick and dirty modifications

The modifications described here do not represent "best practice" coding methodologies. They are presented for information sake.

The following files required some sort of modification to build on Windows. A diff listing will be added to the end of this document.

```
cpp/src/tests/ClientSessionTest.cpp  
cpp/src/tests/FieldTable.cpp  
cpp/src/tests/MessageBuilderTest.cpp  
cpp/src/tests/MessageTest.cpp  
cpp/src/tests/QueueOptionsTest.cpp  
cpp/src/tests/TimerTest.cpp  
cpp/src/tests/Uuid.cpp  
cpp/src/tests/logging.cpp  
cpp/src/tests/perftest.cpp  
cpp/src/tests/topic_listener.cpp  
cpp/src/tests/topic_publisher.cpp  
cpp/src/tests/unit_test.h
```

While the following needed modification, they have been excluded from the build due to larger porting issues.

```
cpp/src/tests/ForkedBroker.h  
cpp/src/tests/failover_soak.cpp  
cpp/src/tests/ConsoleTest.cpp
```

5.2) nmake environment

A small handful of make files have been created to use on a Windows platform to build the various test programs. The nmake utility is used with these files to perform the compilation.

```
tests/libs.mk  
tests/tests.mk
```

5.3) diff listings

```
=====
```

```
cpp/src/tests/ClientSessionTest.cpp  
38a39,44  
> #if defined(_WIN32)  
> # include <windows.h>  
> # include <winbase.h>  
> #endif  
>  
>  
225a232,234  
> #if defined(_WIN32)  
> Sleep(1000);  
> #else  
226a236  
> #endif  
277a288,290  
> #if defined(_WIN32)  
> Sleep(2000);  
> #else  
278a292  
> #endif  
291a306,308  
> #if defined(_WIN32)  
> ::Sleep(300* 1);  
> #else  
292a310  
> #endif
```

```
=====
cpp/src/tests/FieldTable.cpp
25c25,27
< #include <alloca.h>
—
> #if !defined (_WIN32)
> # include <alloca.h>
> #endif
=====

cpp/src/tests/MessageBuilderTest.cpp
39c39
< uint64_t id;
—
> boost::uint64_t id;
215c215
< BOOST_CHECK_EQUAL((uint64_t) 1, builder.getMessage()->getPersistenceId());
—
> BOOST_CHECK_EQUAL((boost::uint64_t) 1, builder.getMessage()->getPersistenceId());
=====

cpp/src/tests/MessageTest.cpp
31c31,33
< #include <alloca.h>
—
> #if !defined (_WIN32)
> # include <alloca.h>
> #endif
81,82c83,84
< BOOST_CHECK_EQUAL((uint64_t) data1.size() + data2.size(), msg->contentSize());
< BOOST_CHECK_EQUAL((uint64_t) data1.size() + data2.size(), msg->getProperties<MessageProperties>()->getContentLength());
—
> BOOST_CHECK_EQUAL((boost::uint64_t) data1.size() + data2.size(), msg->contentSize());
> BOOST_CHECK_EQUAL((boost::uint64_t) data1.size() + data2.size(), msg->getProperties<MessageProperties>()->getContentLength());
85c87
< BOOST_CHECK_EQUAL((uint8_t) PERSISTENT, msg->getProperties<DeliveryProperties>()->getDeliveryMode());
—
> BOOST_CHECK_EQUAL((boost::uint8_t) PERSISTENT, msg->getProperties<DeliveryProperties>()->getDeliveryMode());
=====

cpp/src/tests/QueueOptionsTest.cpp
24c24,26
< #include <alloca.h>
—
> #if !defined(_WIN32)
> # include <alloca.h>
> #endif
=====

cpp/src/tests/TimerTest.cpp
78c78
< uint64_t difference = abs(expected - actual);
—
> uint64_t difference = abs((long)(expected - actual));
=====

cpp/src/tests/Uuid.cpp
25c25,27
< #include <alloca.h>
—
> #if !defined(_WIN32)
> # include <alloca.h>
> #endif
=====

cpp/src/tests/logging.cpp
25a26
> # include "qpid/log/windows/SinkOptions.h"
272a274,276
> #if defined(_WIN32)
> qpid::log::windows::SinkOptions sinks("test");
> #else
273a278
```

```

> #endif
288a294,296
> #if defined(_WIN32)
> qpid::log::windows::SinkOptions sinks("test");
> #else
289a298
> #endif
347a357,360
> #if defined(_WIN32)
> qpid::log::windows::SinkOptions *sinks =
> dynamic_cast<qpid::log::windows::SinkOptions *>(opts.sinkOptions.get());
> #else
349a363
> #endif

=====

```

cpp/src/tests/perftest.cpp

41c41,43

< #include <unistd.h>

```

> #if !defined(_WIN32)
> # include <unistd.h>
> #endif

=====
```

cpp/src/tests/topic_listener.cpp

43a44,46

```

> #if defined(_WIN32)
> # include <process.h>
> #endif

=====
```

cpp/src/tests/topic_publisher.cpp

43c43,45

< #include <unistd.h>

```

> #if !defined(_WIN32)
> # include <unistd.h>
> #endif

=====
```

cpp/src/tests/unit_test.h

56c56

< #if (BOOST_VERSION < 103600)

```

> #if (BOOST_VERSION <= 103600)

5.4) make files
```

Once the Visual Studio build has been run, the objects for the broker are available and can be combined into a broker library. This nmake file can be used in the tests directory to create the library. This will be necessary to build the unit_test test suite.

From the tests directory you can run nmake with the tests.mk file listed below to build the tests. This build file does not build the files:

ConsoleTest.cpp
failover_soak.cpp

5.4.1) lib.mk

```
C89_COMPILER=C:/Program Files (x86)/Microsoft Visual Studio 9.0/VC/bin/cl.exe
C89_LINKER=C:/Program Files (x86)/Microsoft Visual Studio 9.0/VC/bin/link.exe
LINK="link.exe"
```

OBJEXT=.obj

```
INCLUDES=/I "C:/Program Files (x86)/boost/boost_1_36_0/include/103600" /I "C:/Program Files (x86)/boost/boost_1_36_0/." /I ".." /I "../gen"
FLAGS=/D "NDEBUG" /D "WIN32" /D "_CONSOLE" /D "_CRT_NONSTDC_NO_WARNINGS" /D "NOMINMAX" /D "WIN32_LEAN_AND_MEAN" /D "_SCL_SECURE_NO_WARNINGS" /D "_CRT_SECURE_NO_WARNINGS" /FD /EHsc /MT /W3 /c /TP /wd4244 /wd4800 /wd4290 /wd4355
```

```
LIBS=ws2_32.lib secur32.lib rpcrt4.lib kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib
uid.lib odbc32.lib odbc32.lib
```

```
LDFLAGS=/INCREMENTAL:NO /LIBPATH:." /LIBPATH:"C:\Program Files (x86)\boost\boost_1_36_0\lib" /DEBUG  
/SUBSYSTEM:CONSOLE /OPT:REF /OPT:ICF /DYNAMICBASE /NXCOMPAT /MACHINE:X86 /nologo /errorReport:prompt  
  
RELDIR=Static_Release  
ROOTDIR=C:\Users\v-vinseal\Dev\AMQP\qpid-M4\cpp\src\tests  
  
check_PROGRAMS=  
check_LTILIBRARIES=  
TESTS=  
EXTRA_DIST=  
CLEANFILES=  
  
.cpp$(OBJEXT):  
echo $(CC) /O2 $(INCLUDES) $(FLAGS) /Fo..\Static_Release\qmfcosole\l386/" /Fd..\Static_Release\qmfcosole\l386\vc90.pdb" $<  
  
broker_OBJECTS=../Static_Release/broker/l386/Acl.obj \  
../Static_Release/broker/l386/Agent.obj \  
../Static_Release/broker/l386/Binding.obj \  
../Static_Release/broker/l386/Bridge.obj \  
../Static_Release/broker/l386/Bridge2.obj \  
../Static_Release/broker/l386/Broker.obj \  
../Static_Release/broker/l386/Broker2.obj \  
../Static_Release/broker/l386/BrokerDefaults.obj \  
../Static_Release/broker/l386/BrokerSingleton.obj \  
../Static_Release/broker/l386/Connection.obj \  
../Static_Release/broker/l386/Connection2.obj \  
../Static_Release/broker/l386/Connection3.obj \  
../Static_Release/broker/l386/ConnectionFactory.obj \  
../Static_Release/broker/l386/ConnectionHandler.obj \  
../Static_Release/broker/l386/DeliverableMessage.obj \  
../Static_Release/broker/l386/DeliveryRecord.obj \  
../Static_Release/broker/l386/DirectExchange.obj \  
../Static_Release/broker/l386/DtxAck.obj \  
../Static_Release/broker/l386/DtxBuffer.obj \  
../Static_Release/broker/l386/DtxManager.obj \  
../Static_Release/broker/l386/DtxTimeout.obj \  
../Static_Release/broker/l386/DtxWorkRecord.obj \  
../Static_Release/broker/l386/EventAllow.obj \  
../Static_Release/broker/l386/EventBind.obj \  
../Static_Release/broker/l386/EventBrokerLinkDown.obj \  
../Static_Release/broker/l386/EventBrokerLinkUp.obj \  
../Static_Release/broker/l386/EventClientConnect.obj \  
../Static_Release/broker/l386/EventClientConnectFail.obj \  
../Static_Release/broker/l386/EventClientDisconnect.obj \  
../Static_Release/broker/l386/EventDeny.obj \  
../Static_Release/broker/l386/EventExchangeDeclare.obj \  
../Static_Release/broker/l386/EventExchangeDelete.obj \  
../Static_Release/broker/l386/EventFileLoadFailed.obj \  
../Static_Release/broker/l386/EventFileLoaded.obj \  
../Static_Release/broker/l386/EventQueueDeclare.obj \  
../Static_Release/broker/l386/EventQueueDelete.obj \  
../Static_Release/broker/l386/EventSubscribe.obj \  
../Static_Release/broker/l386/EventUnbind.obj \  
../Static_Release/broker/l386/EventUnsubscribe.obj \  
../Static_Release/broker/l386/Exchange.obj \  
../Static_Release/broker/l386/Exchange2.obj \  
../Static_Release/broker/l386/ExchangeRegistry.obj \  
../Static_Release/broker/l386/FanOutExchange.obj \  
../Static_Release/broker/l386/HeadersExchange.obj \  
../Static_Release/broker/l386/IncompleteMessageList.obj \  
../Static_Release/broker/l386/Link.obj \  
../Static_Release/broker/l386/Link2.obj \  
../Static_Release/broker/l386/LinkRegistry.obj \  
../Static_Release/broker/l386/ManagementBroker.obj \  
../Static_Release/broker/l386/ManagementExchange.obj \  
../Static_Release/broker/l386/Message.obj \  
../Static_Release/broker/l386/MessageAdapter.obj \  
../Static_Release/broker/l386/MessageBuilder.obj \  
../Static_Release/broker/l386/MessageStoreModule.obj \  
../Static_Release/broker/l386/NameGenerator.obj \  
../Static_Release/broker/l386/NullMessageStore.obj \  
../Static_Release/broker/l386/Package.obj \  
../Static_Release/broker/l386/Package2.obj \  
../Static_Release/broker/l386/PersistableMessage.obj \  
../Static_Release/broker/l386/QpiddBroker.obj \  
../Static_Release/broker/l386/Queue.obj \  
../Static_Release/broker/l386/Queue2.obj \  
../Static_Release/broker/l386/QueueBindings.obj \  
../Static_Release/broker/l386/QueueCleaner.obj \  
../Static_Release/broker/l386/QueueListeners.obj \  
..
```

```

./Static_Release/broker/I386/QueuePolicy.obj \
./Static_Release/broker/I386/QueueRegistry.obj \
./Static_Release/broker/I386/RateTracker.obj \
./Static_Release/broker/I386/RecoveredDequeue.obj \
./Static_Release/broker/I386/RecoveredEnqueue.obj \
./Static_Release/broker/I386/RecoveryManagerImpl.obj \
./Static_Release/broker/I386/SaslAuthenticator.obj \
./Static_Release/broker/I386/SemanticState.obj \
./Static_Release/broker/I386/Session.obj \
./Static_Release/broker/I386/SessionAdapter.obj \
./Static_Release/broker/I386/SessionHandler.obj \
./Static_Release/broker/I386/SessionManager.obj \
./Static_Release/broker/I386/SessionState.obj \
./Static_Release/broker/I386/System.obj \
./Static_Release/broker/I386/System2.obj \
./Static_Release/broker/I386/TCPIOPlugin.obj \
./Static_Release/broker/I386/Timer.obj \
./Static_Release/broker/I386/TopicExchange.obj \
./Static_Release/broker/I386/TxAccept.obj \
./Static_Release/broker/I386/TxBuffer.obj \
./Static_Release/broker/I386/TxPublish.obj \
./Static_Release/broker/I386/Vhost.obj \
./Static_Release/broker/I386/Vhost2.obj \
./Static_Release/broker/I386/qpidd.obj

qmfconsole_OBJECTS=../Static_Release/qmfconsole/I386/ClassKey.obj

lib:
lib.exe /OUT:"..\\qpidbrokers.lib" $(broker_OBJECTS)

5.4.2) tests.mk
C89_COMPILER=C:/Program Files (x86)/Microsoft Visual Studio 9.0/VC/bin/cl.exe
C89_LINKER=C:/Program Files (x86)/Microsoft Visual Studio 9.0/VC/bin/link.exe
LINK="link.exe"

OBJEXT=.obj
EXEEXT=.exe

INCLUDES=/I "C:/Program Files (x86)/boost/boost_1_36_0/include/103600" /I "C:/Program Files (x86)/boost/boost_1_36_0/." /I ".." /I ".." /I
"../gen"
FLAGS=/D "NDEBUG" /D "WIN32" /D "_CONSOLE" /D "_CRT_NONSTDC_NO_WARNINGS" /D "NOMINMAX" /D
"WIN32_LEAN_AND_MEAN" /D "_SCL_SECURE_NO_WARNINGS" /D "_CRT_SECURE_NO_WARNINGS" /FD /EHsc /MT /W3 /c /TP
/wd4244 /wd4800 /wd4290 /wd4355

lib_client=..\\qpidclients.lib
lib_common=..\\qpidcommons.lib
lib_broker=..\\qpidbrokers.lib
#lib_console=..\\qmfconsoles.lib
LDLIBS=..\\qpidclients.lib ..\\qpidcommons.lib ..\\qpidbrokers.lib
LIBS=ws2_32.lib secur32.lib rpcrt4.lib kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib
uuid.lib odbc32.lib odbc32.lib
LDFLAGS=/INCREMENTAL:NO /LIBPATH:." /LIBPATH:"C:\\Program Files (x86)\\boost\\boost_1_36_0\\lib" /DEBUG
/SUBSYSTEM:CONSOLE /OPT:REF /OPT:ICF /DYNAMICBASE /NXCOMPAT /MACHINE:X86 /nologo /errorReport:prompt

RELDIR=Static_Release
OBJDIR=$(RELDIR)/tests/I386
#UNIT_TEST_OBJDIR=$(RELDIR)/unit_test/I386
ROOTDIR=C:\\Users\\v-vinsea\\Dev\\AMQP\\qpid-M4\\cpp\\src\\tests

check_PROGRAMS=
check_LTLIBRARIES=
TESTS=
EXTRA_DIST=
CLEANFILES=

SOURCES=
UNIT_TEST_OBJECTS=

unit_test_SOURCES= \
ConsoleTest.cpp \
unit_test.cpp \
exception_test.cpp \
RefCounted.cpp \
SessionState.cpp Blob.cpp logging.cpp \
AsyncCompletion.cpp \
Url.cpp Uuid.cpp \
Shlib.cpp FieldValue.cpp FieldTable.cpp Array.cpp \
QueueOptionsTest.cpp \
InlineAllocator.cpp \
InlineVector.cpp \
ClientSessionTest.cpp \

```

```

SequenceSet.cpp \
StringUtil.cpp \
IncompleteMessageList.cpp \
RangeSet.cpp \
AtomicValue.cpp \
QueueTest.cpp \
AccumulatedAckTest.cpp \
DtxWorkRecordTest.cpp \
DeliveryRecordTest.cpp \
ExchangeTest.cpp \
HeadersExchangeTest.cpp \
MessageTest.cpp \
QueueRegistryTest.cpp \
QueuePolicyTest.cpp \
FramingTest.cpp \
HeaderTest.cpp \
SequenceNumberTest.cpp \
TimerTest.cpp \
TopicExchangeTest.cpp \
TxBufferTest.cpp \
TxPublishTest.cpp \
MessageBuilderTest.cpp \
ManagementTest.cpp \
MessageReplayTracker.cpp

perftest_SOURCES=perftest.cpp
txtest_SOURCES=txtest.cpp
latencytest_SOURCES=latencytest.cpp
echotest_SOURCES=echotest.cpp
client_test_SOURCES=client_test.cpp
topic_listener_SOURCES=topic_listener.cpp
topic_publisher_SOURCES=topic_publisher.cpp
publish_SOURCES=publish.cpp
consume_SOURCES=consume.cpp
header_test_SOURCES=header_test.cpp
failover_soak_SOURCES=failover_soak.cpp
declare_queues_SOURCES=declare_queues.cpp
replaying_sender_SOURCES=replaying_sender.cpp
resuming_receiver_SOURCES=resuming_receiver.cpp
txshift_SOURCES=txshift.cpp
txjob_SOURCES=txjob.cpp
receiver_SOURCES=receiver.cpp
sender_SOURCES=sender.cpp

SOURCES=$(SOURCES) $(unit_test_SOURCES)
SOURCES=$(SOURCES) $(perftest_SOURCES)
SOURCES=$(SOURCES) $(txtest_SOURCES)
SOURCES=$(SOURCES) $(latencytest_SOURCES)
SOURCES=$(SOURCES) $(echotest_SOURCES)
SOURCES=$(SOURCES) $(client_test_SOURCES)
SOURCES=$(SOURCES) $(topic_listener_SOURCES)
SOURCES=$(SOURCES) $(topic_publisher_SOURCES)
SOURCES=$(SOURCES) $(publish_SOURCES)
SOURCES=$(SOURCES) $(consume_SOURCES)
SOURCES=$(SOURCES) $(header_test_SOURCES)
#SOURCES=$(SOURCES) $(failover_soak_SOURCES)
SOURCES=$(SOURCES) $(declare_queues_SOURCES)
SOURCES=$(SOURCES) $(replaying_sender_SOURCES)
SOURCES=$(SOURCES) $(resuming_receiver_SOURCES)
SOURCES=$(SOURCES) $(txshift_SOURCES)
SOURCES=$(SOURCES) $(txjob_SOURCES)
SOURCES=$(SOURCES) $(receiver_SOURCES)
SOURCES=$(SOURCES) $(sender_SOURCES)

UNIT_TEST_OBJECTS= \
$(OBJDIR)/unit_test$(OBJEXT) \
$(OBJDIR)/exception_test$(OBJEXT) \
$(OBJDIR)/RefCounted$(OBJEXT) \
$(OBJDIR)/SessionState$(OBJEXT) \
$(OBJDIR)/Blob$(OBJEXT) \
$(OBJDIR)/logging$(OBJEXT) \
$(OBJDIR)/AsyncCompletion$(OBJEXT) \
$(OBJDIR)/Url$(OBJEXT) \
$(OBJDIR)/Uuid$(OBJEXT) \
$(OBJDIR)/Shlib$(OBJEXT) \
$(OBJDIR)/FieldValue$(OBJEXT) \
$(OBJDIR)/FieldTable$(OBJEXT) \
$(OBJDIR)/Array$(OBJEXT) \
$(OBJDIR)/QueueOptionsTest$(OBJEXT) \
$(OBJDIR)/InlineAllocator$(OBJEXT) \

```

```

$(OBJDIR)/InlineVector$(OBJEXT) \
$(OBJDIR)/ClientSessionTest$(OBJEXT) \
$(OBJDIR)/SequenceSet$(OBJEXT) \
$(OBJDIR)/StringUtil$(OBJEXT) \
$(OBJDIR)/IncompleteMessageList$(OBJEXT) \
$(OBJDIR)/RangeSet$(OBJEXT) \
$(OBJDIR)/AtomicValue$(OBJEXT) \
$(OBJDIR)/QueueTest$(OBJEXT) \
$(OBJDIR)/AccumulatedAckTest$(OBJEXT) \
$(OBJDIR)/DtxWorkRecordTest$(OBJEXT) \
$(OBJDIR)/DeliveryRecordTest$(OBJEXT) \
$(OBJDIR)/ExchangeTest$(OBJEXT) \
$(OBJDIR)/HeadersExchangeTest$(OBJEXT) \
$(OBJDIR)/MessageTest$(OBJEXT) \
$(OBJDIR)/QueueRegistryTest$(OBJEXT) \
$(OBJDIR)/QueuePolicyTest$(OBJEXT) \
$(OBJDIR)/FramingTest$(OBJEXT) \
$(OBJDIR)/HeaderTest$(OBJEXT) \
$(OBJDIR)/SequenceNumberTest$(OBJEXT) \
$(OBJDIR)/TimerTest$(OBJEXT) \
$(OBJDIR)/TopicExchangeTest$(OBJEXT) \
$(OBJDIR)/TxBufferTest$(OBJEXT) \
$(OBJDIR)/TxPublishTest$(OBJEXT) \
$(OBJDIR)/MessageBuilderTest$(OBJEXT) \
$(OBJDIR)/ManagementTest$(OBJEXT) \
$(OBJDIR)/MessageReplayTracker$(OBJEXT)

```

1. \$(OBJDIR)/ConsoleTest\$(OBJEXT) \

```

TESTS=$(TESTS) $(RELDIR)/unit_test$(EXEEXT)
TESTS=$(TESTS) $(RELDIR)/perftest$(EXEEXT)
TESTS=$(TESTS) $(RELDIR)/txtest$(EXEEXT)
TESTS=$(TESTS) $(RELDIR)/latencytest$(EXEEXT)
TESTS=$(TESTS) $(RELDIR)/echotest$(EXEEXT)
TESTS=$(TESTS) $(RELDIR)/client_test$(EXEEXT)
TESTS=$(TESTS) $(RELDIR)/topic_listener$(EXEEXT)
TESTS=$(TESTS) $(RELDIR)/topic_publisher$(EXEEXT)
TESTS=$(TESTS) $(RELDIR)/publish$(EXEEXT)
TESTS=$(TESTS) $(RELDIR)/consume$(EXEEXT)
TESTS=$(TESTS) $(RELDIR)/header_test$(EXEEXT)
#TESTS=$(TESTS) $(RELDIR)/failover_soak$(EXEEXT)
TESTS=$(TESTS) $(RELDIR)/declare_queues$(EXEEXT)
TESTS=$(TESTS) $(RELDIR)/replaying_sender$(EXEEXT)
TESTS=$(TESTS) $(RELDIR)/resuming_receiver$(EXEEXT)
TESTS=$(TESTS) $(RELDIR)/txshift$(EXEEXT)
TESTS=$(TESTS) $(RELDIR)/txjob$(EXEEXT)
TESTS=$(TESTS) $(RELDIR)/receiver$(EXEEXT)
TESTS=$(TESTS) $(RELDIR)/sender$(EXEEXT)

```

.cpp.obj:

```

@mkdir -p $(RELDIR)
@mkdir -p $(OBJDIR)
$(CC) /O2 $(INCLUDES) $(FLAGS) /Fo"$(OBJDIR)://" /Fd"$(OBJDIR)/vc90.pdb" $**

```

all: \$(SOURCES:.cpp=.obj) \$(TESTS)

```

$(RELDIR)/perftest$(EXEEXT): {$(OBJDIR)}$(perftest_SOURCES:.cpp=.obj)
$(LINK) /OUT:$(RELDIR)%|fF$(EXEEXT) /PDB:$(ROOTDIR)/$(RELDIR)%|fF.pdb /MANIFEST
/ManifestFile:"$(OBJDIR)%|fF$(EXEEXT).intermediate.manifest" /ManifestUAC:"level='asInvoker' uiAccess='false'" $(LDFLAGS)
$(LDLIBS) $(LIBS) %|pfF$(OBJEXT)

```

```

$(RELDIR)/txtest$(EXEEXT): {$(OBJDIR)}$(txtest_SOURCES:.cpp=.obj)
$(LINK) /OUT:$(RELDIR)%|fF$(EXEEXT) /PDB:$(ROOTDIR)/$(RELDIR)%|fF.pdb /MANIFEST
/ManifestFile:"$(OBJDIR)%|fF$(EXEEXT).intermediate.manifest" /ManifestUAC:"level='asInvoker' uiAccess='false'" $(LDFLAGS)
$(LDLIBS) $(LIBS) %|pfF$(OBJEXT)

```

```

$(RELDIR)/latencytest$(EXEEXT): {$(OBJDIR)}$(latencytest_SOURCES:.cpp=.obj)
$(LINK) /OUT:$(RELDIR)%|fF$(EXEEXT) /PDB:$(ROOTDIR)/$(RELDIR)%|fF.pdb /MANIFEST
/ManifestFile:"$(OBJDIR)%|fF$(EXEEXT).intermediate.manifest" /ManifestUAC:"level='asInvoker' uiAccess='false'" $(LDFLAGS)
$(LDLIBS) $(LIBS) %|pfF$(OBJEXT)

```

```

$(RELDIR)/echotest$(EXEEXT): {$(OBJDIR)}$(echotest_SOURCES:.cpp=.obj)
$(LINK) /OUT:$(RELDIR)%|fF$(EXEEXT) /PDB:$(ROOTDIR)/$(RELDIR)%|fF.pdb /MANIFEST
/ManifestFile:"$(OBJDIR)%|fF$(EXEEXT).intermediate.manifest" /ManifestUAC:"level='asInvoker' uiAccess='false'" $(LDFLAGS)
$(LDLIBS) $(LIBS) %|pfF$(OBJEXT)

```

```

$(RELDIR)/client_test$(EXEEXT): {$(OBJDIR)}$(client_test_SOURCES:.cpp=.obj)
$(LINK) /OUT:$(RELDIR)%|fF$(EXEEXT) /PDB:$(ROOTDIR)/$(RELDIR)%|fF.pdb /MANIFEST
/ManifestFile:"$(OBJDIR)%|fF$(EXEEXT).intermediate.manifest" /ManifestUAC:"level='asInvoker' uiAccess='false'" $(LDFLAGS)
$(LDLIBS) $(LIBS) %|pfF$(OBJEXT)

```

```

$(RELDIR)/topic_listener$(EXEEXT): {$(OBJDIR)}$(topic_listener_SOURCES:.cpp=.obj)
$(LINK) /OUT:$(RELDIR)%|fF$(EXEEXT) /PDB:$(ROOTDIR)/$(RELDIR)%|fF.pdb /MANIFEST
/ManifestFile:"$(OBJDIR)%|fF$(EXEEXT).intermediate.manifest" /ManifestUAC:"level='asInvoker' uiAccess='false'" $(LDFLAGS)
$(LDLIBS) $(LIBS) %|pfF$(OBJEXT)

$(RELDIR)/topic_publisher$(EXEEXT): {$(OBJDIR)}$(topic_publisher_SOURCES:.cpp=.obj)
$(LINK) /OUT:$(RELDIR)%|fF$(EXEEXT) /PDB:$(ROOTDIR)/$(RELDIR)%|fF.pdb /MANIFEST
/ManifestFile:"$(OBJDIR)%|fF$(EXEEXT).intermediate.manifest" /ManifestUAC:"level='asInvoker' uiAccess='false'" $(LDFLAGS)
$(LDLIBS) $(LIBS) %|pfF$(OBJEXT)

$(RELDIR)/publish$(EXEEXT): {$(OBJDIR)}$(publish_SOURCES:.cpp=.obj)
$(LINK) /OUT:$(RELDIR)%|fF$(EXEEXT) /PDB:$(ROOTDIR)/$(RELDIR)%|fF.pdb /MANIFEST
/ManifestFile:"$(OBJDIR)%|fF$(EXEEXT).intermediate.manifest" /ManifestUAC:"level='asInvoker' uiAccess='false'" $(LDFLAGS)
$(LDLIBS) $(LIBS) %|pfF$(OBJEXT)

$(RELDIR)/consume$(EXEEXT): {$(OBJDIR)}$(consume_SOURCES:.cpp=.obj)
$(LINK) /OUT:$(RELDIR)%|fF$(EXEEXT) /PDB:$(ROOTDIR)/$(RELDIR)%|fF.pdb /MANIFEST
/ManifestFile:"$(OBJDIR)%|fF$(EXEEXT).intermediate.manifest" /ManifestUAC:"level='asInvoker' uiAccess='false'" $(LDFLAGS)
$(LDLIBS) $(LIBS) %|pfF$(OBJEXT)

$(RELDIR)/header_test$(EXEEXT): {$(OBJDIR)}$(header_test_SOURCES:.cpp=.obj)
$(LINK) /OUT:$(RELDIR)%|fF$(EXEEXT) /PDB:$(ROOTDIR)/$(RELDIR)%|fF.pdb /MANIFEST
/ManifestFile:"$(OBJDIR)%|fF$(EXEEXT).intermediate.manifest" /ManifestUAC:"level='asInvoker' uiAccess='false'" $(LDFLAGS)
$(LDLIBS) $(LIBS) %|pfF$(OBJEXT)

$(RELDIR)/failover_soak$(EXEEXT): {$(OBJDIR)}$(failover_soak_SOURCES:.cpp=.obj)
$(LINK) /OUT:$(RELDIR)%|fF$(EXEEXT) /PDB:$(ROOTDIR)/$(RELDIR)%|fF.pdb /MANIFEST
/ManifestFile:"$(OBJDIR)%|fF$(EXEEXT).intermediate.manifest" /ManifestUAC:"level='asInvoker' uiAccess='false'" $(LDFLAGS)
$(LDLIBS) $(LIBS) %|pfF$(OBJEXT)

$(RELDIR)/declare_queues$(EXEEXT): {$(OBJDIR)}$(declare_queues_SOURCES:.cpp=.obj)
$(LINK) /OUT:$(RELDIR)%|fF$(EXEEXT) /PDB:$(ROOTDIR)/$(RELDIR)%|fF.pdb /MANIFEST
/ManifestFile:"$(OBJDIR)%|fF$(EXEEXT).intermediate.manifest" /ManifestUAC:"level='asInvoker' uiAccess='false'" $(LDFLAGS)
$(LDLIBS) $(LIBS) %|pfF$(OBJEXT)

$(RELDIR)/replaying_sender$(EXEEXT): {$(OBJDIR)}$(replaying_sender_SOURCES:.cpp=.obj)
$(LINK) /OUT:$(RELDIR)%|fF$(EXEEXT) /PDB:$(ROOTDIR)/$(RELDIR)%|fF.pdb /MANIFEST
/ManifestFile:"$(OBJDIR)%|fF$(EXEEXT).intermediate.manifest" /ManifestUAC:"level='asInvoker' uiAccess='false'" $(LDFLAGS)
$(LDLIBS) $(LIBS) %|pfF$(OBJEXT)

$(RELDIR)/resuming_receiver$(EXEEXT): {$(OBJDIR)}$(resuming_receiver_SOURCES:.cpp=.obj)
$(LINK) /OUT:$(RELDIR)%|fF$(EXEEXT) /PDB:$(ROOTDIR)/$(RELDIR)%|fF.pdb /MANIFEST
/ManifestFile:"$(OBJDIR)%|fF$(EXEEXT).intermediate.manifest" /ManifestUAC:"level='asInvoker' uiAccess='false'" $(LDFLAGS)
$(LDLIBS) $(LIBS) %|pfF$(OBJEXT)

$(RELDIR)/txshift$(EXEEXT): {$(OBJDIR)}$(txshift_SOURCES:.cpp=.obj)
$(LINK) /OUT:$(RELDIR)%|fF$(EXEEXT) /PDB:$(ROOTDIR)/$(RELDIR)%|fF.pdb /MANIFEST
/ManifestFile:"$(OBJDIR)%|fF$(EXEEXT).intermediate.manifest" /ManifestUAC:"level='asInvoker' uiAccess='false'" $(LDFLAGS)
$(LDLIBS) $(LIBS) %|pfF$(OBJEXT)

$(RELDIR)/txjob$(EXEEXT): {$(OBJDIR)}$(txjob_SOURCES:.cpp=.obj)
$(LINK) /OUT:$(RELDIR)%|fF$(EXEEXT) /PDB:$(ROOTDIR)/$(RELDIR)%|fF.pdb /MANIFEST
/ManifestFile:"$(OBJDIR)%|fF$(EXEEXT).intermediate.manifest" /ManifestUAC:"level='asInvoker' uiAccess='false'" $(LDFLAGS)
$(LDLIBS) $(LIBS) %|pfF$(OBJEXT)

$(RELDIR)/receiver$(EXEEXT): {$(OBJDIR)}$(receiver_SOURCES:.cpp=.obj)
$(LINK) /OUT:$(RELDIR)%|fF$(EXEEXT) /PDB:$(ROOTDIR)/$(RELDIR)%|fF.pdb /MANIFEST
/ManifestFile:"$(OBJDIR)%|fF$(EXEEXT).intermediate.manifest" /ManifestUAC:"level='asInvoker' uiAccess='false'" $(LDFLAGS)
$(LDLIBS) $(LIBS) %|pfF$(OBJEXT)

$(RELDIR)/sender$(EXEEXT): {$(OBJDIR)}$(sender_SOURCES:.cpp=.obj)
$(LINK) /OUT:$(RELDIR)%|fF$(EXEEXT) /PDB:$(ROOTDIR)/$(RELDIR)%|fF.pdb /MANIFEST
/ManifestFile:"$(OBJDIR)%|fF$(EXEEXT).intermediate.manifest" /ManifestUAC:"level='asInvoker' uiAccess='false'" $(LDFLAGS)
$(LDLIBS) $(LIBS) %|pfF$(OBJEXT)

$(RELDIR)/unit_test$(EXEEXT): {$(OBJDIR)}$(unit_test_SOURCES:.cpp=.obj)
$(LINK) /OUT:$(RELDIR)%|fF$(EXEEXT) /PDB:$(ROOTDIR)/$(RELDIR)%|fF.pdb /MANIFEST
/ManifestFile:"$(OBJDIR)%|fF$(EXEEXT).intermediate.manifest" /ManifestUAC:"level='asInvoker' uiAccess='false'" $(LDFLAGS)
$(LDLIBS) $(LIBS) $(UNIT_TEST_OBJECTS)

```

Split configuration files

Qpid 0.5 and later supports using multiple configuration files. This is useful for separating server-wide and instance-specific configuration settings.

The top level combined configuration file might contain the following:

```

<configuration>
  <system/>
  <xml fileName="/path/to/config"/>
  <xml fileName="/path/to/local/config"/>
</configuration>

```

The `<system/>` element is important to ensure proper substitution of system properties.

The file referenced by `"/path/to/config"` might contain:

```

<broker>
  <management>
    <enabled>true</enabled>
    <ssl>
      <enabled>true</enabled>
      <keyStorePath>${prefix}../test_resources/ssl/keystore.jks</keyStorePath>
      <keyStorePassword>password</keyStorePassword>
    </ssl>
  </management>
</broker>

```

and `"/path/to/local/config"`:

```

<broker>
  <management>
    <jmxport>8999</jmxport>
  </management>
</broker>

```

The values from `/path/to/config` would override any set in `/path/to/local/config` since they are defined first, but this would result in all instances using the same keyStore and keyStorePassword without running on the same jmxport (whether this is a good idea to follow or not obviously depends on your environment).

A more complete example can be found attached to this page:

| Name | Size | Creator | Creation Date | Comment |
|-------------------------|------|---------------|--------------------|---------|
| split-config-part1.xml | 3 kB | Aidan Skinner | Apr 24, 2009 03:47 | |
| split-config-master.xml | 1 kB | Aidan Skinner | Apr 24, 2009 03:47 | |
| split-config-part2.xml | 1 kB | Aidan Skinner | Apr 24, 2009 06:10 | |

Tune Broker and Client Memory Usage

Tuning the broker for your message size.

The default buffer size used per message on the broker and client is 32kb if your message is significantly smaller you can improve your memory usage by lowering this value.

What size to use

When selecting what size of buffer to include space for any JMS Headers that may be defined (key and value). You should also include 200 bytes for the AMQP routing details, if you have very long queue or topic names you may wish to increase this value further.

How to change the buffer sizes

Broker Buffers

In your broker configuration file the `socketSend/ReceiveBuffer` value of 32768 is where the buffer size is specified. Currently your configuration file will contain the following two entries:

```

<connector>
  ...
  <socketReceiveBuffer>32768</socketReceiveBuffer>
  <socketSendBuffer>32768</socketSendBuffer>
</connector>

```

Modifying these will adjust the size of the ByteBuffers used in conjunction with the socket.

Client Buffers

Adjusting the client buffers can also assist your client heap management if you are prefetching a large number of messages. However, adjusting this will not have any affect on the broker's memory usage. If you also want to modify the client buffer size then there are two system properties that need set:

```

amqj.sendBufferSize
amqj.receiveBufferSize

```

These need to be set prior to making the initial connection. For more details on these properties see: [System Properties](#)

Use Last Value Queues (LVQ)

General Information

The Qpid 0.7 release introduces Last Value Queues (LVQs) into the Java Messaging Broker. The LVQ implementation in the Java Broker provides a superset of the behaviour of the LVQ in the C++ Broker. It is hoped that later revisions of the C++ Broker will adopt the same enhanced semantics.

LVQ Semantics

In an LVQ messages in the queue are be "replaced" by newer messages having the same value for some specified header. An example of an LVQ might be where a queue representing prices on a stock exchange: when you first consume from the queue you get the latest quote for each stock, and then as new prices come in you are sent only these updates.

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

Defining LVQs

You must define an LVQ specifically before you start to use it. You cannot subsequently change a queue to/from an LVQ (without deleting it and re-creating). Also note that a queue cannot be both an LVQ and a Priority Queue. If a queue is defined with both LVQ and Priority Queue attributes, it will only take on an LVQ nature.

You define a queue as an LVQ in the virtualhost configuration file, which the broker loads at startup. When defining the queue, add a `<lvq>true</lvq>` element. Without any further configuration this will define an LVQ which uses the header "qpид.LVQ_key" as the key for replacement. If you wish to define your own key then you can do so using the `<lvqKey>myKey</lvqKey>` element (e.g. `<lvqKey>myKey</lvqKey>` will use the header "myKey" as the replacement key).

```

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

```

LVQs can also be declared by setting arguments in the Queue.Declare AMQP command.

```

Map<String, Object> arguments = new HashMap<String, Object>();
arguments.put("qpид.last_value_queue", true);
arguments.put("qpид.last_value_queue_key", "key");
(AMQSession) session.createQueue(queueName, autoDelete, durable, exclusive, arguments);

```

Receiving messages from an LVQ

When receiving messages from an LVQ you may wish to change the default pre-fetch setting for your client. With a large pre-fetch value message will be sent from the queue as soon as they arrive, and then be buffered on the client. This may lead to cases where newer versions of messages are buffered behind older versions within the client. With a lower pre-fetch value the broker has an opportunity to replace older versions of the message with newer versions before sending to the client.

Set low pre-fetch

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

In order to set pre-fetch 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 conflation 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 connection url

```
amqp://guest:guest@client1/development?maxprefetch='1'&brokerlist='tcp://localhost:5672'
```

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 conflation is weighted towards the former for your application. (If you're using OnMessage() then this is not a concern.)

Implication of low pre-fetch

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

Browsing an LVQ from Java

One way to use an LVQ is to have a single queue which receives all updates, and many subscribers which only "browse" the queue without actually consuming messages. Unfortunately the standard JMS MessageBrowser is not suitable for this use case as it closes at the point where there are no more updates to be received. Instead we wish to use a browser more like a standard QueueReceiver, but with the caveat that the receiver is not actually consuming messages from the queue.

To do this we can specify in the URL for a destination that any created consumer should be browse only, e.g.:

```
direct://amq.direct//myqueue?browse='true'
```

Note that browsing will only work on consumers in NO_ACKNOWLEDGE mode.

Browsing an LVQ from .net

To allow the creation of a "consumer" in browse-only mode from the AMQP 0-8 .net client the following method has been added to the API for IChannel:

```
IMessageConsumer CreateConsumer(string queueName,
                                int prefetchLow,
                                int prefetchHigh,
                                bool noLocal,
                                bool exclusive,
                                bool browse);
```

where passing browse as true will get the desired behaviour.

Similarly the following method has been added to MessageConsumerBuilder:

```
public MessageConsumerBuilder WithBrowse(bool browse)
```

Note that browsing will only work on consumers in NO_ACKNOWLEDGE mode.

Use Priority Queues

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.

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>
```

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.

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 [connection url](#)

```
amqp://guest:guest@client1/development?maxprefetch='1'&brokerlist='tcp://localhost:5672'
```

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.)

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.

Use Producer Flow Control

General Information

The Qpid 0.6 release introduces 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.

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 capacity to 10Mb -->
    <flowResumeCapacity>8388608</flowResumeCapacity> <!-- set the resume capacity to 8Mb -->
  </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 capacity to 10Mb
-->
      <flowResumeCapacity>8388608</flowResumeCapacity> <!-- set the resume capacity to 8Mb
-->
    </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.

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 AMQSession - Broker enforced flow control has been enforced
WARN AMQSession - Message send delayed by 5s due to broker enforced flow control
WARN AMQSession - Message send delayed by 10s due to broker enforced flow control
```

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

```
ERROR AMQSession - Message send failed due to timeout waiting on broker enforced flow control
```

If such a JMSEException 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 rollback 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 default is 120000 - which is two minutes), 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: the default value is 5000 milliseconds (i.e. 5 seconds).

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
```

Older Clients

This feature was added for the 0.6 release of the Java Broker. If an older client connects to the broker then the flow control commands will be ignored and they will not be blocked. So to fully benefit from this new feature both Client and Broker need to be at least version 0.6.

Broker Log Messages

There are four new Broker log messages that may occur if flow control through queue capacity limits is enabled.

Firstly, when a capacity limited queue becomes overfull, a log message similar to the following is produced

```
MESSAGE [vh(/test)/qu(MyQueue)] [vh(/test)/qu(MyQueue)] QUE-1003 : Overfull : Size : 1,200 bytes,  
Capacity : 1,000
```

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(713889609)/test)/ch:1] CHN-1005 : Flow Control Enforced (Queue MyQueue)
```

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 : Size : 600 bytes,  
Resume Capacity : 800
```

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(713889609)/test)/ch:1] CHN-1006 : Flow Control Removed
```

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

Qpid Java Run Scripts

Qpid Java Broker Run Scripts

The following scripts are used to run the Qpid broker:

qpid-server
qpid-server.bat
qpid-run

These scripts are described in more detail below.

qpid-server

Overview

This script starts the Qpid Java Broker on Linux/Solaris/Cygwin platforms.

It is extremely simple, delegating the real work to the qpid-run script.

In fact, all it really provides is the main class to execute and passes through any command line arguments to qpid-run i.e.

```
. qpid-run org.apache.qpid.server.Main "$@"
```

qpid-server.bat

Overview

This script starts the Qpid Java Broker on Windows platforms. It provides a limited version of the qpid-run functionality, though is not nearly as sophisticated i.e. does not support run arguments or the full set of argument variables.

However, it does support the following features:

- validates that JAVA_HOME is set
- validates that QPID_HOME is set
- passes any command line arguments to the main broker class
- supports the use of QPID_OPTS to pass through java system properties

Note that a JIRA exists for enhancing the features this script supports <http://issues.apache.org/jira/browse/QPID-168>

qpид-run

Overview

The qpid-run script allows the calling program to run any given command, and provides a flexible surround supporting configurable runtime arguments for the script itself, the broker and java arguments.

Environment Variables and Defaulting

The variables noted below are used by the qpid-run script. Any default value used if not specified is noted below.

| Variable | Description | Default |
|--------------------|---|------------------------|
| QPID_HOME | Used as root for installed application path. Mandatory that users set this | None |
| QPID_WORK | Used as root for any working directories to which the Qpid broker writes, for logging and bdb etc | Current User's Homedir |
| AMQJ_LOGGING_LEVEL | Logging level for broker code | info |
| QPID_LOG_PREFIX | Used as a prefix for qpid broker log, see FAQ for more details | None |
| QPID_LOG_SUFFIX | Used as a suffix for qpid broker log, see FAQ for more details | None |
| JPDA_OPTS | If set and -run:jpda argument provided used for debugging props, see below | None |
| QPID_OPTS | Use to pass custom system properties, including management console connection info | None |
| JAVA_OPTS | Use to pass custom Java options, for example gc options etc | None |

Run Arguments

You can provide run arguments to the qpid-run script using the syntax

```
-run:argument
```

The table below provides details of the available arguments.

| Argument | Description |
|--------------------|---|
| debug | Prints classpath and command before running it |
| jpda | Adds remote debugging info using JPDA_OPTS. Use JPDA_TRANSPORT and JPDA_ADDRESS to customize, JPDA_OPTS to override |
| external-classpath | Valid values are: ignore, first, last and only. See below for more info |
| print-classpath | Prints classpath before running command |
| help | Prints Usage information |

Qpid Troubleshooting Guide

Contents

- I'm getting a `java.lang.UnsupportedClassVersionError` when I try to start the broker. What does this mean ?
- I'm having a problem binding to the required host:port at broker startup ?
- I'm having problems with my classpath. How can I ensure that my classpath is ok ?
- I can't get the broker to start. How can I diagnose the problem ?
- When I try to send messages to a queue I'm getting a error as the queue does not exist. What can I do ?

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/qpidd/server/Main (Unsupported major.minor version 49.0)`

```
at java.lang.ClassLoader.defineClass(Ljava.lang.String;[B)I)Ljava.lang.Class;(Unknown Source)
at
java.security.SecureClassLoader.defineClass(Ljava.lang.String;[B)I)Ljava.lang.Class;(SecureClassLoader.java:12
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
at java.net.URLClassLoader$1.run()Ljava.lang.Object;
(URLClassLoader.java:194)
at
jrockit.vm.AccessController.do_privileged_exc(Ljava.security.PrivilegedExceptionAction;Ljava.security.AccessControlContext;I)Ljava.lang.Obj
Source)
at
jrockit.vm.AccessController.doPrivileged(Ljava.security.PrivilegedExceptionAction;Ljava.security.AccessControlContext;)Ljava.lang.Object;(U
Source)
at java.net.URLClassLoader.findClass(Ljava.lang.String;)Ljava.lang.Class;(URLClassLoader.java:187)
at java.lang.ClassLoader.loadClass(Ljava.lang.String;Z)Ljava.lang.Class; (Unknown Source)
at sun.misc.Launcher$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)
```

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>

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.

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.

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 [Qpid Java FAQ](#) for more details of using the virtualhosts.xml route.

Release Plans

Currently we have tentative plans for the following releases of the Qpid Java Broker & Client:

M1 - November 2006

The content for this release is available via our JIRA release notes:

<https://issues.apache.org/jira/secure/ReleaseNote.jspa?version=12312115&styleName=Html&projectId=12310520>

M2 - December 2006

Details of content tbc shortly.

roadmap

looking to pitch in Note that this page only lists what has been added in each release

At some point we should get a full list of features published. If you don't see something, mail the users@qpid.apache.org list and ask. good chance we can do it 😊

M4 Adds the following Features - Current release

- .NET, WCF and excel support for AMQP 0-10
- SSL added for C++ broker and all clients
- Windows port for C++ client & broker
- Solaris port for C++ client & broker
- C++ Broker
 - ACL
 - Active-Active clustering
 - Federation, push bridges & dynamic routes
 - RDMA for C++ broker & C++ client (70-80us, yes us max latency on a well setup machines)
 - support for message TTL
 - Queue options
 - added RING/ STRICT ring
 - LVQ
 - Exchange options
 - LVE
 - message sequencing
 - XQuery based XML Exchange now as plugin
- Performance work
- Management for AMQP 0-10
 - QMF C updates
 - Python
 - C++
 - QMF Agent
 - C++
 - QMan JMX bridge for QMF
 - Alerts/ logger for QMF events
- JMSXUserId
- Java broker
 - Message Priority
 - bug fixes
 - some prep work for AMQP 0-10

M3 This release implements the AMQP 0-9 & AMQP 0-10 protocol versions

Top level themes for this release:

- C++ broker supporting AMQP 0-10
- C++ client supporting AMQP 0-10
- Java Client support AMQP 0-9 (M2.1) and AMQP 0-10
- Python Client supporting AMQP 0-9 (M2.1) and AMQP 0-10 (in addition to older versions)
- Landed the 0-10 infra for Java Broker
- Federation static routes
- Solaris Client C++ support
- bug fixes
- QMF for C++ / Python.

JIRA for M3:

<http://issues.apache.org/jira/secure/IssueNavigator.jspa?reset=true&mode=hide&sorter/order=DESC&sorter/field=priority&resolution=-1&pid=>

M2.1

M2.1 is a maintainer release for the M2 code branch. The detailed breakdown of JIRA's for M2.1 can be found here:

This release implements the the AMQP 0-9 protocol version

<http://issues.apache.org/jira/secure/IssueNavigator.jspa?reset=true&mode=hide&sorter/order=DESC&sorter/field=priority&resolution=-1&pid=>

looking to pitch in

Best way to get involved is mail dev@qpid.apache.org and say, "want to help, and state your interests"

Want needs to get done (the short term/ short list)

| Theme | Item | component | who |
|-------------|------|-----------|-----|
| Performance | | | |

| | | | |
|------------------------|---|--------------------|--|
| | increase fanout performance | c++ | |
| | increase per-connection throughput | c++ | |
| | Shared memory transport | c++ | |
| Broker 'model' | | | |
| | Selectors | c++ | |
| | queue policies **1 | c++ | |
| | Priorities | c++ | |
| | PB exchange | c++ | |
| Security | | | |
| | SASL authentication (kerb) | python | |
| | SASL authentication (kerb) | java | |
| | SASL encryption (kerb) | python | |
| | SASL encryption (kerb) | java | |
| | SASL encryption (kerb) | .net | |
| | SASL encryption & authentication | inter broker links | |
| | Tests for federation and clustering with all transport and security options | c++ | |
| | SELinux ACL delegation | c++ | |
| Clustering | | | |
| | Session resume across | all client | |
| | Processing updates to known broker urls | python and ruby | |
| | auto detect restart durable stores for cluster | c++ | |
| Management | | | |
| | more stats (e.g. avg/min/max queue 'latencies') | c++ | |
| | configurable alerts for e.g. hitting preset queue limits | c++ | |
| | more flexible logging (more useful to end user) | c++ | |
| | QMF agents for missing clients | java, python, .net | |
| | QMF console API for | .net | |
| | JMX -> QMF bridge | java | |
| | qpid-queue-tool | python | |
| Builds | | | |
| | c++ on windows (per steve) | c++ | |
| | complete solaris | c++ | |
| | default IO for other OS's ? | c++ | |
| Future Proofing | | | |
| | High level API design work | All client | |
| | ABI computability strategy | c++ | |
| | .Net test coverage and API review (WCF/QMF) | .net | |
| Interop | | | |
| | 0-10 for Java broker | java | |
| | Updated IO layer for Java broker | java | |
| Miscellaneous | | | |
| | test card i warp support for RDMA | c++ | |

**1

- allow for killing (slow) consumer, rather than producer – python cmd line tool?
- dequeue when all current browsers have received message (more efficient topic, also represents a step towards 1.0 model)

Target Features/work areas for Qpid 0.5

- Kerberos Auth and encryption from all clients – in progress
- A-A Cluster support from all clients
- RT JVM Thread support for Java — done
- DR links for stand-alone and clustered brokers — done
- Producer side flow control — done
- Updated IO layer for Java
- Shared memory transport
- .NET test coverage
- Windows client build from cmd line.
- AMQP-MGMT to WS-DM and JMX bridge committed (QMan) – mostly done
- IP address /Hostname access control to virtualhosts (Java broker)

scratch list of stuff to help with

You can also look in JIRA under the starter section.

Here are some additional things... The following are nice task to pick up for new committers, (Please mail the list if you have interest in doing any of these)

- Creation of Perl Client — Nice task to start on
- Integrating AMQP-MGMT into the Java broker
- Writing examples and getting started pages — Nice task to start on
- Help update this site  — Nice task to start on
- Profiling for performance improvements
- Spring integration
- JCA / App server integrations
- C++ broker
 - Message Priority
 - Selectors
 - Lock Free queue implementation
- Java broker
 - Transparent 0.8/0.9/1.0 support in the Java broker
 - Add Flow To Disk
 - Message Federation for Java (Fan out & forward to remote queue) – look at C++ broker and impl in Java (might need to wait for 0-10 (M4))
 - Flow control - throttling of overactive producers – might need 0-10 (M4)

Sustained Tests

Pub/Sub Sustained Tests

We currently have one sustained test for pub / sub messaging. The test is based on the interop testing framework and as such there are two classes that are involve.

1. org.apache.qpid.sustained.TestClient
2. org.apache.qpid.sustained.TestCoordinator

As with the [Interop Tests] the Test Coordinator collects various clients to work on the specified test. The sustained test suit currently only has one test which is the SustainedTestClient.

SustainedTestClient Test

This test is a pub/sub test. There is a single publisher that sends batches of messages to a known topic. The clients then receive these messages and report the time required to retrieve all of the batch. This reported time is sent to the publisher so that it can adjust its publication rate to ensure that messages are sent at a rate that all clients can maintain.

Usage

The coordinator can take a number of parameters.

- numReceives (Default : 2)
This is the number of receivers each client node should create
- batchSize (Default : 1000)
This is the number of messages to send per batch
- ackMode (Default : 1 - AUTO_ACK)
The acknowledgement mode to use. Currently No_ack (257) appears not to work correctly.

The client can take the following system parameters (set via -D properties)
(These values should be moved to the Coordinator so that you do not need to guess which client will become the sender as these values are only of use to the sending client.)

- sleepPerMessage (Default : false)
Divides the current _delay value into small sleeps between messages. Useful if your Thread.sleep implementation works at the nanosecond level. Under windows the smallest sleep value is around 10ms.
- warmUpBatches (Default : 10)
Adjusts the number of batches sent before resetting the delay calculations.
- stableReportCount (Default : 5)
The number of reports that need to arrive without changing the delay before reporting the delay is stable.
- batchVariance (Default : 3)
The difference between the batch number sent and the received batch number.

The client also has one additional parameter.

- "-j" can be used to join an existing test run. Despite there being only one test case at present this class must be provided as future sustained test classes may be available.

```
java -cp <qpid.jar> org.apache.qpid.sustained.TestClient -n client2 -j  
org.apache.qpid.performance.sustainedrate.SustainedTestClient
```

It is important to remember that each client must be uniquely named for the test to accurately work. This can be done by ensuring that all clients specify a unique "-n" value. So running

```
java -cp <qpid.jar> org.apache.qpid.sustained.TestClient -n client2
```

Will correctly name the client 'client2' and attempt to join the client to any running SustainedTestClient test. If there is no test running then the client will simply wait for the test to start.

System Properties

Explanation of System properties used in Qpid

This page documents the various System Properties that are currently used in the Qpid Java code base.

- Client Properties
 - STRICT_AMQP
 - Features disabled by STRICT_AMQP
 - STRICT_AMQP_FATAL
 - IMMEDIATE_PREFETCH
 - amqj.default_syncwrite_timeout
 - amq.dynamicsaslsregistrar.properties
 - amqj.heartbeat.timeoutFactor
 - amqj.tcpNoDelay
 - amqj.sendBufferSize
 - amqj.receiveBufferSize
 - amqj.protocolprovider.class
 - amqj.protocol.logging.level
 - jboss.host
 - jboss.port
 - amqj.MaximumStateWait
- Management Properties
 - security
 - jmxconnector
 - timeout
- Properties used in Examples
 - archivepath

Client Properties

STRICT_AMQP

Type : boolean

Default : FALSE

This forces the client to only send AMQP compliant frames. This will disable a number of JMS features.

Features disabled by STRICT_AMQP

- Queue Browser
- Message Selectors
- Durable Subscriptions

- Session Recover may result in duplicate message delivery
- Destination validation, so no InvalidDestinationException will be thrown

This is associated with property **STRICT_AMQP_FATAL**

STRICT_AMQP_FATAL

Type : boolean
Default : FALSE

This will cause any attempt to utilise an enhanced feature to throw an UnsupportedOperationException. When set to false then the exception will not occur but the feature will be disabled.

e.g.
The Queue Browser will always show no messages.
Any message selector will be removed.

IMMEDIATE_PREFETCH

Type : boolean
Default : FALSE

The default with AMQP is to start prefetching messages. However, with certain 3rd party Java tools, such as Mule this can cause a problem. Mule will create a consumer but never consume from it so any any prefetched messages will be stuck until that session is closed. This property is used to re-instate the default AMQP behaviour. The default Qpid behaviour is to prevent prefetch occurring, by starting the connection Flow Controlled, until a request for a message is made on the consumer either via a receive() or setting a message listener.

amqj.default_syncwrite_timeout

Type : long
Default: 30000
The number length of time in millisecond to wait for a synchronous write to complete.

amq.dynamicsasregistrar.properties

Type : String
Default: org/apache/qpid/client/security/DynamicSaslRegistrar.properties
The name of the SASL configuration properties file.

amqj.heartbeat.timeoutFactor

Type : float
Default : 2.0
The factor used to get the timeout from the delay between heartbeats

amqj.tcpNoDelay

Type : boolean
Default : TRUE
Disable Nagle's algorithm on the TCP connection.

amqj.sendBufferSize

Type : integer
Default : 32768
This is the default buffer sized created by Mina.

amqj.receiveBufferSize

Type : integer
Default : 32768
This is the default buffer sized created by Mina.

amqj.protocolprovider.class

Type : String
Default : org.apache.qpid.server.protocol.AMQPFastProtocolHandler
This specifies the default IoHandlerAdapter that represents the InVM broker. The IoHandlerAdapter must have a constructor that takes a single Integer that represents the InVM port number.

amqj.protocol.logging.level

Type : boolean
Default : null
If set this will turn on protocol logging on the client

jboss.host

Used by the JBossConnectionFactoryInitialiser to specify the host to connect to perform JNDI lookups.

jboss.port

Used by the JBossConnectionFactoryInitialiser to specify the port to connect to perform JNDI lookups.

amqj.MaximumStateWait

Default : 30000

Used to set the maximum time the State Manager should wait before timing out a frame wait.

Management Properties

security

Default: null

String representing the Security level to be used to on the connection to the broker. The null default results in no security or PLAIN. When used with jmxconnector 'javax.management.remote.jmxmp.JMXMPConnector' a security value of 'CRAM-MD5' will result in all communication to the broker being encrypted.

jmxconnector

Default: null

String representing the JMXConnector class used to perform the connection to the broker. The null default results in the standard JMX connector. Utilising 'javax.management.remote.jmxmp.JMXMPConnector' and security 'CRAM-MD5' will result in all communication to the broker being encrypted.

timeout

Default: 5000

Long value representing the milli seconds before connection to the broker should timeout.

Properties used in Examples

archivepath

Used in : FileMessageDispatcher

This properties specifies the directory to move payload file(s) to archive location as no error

URL Formats

URL Format for connections and binding

There are currently two formats implemented in the Java code base. One is for connection and the other is an exchange binding URL. The URL formats have been designed around the Java URI format. This allows the parsing majority of the parsing work to be handled for us.

- **Connection URL Format** - The format used to describe a connection.
- **BindingURLFormat** - The format used for creating bindings within and to a broker.

The C++ broker uses a different connection URL format and has no binding URL format.

There is a new [Url Format Proposal](#) to define a single format for Qpid and also is being submitted to the AMQP working group for standardization.

0.10 Connection URL Format

```
qpid:[<property>=<value>[ ;<property>=<value> ] ]@<transport>:<host>[:<port>]
```

Currently handled properties

| Option | Default | Description |
|-------------|---------|--|
| virtualhost | "" | Set the virtual host to be used (this is currently not supported by the 0.10 c++ broker) |
| username | guest | The user name for the client. |
| password | guest | The password used for creating a connection. |

| | | |
|-----------|----------------|---|
| client_id | auto-generated | The client identifier used for creating a connection. |
|-----------|----------------|---|

Currently handled transports

Currently only 'tcp' transport is supported. 'tls' will however be supported and additional properties like 'keystore' and 'keystorelocation' will be added. The 'vm' transport should also be supported when a 0.10 Java broker will be available.

| Option | Default | Description |
|--------|---------|----------------------|
| tcp | true | Use a TCP connection |
| tls | false | Use a tls connection |

Host port

The default host port used is 5672 for 'tcp'.

Failover

It is planned to introduce a failover property for controlling how failover occurs when presented with a list of brokers. This is however not yet supported.

Sample URLs

```
qpid:@tcp:myHost
qpid:client_id=myClientIDpassword=myPassword;username=myUsername@tcp:myHost:5644
```

BindingURLFormat

```
<Exchange Class>://<Exchange
Name>/[<Destination>]/[<Queue>][?<option>='<value>' [&<option>='<value>']]
```

This URL format is used for two purposes in the code base. The broker uses this in the XML configuration file to create and bind queues at broker startup. It is also used by the client as a destination.

This format was used because it allows an explicit description of exchange and queue relationship.

The Exchange Class is not normally required for client connection as clients only publish to a named exchange however if exchanges are being dynamically instantiated it will be required. The class is required for the server to instantiate an exchange.

There are a number of options that are currently defined:

| Option | type | Description |
|--------------|---------|--|
| exclusive | boolean | Is this an exclusive connection |
| autodelete | boolean | Should this queue be deleted on client disconnection |
| durable | boolean | Create a durable queue |
| clientid | string | Use the following client id |
| subscription | boolean | Create a subscription to this destination |
| routingkey | string | Use this value as the routing key |

Using these options in conjunction with the Binding URL format should allow future expansion as new and custom exchange types are created.

The URL format requires **that at least one** Queue or routingkey option be present on the URL.

The routingkey would be used to encode a topic as shown in the examples section below.

Examples

Queues

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

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

For example: direct://amq.direct//simpleQueue

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

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.

```
direct://amq.direct/SimpleQueue  
direct://amq.direct/UnusuallyBoundQueue?routingkey='queue'  
topic://amq.topic?routingkey='stocks.#'  
topic://amq.topic?routingkey='stocks.nyse.ibm'
```

Connection URL Format

Format

```
amqp://[<user>:<pass>@][<clientid>]<virtualhost>[?<option>='<value>' [&<option>='<value>']]
```

The connection url defines the values that are common across the cluster of brokers. The virtual host is second in the list as the AMQP specification demands that it start with a '/' otherwise it be more readable to be swapped with clientid. There is currently only one required option and that is the **brokerlist** option. In addition the following options are recognised.

Worked Example

You could use a URL which looks something like this:

```
amqp://guest:guest@client1/development?brokerlist='tcp://localhost:5672'
```

Breaking this example down, here's what it all means:

amqp = the protocol we're using

guest:guest@localhost = username:password@clientid where the clientid is the name of your server (used under the covers but don't worry about this for now). Always use the guest:guest combination at the moment.

development = the name of the virtualhost, where the virtualhost is a path which acts as a namespace. You can effectively use any value here so long as you're consistent throughout. The virtualhost must start with a slash "/" and continue with names separated by slashes. A name consists of any combination of at least one of [A-Za-z0-9] plus zero or more of [._+!=:].

brokerlist = this is the host address and port for the broker you want to connect to. The connection factory will assume tcp if you don't specify a transport protocol. The port also defaults to 5672. Naturally you have to put at least one broker in this list.

This example is not using failover so only provides one host for the broker. If you do wish to connect using failover you can provide two (or more) brokers in the format:

```
brokerlist='tcp://host1&tcp://host2:5673'
```

The default failover setup will automatically retry each broker once after a failed connection. If the brokerlist contains more than one server then these servers are tried in a round robin. Details on how to modify this behaviour will follow soon !

Options

| Option | Default | Description |
|-------------|-----------|---|
| brokerlist | see below | The list of brokers to use for this connection |
| failover | see below | The type of failover method to use with the broker list. |
| maxprefetch | 5000 | The maximum number of messages to prefetch from the broker. |

Brokerlist option

```
brokerlist='<broker url>[;<broker url>]'
```

The broker list defines the various brokers that can be used for this connection. A minimum of one broker url is required additional URLs are semi-colon(';) delimited.

Broker URL format

```
<transport>://<host>[:<port>][?<option>='<value>' [&<option>='<value>']]
```

There are currently quite a few default values that can be assumed. This was done so that the current client examples would not have to be re-written. The result is if there is no transport, 'tcp' is assumed and the default AMQP port of 5672 is used if no port is specified.

| Transport |
|-----------|
| tcp |
| vm |

Currently only 'tcp' and 'vm' transports are supported. Each broker can take have additional options that are specific to that broker. The following are currently implemented options. To add support for further transports the "client.transportTransportConnection" class needs updating along with the parsing to handle the transport.

| Option | Default | Description |
|----------------|---------|---|
| retries | 1 | The number of times to retry connection to this Broker |
| ssl | false | Use ssl on the connection |
| connecttimeout | 30000 | How long in (milliseconds) to wait for the connection to succeed |
| connectdelay | none | How long in (milliseconds) to wait before attempting to reconnect |

Brokerlist failover option

```
failover='<method>[?<options>]'
```

This option controls how failover occurs when presented with a list of brokers. There are only two methods currently implemented but interface `qpid.jms.failover.FailoverMethod` can be used for defining further methods.

Currently implemented failover methods.

| Method | Description |
|--------------|---|
| singlebroker | This will only use the first broker in the list. |
| roundrobin | This method tries each broker in turn. |
| nofailover | [New in 0.5] This method disables all retry and failover logic. |

The current defaults are naturally to use the 'singlebroker' when only one broker is present and the 'roundrobin' method with multiple brokers. The "method" value in the URL may also be any valid class on the classpath that implements the `FailoverMethod` interface.

The 'nofailover' method is useful if you are using a 3rd party tool such as Mule that has its own reconnection strategy that you wish to use.

Options

| Option | Default | Description |
|------------|---------|---|
| cyclecount | 1 | The number of times to loop through the list of available brokers before failure. |

Note: Default was changed from 0 to 1 in Release 0.5

Sample URLs

```
amqp:///test?brokerlist='localhost'
amqp:///test?brokerlist='tcp://anotherhost:5684?retries='10''
amqp://guest:guest@test?brokerlist='vm://:1;vm://:2'&failover='roundrobin'
amqp://guest:guest@test?brokerlist='vm://:1;vm://:2'&failover='roundrobin?cyclecount='20''
amqp://guest:guest@client/test?brokerlist='tcp://localhost;tcp://redundant-server:5673?ssl='true''&fa
```

Url Format Proposal

New URL format for AMQP + Qpid

The Qpid M4 Java and C++ clients use different URL formats.

Java uses: <http://cwiki.apache.org/qpid/connection-url-format.html>

C++ uses the AMQP 0-10 format: section 9.1.2

The AMQP 0-10 format only provides protocol address information for a (list of) brokers. The Qpid M4 Java format provides additional options for connection options (user, password, vhost etc.)

This is a proposal for the AMQP working group to extend the 0-10 URL format to have the flexibility of the Qpid format.

Proposed URL syntax

This proposal extends the AMQP 0-10 URL syntax to include user:pass@ style authentication information, virtual host and extensible name/value options. It also makes the implied extension points of the original grammar more explicit.

Note: terms not defined below (userinfo, host, port, pchar, scheme) are taken from

```
amqp_url      = "amqp://" [ userinfo "@" ] addr_list [ vhost ]
addr_list     = addr *( "," addr )
addr          = prot_addr [ options ]
prot_addr     = tcp_prot_addr | other_prot_addr
vhost         = "/" *pchar [ options ]

tcp_prot_addr = tcp_id tcp_addr
tcp_id        = "tcp:" / "" ; tcp is the default
tcp_addr       = [ host [ ":" port ] ]

other_prot_addr= other_prot_id ":" *pchar
other_prot_id  = scheme

options        = "?" option *( ";" option )
option         = name "=" value
name           = *pchar
value          = *pchar
```

Rationale

Multiple addresses vs. single address.

The URL provides a standard way to group multiple addresses that belong to "the same broker". As well as being used in the AMQP protocol handshake, a URL is a convenient format to "bootstrap" the initial client-broker connection. It can be easily stored in a file, registered in a directory service passed as an argument to a program etc.

A clustered broker can advertise addresses of multiple nodes for fault tolerant connection by clients.

A broker that is connected to multiple networks or via multiple protocols can advertise addresses for each protocol.

Clients can select an address randomly, round robin or based on preference for a particular network or protocol.

Multiple addresses in a single URL should only be used for a set addresses belonging to "the same broker", with equivalent service at each address.

Options

Options can be set for the entire connection or for a specific protocol addresses, providing a flexible way to extend the URL syntax. The AMQP standard will define some options, implementations may define proprietary options. Implementations should ignore any option they do not recognize.

TODO: standard options.

Reserved characters in option names and values must be percent-encoded as per .

Userinfo

Putting security credentials in a URL is generally bad practice in a deployed system but useful in development and testing, so supported here.

The meaning of the userpass information depends on the security mechanism in use. Eg. for SASL authentication with the PLAIN mechanism, the userinfo would be username:password.

TODO need to define:

- protocol address formats and options for other transports - ssl/tls, infiniband, vm...
- standard options for values in the standard connection negotiation.
- qpid proprietary options (e.g. JMS clientid?)

Incompatibility with AMQP 0-10 format

This syntax is backward compatible with AMQP 0-10 with one exception: AMQP 0-10 did not have an initial // after amqp: The justification was that that the // form is only used for URIs with hierarchical structure as per

However it's been pointed out that in fact the URL does already specify a 1-level hierarchy of address / vhost. In the future the hierarchy could be extended to address objects within a vhost such as queues, exchanges etc. So this proposal adopts amqp:// syntax.

It's easy to write a backward-compatible parser by relaxing the grammar as follows:

```
amqp_url = "amqp:" [ "//" ] [ userinfo "@" ] addr_list [ vhost ]
```

Examples

```
# TCP connection to host1 on port 1234, virtual host "vhost" passing username "foo",
# password "bar" for authentication, and specifying a Qpid JMS client-id.
amqp://foo:bar@host1:1234/vhost?clientid=baz

# TCP connection to standard port 5672 on localhost
amqp://localhost

# Choice of infiniband or tcp connections to host foo on different ports.
# TODO: merits of ib vs. rdma as a prefix?
amqp://ib:foo:1234,tcp:foo:5678/

# Connect to the first of host1,host2,host3 that succeeds, retry each twice.
# retry property at connection level is applied to all addresses.
amqp://host1,host2,host3/?retry=2

# Connect to the first of host1, host2, host3 that succeeds, retry host2 twice.
amqp://host1,host2?retry=2,host3

TODO: retry example above is a bit forced - better example for address options vs. url options.

# Connect to vhost over imaginary protocol grok: which has a bunch of optional parameters
# parameters to connect.
amqp://grok:hostname?flavour=strawberry;frobnciation=on/vhost

# Connect using SSL (taken from qpid) see discussion below.
amqp://host?ssl=true
```

Protocols and security

tcp: is the only protocol in AMQP 0-10. Possible additional protocols: sctp, ib (infiniband), ssl, tls.

There are 3 possible ways to indicate a secure connection:

1. Change the URL scheme: amqps://host # Change the protocol: amqp://ssl:host # Use a property: amqp://host?ssl=true

1. the problem here is AMQP supports non-TCP protocols. It's not so clear what amqps://ib:host means.

2. and 3. are more flexible. We can easily add a new protocol identifiers or options for secure variants of any protocol. We can't easily use an open-ended set of URL schemes.

TODO: what's the best thing here?

Differences from Qpid Java format

Addresses are at the start of the URL rather than in the "brokerlist" option.

Option format is ?foo=bar;x=y rather than ?foo='bar'&x='y'. The use of "" quotes is not common for URI query strings, see . The use of "&" as a separator creates problems, see

user, pass and clientid are options rather than having a special place at the front of the URL. clientid is a Qpid proprietary property and user/pass are not relevant in all authentication schemes.

Qpid M4 Java URLs requires the brokerlist option, so this is an easy way to detect a Qpid M4 URL vs. a URL as defined here and parse accordingly.

This proposal only covers connection URLs. A fully-qualified binding URL could be represented as amqp://<addrs>/<vhost>/<binding_url>

Qpid Java Broker Management CLI

How to build Apache Qpid CLI

Build Instructions - General

At the very beginning please build Apache Qpid by refering this installation guide from here
<http://cwiki.apache.org/qpid/qpid-java-build-how-to.html>.

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.

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.

<http://code.google.com/p/lahirugsoc2008/downloads/list>

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 !

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.

Compiling

To compile the source please run following command

```
ant compile
```

To compile the test source run the following command

```
ant compile-tests
```

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 and 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.

Other 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

For now we are supporting those ant targets.

Qpid Design - Access Control Lists

Overview

The AMQ Protocol specification has not yet formally specified how access control lists should be specified or implemented as a result this is subject to change

The Java Qpid Broker provides an authentication framework based on SASL, that provides the ability to plug in arbitrary user (or more strictly *principal*) databases and different SASL-compliant mechanisms.

SASL/Authentication Design

[Qpid Interoperability Documentation](#) : For details on the SASL mechanismsm.

[Qpid Design - PrincipalDatabase] : The Interface for adding new authentication sources

[Qpid Design - Dynamic SASL Mechanisms#server] : How SASL mechanisms are incorporated in the Java broker

[Qpid Design - Dynamic SASL Mechanisms#client] : How AMQPLAIN other Qpid specific SASL mechanisms are added to the Java Client.

ACL Plugin Design Details

[java ACLPlugin]

Continuing work on this design can be found [here](#)

ACL Formats

The Qpid project has two ACL implementations. An initial version of ACLs was added to the Java Broker for M2.1 that uses XML configuration. For M4 a new format was designed to be implemented by both C++ and Java brokers. M4 release includes the initial C++ implementation and M5 is expected to include the Java implementation.

Specifications

The specifications for each of the ACL formats are linked here:

[v1 XML ACLs \(Java Broker Only\)](#)

[v2 All brokers](#)

User Guides

To aid users in defining their ACLs we have a user guide for each of the ACL formats.

[v1 XML ACLs \(Java Broker Only\)](#)

[v2 All brokers](#)

Qpid Design - Authentication

Overview

The AMQ Protocol specifies that SASL is used to exchange authentication information. However, SASL on its own is only a small part of any authentication solution.

QPID provides an authentication framework based on SASL, that provides the ability to plug in arbitrary user (or more strictly *principal*) databases and different SASL-compliant mechanisms. This section describes how to configure both the client and broker and how to add new providers.

It is strongly recommended that any developer who needs to write a security provider or understand this in depth reads the [SASL Guide](#) that is included with the JDK documentation.

Principal Databases

NEEDS REVISION

NOTE : This documentation is outdated as of M2.

Classes

Ignoring the exchange of credentials, a key thing that needs to be done is validate that the credentials are valid. The simple example is checking the username and password match those in a password file.

The interface `org.apache.qpid.server.security.auth.PrincipalDatabase` must be implemented by any "user database". It contains a single method, which takes the `Principal` and the `javax.security.auth.callback.PasswordCallback` that wants to receive the password. This might seem odd, since the obvious approach would be a store that took a principal and some credentials and returned a true or false indicating whether the credentials are valid. However, this approach is required to fit in with the SASL APIs which use callbacks exclusively.

The only implementation currently provided is `PasswordFilePrincipalDatabase`. This expects to read password from a file which is in the format `username:password` where the password is in plaintext and a carriage return separates each username and password pair.

Configuration

Clearly different databases will potentially require different configuration options. For example the `PasswordFilePrincipalDatabase` needs to be configured with the location of a password file whereas a Kerberos realm database would need to know the location of a keytab file (for example).

Configuration is specified in the configuration file like this:

config.xml

```
<security>
  <principal-databases>
    <principal-database>
      <name>passwordfile</name>
      <class>org.apache.qpid.server.security.auth.PasswordFilePrincipalDatabase</class>
      <attributes>
        <attribute>
          <name>passwordFile</name>
          <value>broker/etc/passwd</value>
        </attribute>
      </attributes>
    </principal-database>
  </principal-databases>
</security>
```

After instantiating the class specified by the `class` element, for each attribute an attempt is made to invoke a setter method with value passed in as an argument. In the above example, the method with signature `void setPasswordFile(String arg)` is invoked.

The name element is significant and must be unique. The name is passed as an argument to the SASL provider implementations (described below).

Authentication Providers

The authentication process as described in the AMQ protocol specification is as follows:

1. Broker sends list of authentication mechanisms to the client, in order of preference
2. Client responds with chosen mechanism plus any initial response
3. Broker evaluates response and determines whether authentication has succeeded. If authentication is not yet complete, another set of challenge/responses takes place until authentication is completed (with either success or failure).

The broker configuration allows the administrator to configure the set of supported authentication mechanisms and the principal database used by each mechanism. It also allows the dynamic registration of additional SASL providers (this avoids the need to modify the JRE configuration).

AuthenticationProviderInitialiser

The `org.apache.qpid.server.security.auth.sasl.AuthenticationProviderInitialiser` interface must be implemented for each mechanism. Note this includes the SASL mechanisms that are supported by default by the JRE (e.g. CRAM-MD5). In particular, the interface allows arbitrary configuration (since each provider may have its own specialised configuration requirements), the specification of a callback handler and a factory class for JCA registration.

Configuration

The following example section from the configuration file illustrates how it might be used:

config.xml

```
<sasl>
  <mechanisms>
    <mechanism>
      <initialiser>
        <class>org.apache.qpid.server.security.auth.CRAMMD5Initialiser</class>
        <principal-database>passwordfile</principal-database>
      </initialiser>
    </mechanism>
    <mechanism>
      <initialiser>
        <class>org.apache.qpid.server.security.auth.amqplain.AmqPlainInitialiser</class>
        <principal-database>passwordfile</principal-database>
      </initialiser>
    </mechanism>
  </mechanisms>
</sasl>
```

The above section defines two authentication mechanisms: CRAM-MD5 and AMQPLAIN. Since CRAM-MD5 appears first it will appear on the list of mechanisms offered to the client first. Both mechanisms are configured to use the principal database named "passwordfile" which must be defined in the appropriate section in the config file.

CallbackHandlers

Each SASL provider works with CallbackHandlers. This enables the provider to obtain information (e.g. the password) from the application as well as set the result of authentication (such as the canonical name of the principal just authenticated). The particular CallbackHandlers vary

depending on the mechanism being used. The method `getCallbackHandler` in the `AuthenticationProviderInitialiser` class must return the appropriate handler for the mechanism.

AuthenticationProviderInitialiser Configuration

Since the initialisation of an authentication provider will require potentially arbitrary configuration, the interface supplies to the initialiser the `Configuration` object as well as the base path into the configuration file so that the initialiser can look for arbitrary configuration elements.

Client Authentication

Client authentication is similar although it is made simpler by the fact that there is no need for a `PrincipalDatabase`. Since we do not use any heavyweight configuration mechanism (such as the Apache Commons Configuration used by the broker) on the client side, we also need an alternative way to configure providers.

CallbackHandlerRegistry

The most important things needed on the client are:

1. a way to specify which SASL mechanisms we want to use, in order of preference
2. a way to associate a callback handler with a mechanism (remember that different mechanisms require or support different callbacks)

The `CallbackHandlerRegistry` provides a way to do this. Using a properties file to provide the configuration information, it maps from mechanisms to `AMQCallbackHandler` instances.

`AMQCallbackHandler` extends `javax.security.auth.CallbackHandler` and only adds one method, which associates a protocol session with a callback handler. This enables the callback handler to retrieve any specific information from the `AMQProtocolSession` and by extension from the `javax.jms.Connection`. The obvious example is the username and password.

To specify callback handlers a property file is used. The default property contains this:

```
CallbackHandler.CRAM-MD5=org.apache.qpid.client.security.UsernamePasswordCallbackHandler
CallbackHandler.AMQPLAIN=org.apache.qpid.client.security.UsernamePasswordCallbackHandler
```

The default callbackhandler registry initialises handlers for CRAM-MD5 and AMQPLAIN. To specify your own propertyfile use the java system property `amq.callbackhandler.properties`.

DynamicSaslRegistrar

For SASL providers that are part of the JRE or have been added manually to the JRE security configuration all that is required is the creation of an `AMQCallbackHandler`. However, if you need to use a custom SASL provider it needs to be registered with JCA. Rather than place the burden for doing this on the application developer, we provide a `DynamicSaslRegistrar` which does this.

To configure Sasl providers, you need to create a properties file in this format:

```
AMQPLAIN=org.apache.qpid.client.security.amqplain.AmqPlainSaslClientFactory
```

The key is the mechanism name and the value is the Sasl client factory (which must implement `javax.security.sasl.SaslClientFactory`).

The default properties file registers only the AMQPLAIN mechanism; you can specify a custom property file using the system property `amq.dynamicsaslregistrar.properties`.

Qpid Design - Configuration

Configuration Methods

QPID supports two methods of configuration:

1. command line switches (e.g. passing a `-p` flag on startup to specify the port)
2. configuration file

It is intended that the configuration file will be used for nearly all configuration but that some very common or useful options are exposed using command line switches.

CLI

QPID uses Commons [CLI](#) to parse command line arguments. It provides the following features:

- Ability to parse both short and long flags (e.g. `-p` and `--port`) and treat them as the same logical option
- Generation of well formatted usage messages
- Ability to specify configuration options in different ways, such as from files or from system properties, which can help when writing unit tests

The result of parsing options, however they are specified, is a `CommandLine` object which can then be queried to find out specific values. Currently this is done in `org.apache.qpid.server.Main` and the `CommandLine` object is not exposed elsewhere but if it does require to be more widely used it could be added to the `ApplicationRegistry`. However it is strongly recommended that the configuration approach in the follow

section is used where possible.

Configuration File

QPID uses [Commons Configuration](#) to handle all configuration. It provides methods that allow parsing of options from a range of sources, including configuration files, system properties or simply hard coded classes of values (which is very useful in unit test cases).

Broker configuration is accessed through the class in the org.apache.qpid.server.configuration, primarily starting at ServerConfiguration and retrieving values or other Configuration classes from there.

Command Line Options

The following options are available:

| 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 |

Logging

Logging is handled slightly differently. The main reason for this is that logging is something we want configured before the main configuration file is processed.

The broker uses log4j as the logging implementation, and configuration must be done using the more expressive XML format. A couple of command line switches are used to configure logging:

- -l, --logconfig specifies the log configuration file to use. By default it looks for a file called log4j.xml located in the same directory as the config.xml file
- -w, --logwatch the interval in seconds to poll the log configuration file for changes. The default is 60 seconds and zero means do not poll for changes.

By using the logwatch option it is possible to make changes to the logging configuration at runtime without restarting the broker. (For example, enabling more logging on certain packages in order to diagnose a problem).

QpidBrokerCommandLineOptions

Command Line Options

The following options are available:

| 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 |

Prefetch

AMQP supports prefetch which specifies how many messages the client wishes to cache for delivery. The method Session.setDefaultPrefetch(int) allows a default value to be configured at the session level and extra variants of the createConsumer() method exist allowing it to be specified at the point of consumer construction. It can also be specified as part of the [Connection URL](#). It can be configured with the **IMMEDIATE_PREFETCH** System property

By default, the number of messages which will be filled by the broker on any particular client is 5000.

It is important to consider the implications of prefetch if you wish to [Use Priority Queues](#).

Qpid Meetup at ApacheCon 2009

Qpid Meetup at ApacheCon, Tuesday @ 8 PM

There will be a free Qpid Meetup at [ApacheCon 2009](#) on Tuesday night, 3 November 2009, at 8 PM. This event is open to anyone who wants to come, even if you are not registered for the conference.

Attendees

If you plan to attend, please let us know by signing up [here](#). We're using a form so that people without editing rights on this Wiki can sign up.

- Jonathan Robie (committer)
- David Ingham
- Esteve Fernandez (committer)
- (list is not complete)

Program

If you have something you would like to present, please let us know by signing up [here](#).

The program will be finalized at the beginning of the session. Presentations should be no longer than 15 minutes.

- Apache Qpid Overview - Jonathan Robie
- Qpid on Windows - David Ingham
- (list is not complete)

Suggested topics:

- A talk on any Qpid feature
- Programming Qpid using the Java JMS API, C++ API, or Python API
- Requirements for messaging in some system - we can try to brainstorm a design for the system on the fly
- Comparisons of Qpid to other messaging systems
- Qpid implementation

Qpid Release Page

Qpid Release Page

General Information

This page contains details about the Qpid release process.

- [QpidReleaseProcess](#)
- Release guidelines for Incubator projects
 - http://incubator.apache.org/incubation/Incubation_Policy.html
 - <http://incubator.apache.org/guides/releasemanagement.html>
 - <http://incubator.apache.org/guides/sites.html>

Qpid Release Pages

These pages capture the requirements/feature list for each release.

- 0.6 Release
- M2 Release
- M1 Release
- RC Multi-Platform Testing
- M4 Release Process Notes

0.6 Release

Supported Components and Platforms:

Note that the official Qpid release is a source release. The entire source tree will be released. In addition the source for various individual Qpid components will also be released separately.

The released source will build and have been tested under the following platforms.

- C++ Broker
 - Linux
 - Fedora 11 (gcc 4.4, boost 1.37)
 - Windows XP, Vista (32 and 64 bit) (Visual Studio 2008, Boost 1.35, 1.41)
- Java Broker
 - ??
- C++ Client
- JMS Client (Java)
- Python client
- Ruby client
- WCF client (.NET)

There may also be binary packages for various platforms available for download, but they are not the primary Qpid release and they are provided entirely for user convenience.

Checklist of items for the Release:

- Ensure licensing files are correct
- Release notes/Readmes
- Ensure svn revision targeted for release builds succeed (or are reported to succeed) for the necessary platforms. Including the build time tests.
- Tag candidate release
- Run build script to build and sign release artifacts
- Upload to people.apache.org
- Move to release area.
- Announce

Release Notes:

Please add release note items here:

- Greatly improved cluster stability and performance.
- Persistent cluster automatically restarts from clean database
- C++ broker on Windows now has a persistence module; requires Microsoft SQL Express or higher.

Jiras to be closed:

Below are all the Blocker and critical Qpid Jiras that are targeted for 0.6.

Most of these jira's look like they just need to be reviewed. I assume this means that really they just have to be closed at this stage.

| JIRA Issues (1 issues) | | | | | | | | | | | |
|------------------------|-----------|--|----------------|----------------|----------|-----------------|------------|--------------|--------------|-----|--|
| Type | Key | Summary | Assignee | Reporter | Priority | Status | Resolution | Created | Updated | Due | |
| | QPID-2096 | ExchangeRegistration shoud NOT automatically add durable Exchanges to messageStore | Robbie Gemmell | Martin Ritchie | | Ready To Review | Unresolved | Sep 11, 2009 | Jan 04, 2010 | | |

Below are all other unresolved JIRAs tagged Fix for 0.6 which we should either close or move to 'Affects 0.6' and clear the fix for value.

| JIRA Issues (30 issues) | | | | | | | | | |
|-------------------------|-----------|--|----------------------|----------------------|----------|-----------------|------------|--------------|--------|
| Type | Key | Summary | Assignee | Reporter | Priority | Status | Resolution | Create | Update |
| | QPID-2543 | Change uint to standard type | Unassigned | Bruno Matos | | Open | Unresolved | Apr 23, 2010 | |
| | QPID-2449 | bindings are not removed from the persistent store when a durable queue bound to a durable exchange is deleted | Robbie Gemmell | Robbie Gemmell | | Ready To Review | Unresolved | Mar 16 2010 | |
| | QPID-2292 | the Message class is missing some properties, e.g. the redelivered flag | Rafael H. Schloeming | Rafael H. Schloeming | | Open | Unresolved | Dec 17 2009 | |
| | QPID-2285 | doc generation picks up installed package instead of source package | Rafael H. Schloeming | Rafael H. Schloeming | | Open | Unresolved | Dec 16 2009 | |
| | QPID-2275 | Time based roll over with DatePattern can result in log loss. | Martin Ritchie | Martin Ritchie | | Ready To Review | Unresolved | Dec 15 2009 | |
| | QPID-2274 | Async compression will result in log file deletion when used in conjunction with staticFileName | Martin Ritchie | Martin Ritchie | | Ready To Review | Unresolved | Dec 15 2009 | |
| | QPID-2257 | Tests for Transactional WCF channel | Unassigned | Devang Gandhi | | Open | Unresolved | Dec 09 2009 | |

| | | | | | | | |
|-----------|---|--------------------|----------------|--|-----------------|------------|--------------|
| QPID-2250 | copying messages to a new queue only works with a persistent message and a durable queue | Robbie Gemmell | Robbie Gemmell | | Ready To Review | Unresolved | Dec 07 2009 |
| QPID-2242 | JMS_QPID_DESTTYPE is not set making getJMSDestination unusable. | Martin Ritchie | Martin Ritchie | | Ready To Review | Unresolved | Dec 04 2009 |
| QPID-2232 | rename the JMX Management Console release artifacts | Robbie Gemmell | Robbie Gemmell | | Ready To Review | Unresolved | Dec 03 2009 |
| QPID-2209 | FailedDequeueException whilst clearing queue of messages moved/copied from another queue | Robbie Gemmell | Robbie Gemmell | | Ready To Review | Unresolved | Nov 17 2009 |
| QPID-2196 | JMX Management Console may try to auto-refresh between receiving notification of JMXConnector failure/closure and the server view actually being closed | Robbie Gemmell | Robbie Gemmell | | Ready To Review | Unresolved | Nov 10 2009 |
| QPID-2195 | disable moving messages using JMX Management Console for older brokers with known defects | Robbie Gemmell | Robbie Gemmell | | Ready To Review | Unresolved | Nov 10 2009 |
| QPID-2193 | Allow deleting the first message on a queue through the JMX Management Console when connected to older brokers | Robbie Gemmell | Robbie Gemmell | | Ready To Review | Unresolved | Nov 10 2009 |
| QPID-2190 | enable the updated jmx management console to connect to very old brokers | Robbie Gemmell | Robbie Gemmell | | Ready To Review | Unresolved | Nov 06 2009 |
| QPID-2189 | only admin level users can complete connection to 2.5.0.0 or below (when configured to use <security-enabled> / JMXMP) | Robbie Gemmell | Robbie Gemmell | | Ready To Review | Unresolved | Nov 06 2009 |
| QPID-2178 | Allow viewing of channel flow control status via JMX Management Console | Robbie Gemmell | Robbie Gemmell | | Ready To Review | Unresolved | Oct 30, 2009 |
| QPID-2177 | Allow for configuration of producer flow control queue properties through Management Console | Robbie Gemmell | Robbie Gemmell | | Ready To Review | Unresolved | Oct 30, 2009 |
| QPID-2155 | QpidRollingFileAppender has no tests and does not appear to work. | Martin Ritchie | Martin Ritchie | | Ready To Review | Unresolved | Oct 20, 2009 |
| QPID-2152 | qPID JMX Management Console does not connect to qpid java broker on Windows. Server Connection Failed. Reason: An unknown error has occurred. | Robbie Gemmell | Ignacio Ybarra | | Ready To Review | Unresolved | Oct 17, 2009 |
| QPID-2059 | SubscriptionLoggingTest fails intermittently | Martin Ritchie | Aidan Skinner | | In Progress | Unresolved | Aug 19 2009 |
| QPID-2024 | Add MINANetworkDriver | Rob Godfrey | Aidan Skinner | | Ready To Review | Unresolved | Aug 04 2009 |
| QPID-2019 | Exclude lists are unable to exclude packages | Rafael H. Schlomig | Martin Ritchie | | Ready To Review | Unresolved | Aug 03 2009 |
| QPID-2011 | AlertingTest.testAlertingReallyWorksWithChanges does not test desired functionality. | Martin Ritchie | Martin Ritchie | | Ready To Review | Unresolved | Jul 29, 2009 |
| QPID-1992 | [Logging] Create Operational logging framework | Martin Ritchie | Martin Ritchie | | Open | Unresolved | Jul 17, 2009 |
| QPID-1978 | New list views in MC should allow multiple selection | Martin Ritchie | Martin Ritchie | | Ready To Review | Unresolved | Jul 09, 2009 |
| QPID-1907 | [Java Broker] Improve INFO and above broker log messages to make them more useful in a production environment | Martin Ritchie | Rob Godfrey | | Ready To Review | Unresolved | Jun 17, 2009 |
| QPID-1802 | [Java broker] failure to startup when recovering persistent messages from store for a queue in the configuration file which was not previously added to the store | Robbie Gemmell | Robbie Gemmell | | Ready To Review | Unresolved | Apr 10, 2009 |
| QPID-1753 | Create QMan / WsDmAdapter package | Robbie Gemmell | Martin Ritchie | | Ready To Review | Unresolved | Mar 17 2009 |

| | | | | | | |
|-----------|--|-------------|-------------|-----------|------------|-------------|
| QPID-1440 | [Java Client] - Tidy up tasks from QPID-1289 | Rob Godfrey | Rob Godfrey | To Review | Unresolved | Nov 07 2008 |
|-----------|--|-------------|-------------|-----------|------------|-------------|

M1 Release

M1 Release - COMPLETED 14th Dec 2006

Target Date - November 2006 (tbc)

Release Manager - Rajith Attapattu

Release Notes

You can view our release notes for M1 at:

<https://issues.apache.org/jira/secure/ReleaseNote.jspa?version=12312115&styleName=Html&projectId=12310520>

Release Check list

[M1 Release Check list](#)

Supported Feature List

M1 Release Check list

| Item Description | Status |
|---|---|
| Check license for dependencies | Complete MM 10 Nov, required licenses added |
| Create NOTICE.txt and LICENSE.txt in the root folder | Complete QPID-74, QPID-79 |
| Add all attributions to NOTICE.txt | Complete MM 10 Nov, required attributions added |
| Task to create source distribution | Complete QPID-74 ant target 'std-src-release' |
| Task to create binary distribution | Complete QPID-73 ant target 'std-bin-release' |
| Proper naming of release artifacts | Complete QPID-77 |
| Check if all release jars containing a LICENSE.txt and NOTICE.txt | Complete QPID-78 |
| Proper versioning (adding the correct version numbers to the jar/src files) | QPID-73 & QPID-74 ? |
| Check if Subversion tag is created for release | Rajith to action please |
| Release Notes | Complete QPID-79 |
| README doc in the root folder | Complete QPID-79 |
| Basic documentation | Complete and ref'd in release notes and readme |
| Keys ready for signing the release | Rajith/Gordon to action please QPID-82 pending |
| Adding minimum JDK version for the client/common (1.4) and broker (1.5) | Complete QPID-83 |

M2 Release

M2 Release - Overview

The Qpid project voted to make an M2 release (see this [thread](#)) which will include:

Java Broker (with a caveat on the clustered broker code)
 Java Client
 C++ Broker
 C++ Client
 .NET Client
 Python Client
 Ruby Client

This will be a significant step up from the M1 Release, which only included the Java components.

M2 Release Tasks

| Task | Status | Owner | Notes |
|--|-------------|------------------|--|
| Identify Release Manager | Complete | Marnie McCormack | Rajith Attapattu volunteered |
| Identify JIRAs for inclusion in M2 | In Progress | Marnie McCormack | Tidying up M2 tasks to leave only open tasks for release + showstoppers |
| Define minimal interop requirements for M2 | Complete | Rupert Smith | Interop test suites now exist in Java/C++.Net. Covers fairly minimal use-cases of basic p2p and pub/sub messaging. |
| Identify License Issues | Not Started | ? | Aware that there are probably license tidy ups for Java and need to review license list for C++ etc |
| c++ broker and client owner | In progress | Rajith | Alan volunteered to handle the c++ broker and client release |
| Python and Ruby client owner | In progress | Rajith | Rafi volunteered to handle the Ruby and Python client release |
| java broker and client owner | In progress | Rajith | Martin volunteered to handle the Java broker and client release |

C++ Client/Broker Action - Items for M2 release

[C++ JIRAs for M2](#)

Python & Ruby Clients - Action Items for M2 release

Java Client/Broker - Action Items for M2 release

[Java JIRAs for M2](#)

M4 Release Process Notes

1. Release notes generated from Jira are incomplete/inaccurate
2. Release note text files are included in the artifacts, so can only contain things which are known in advanced
3. Releases shouldn't be scheduled right before the holiday break
4. Trunk was closed for a long time
5. Was difficult to know what the status of the release was due to Jira inaccuracies
6. Lack of clear documentation about release artifacts
7. Lack of interest in fixing problems with some artifacts
8. Many java commits unreviewed: jiras randomly set to resolved, sat in in-progress for ages etc.
9. Addition of GPL library as dependency
10. branching and tagging conventions are inconsistent across releases
11. windows build files require manual updates was a problem
12. We should be time boxed or scope bound
13. Test profiles needs to be ok against defined set before RC spun (inc TCK)
14. System testing & smoke testing (cross-platform) needs to be defined and signed up too/off before RC vote
15. Criteria should be defined for RC sign off thus making the signing off more useful for the RM
16. JIRAs for a release should be scoped in as work starts on the task, scoped out as they get dropped

Qpid Release Notes

[Qpid Java M1 Release Notes](#)

Qpid Java M1 Release Notes

This is the list of JIRA tasks completed for M1

Bugs

- QPID-4 - Remove '/' and ':' from generated queue names
- QPID-7 - Occasionally messages are ack'd more than once
- QPID-10- Broker throughput falls off with transactions
- QPID-56 - AMQQueueMBean - MessageCount on the management interface is not correct.
- QPID-58 - Creating a QueueReceiver results in ClassCastException
- QPID-66 - AMQSession implementation of TopicSession and QueueSession interfaces not JMS compliant
- QPID-68 - Ant build system fails if the project path contains a space
- QPID-69 - Race condition in Delivery Manager

Improvements

- QPID-36 - Add high and low watermark to flow control
- QPID-44 - Add high and low watermark to flow control
- QPID-57 - AMQQueueMBean - Message header attributes should be sent along with message content.

New Features

- QPID-13 - Add option to include prefix and suffix in log file name for broker
- QPID-23 - Extend JNDI support provided to include initial context factory
- QPID-29 - Provide support for using Qpid JMX with Tivoli for application monitoring
- QPID-30 - Allow configuration of working/log directories written to by broker
- QPID-40 - Implement tx.select, tx.commit & tx.rollback from AMQP

Tasks

- QPID-18 - Update Java client and broker to MINA 1.0 release
- QPID-73 - Create Build Artifacts for release process using ant/maven
- QPID-74 - Create source distribution using build system
- QPID-75 - Create Standard Binary distribution using build system

QpidReleaseProcess

Qpid Release Process - Background

Qpid Pre Release Steps

1. Create a wiki page and start capturing the feature/bug fix list for the release
2. We can start a discussion thread and then come to a consensus on the final list
3. These items should be tracked by jira or other means (jira is preferred)
4. We can start a parallel thread on the release dates.

Detailed Qpid Release Process

1. We should do a code freeze and put out a release candidate (ex RC1)
2. We should allow a minimum of one week for people to test/check out the RC
3. If there are major issues maybe do a RC2 and follow the same process
4. If a majority is happy then we can do a code freeze and cut out a release.
5. We should provide a 1.4 [retrotranslator](#) build of the java client (only) with each release we make

Qpid Release Process - Our Process Definition

Introduction

This document describes the general release policies used by the Apache Qpid Project to create releases of the Qpid components. As described herein, this policy is not set in stone and may be adjusted by the Release Manager. We'd like to credit the Apache HTTPD project as our heavily borrowed from template for this document - thanks !

Who can make a release?

Technically, any one can make a release of the source code due to the Apache Software License. However, only members of the Apache Qpid Project (committers) can make a release designated with Apache. Other people must call their release something other than "Apache" unless they obtain written permission from the Apache Software Foundation.

Following our official release policies, we will only accept release binaries from members of the Apache Qpid Project for inclusion on our website. This ensures that our binaries can be supported by members of the project. Other people are free to make binaries, but we will not post them on our website.

Who is in charge of a release?

The release is coordinated by the Release Manager (hereafter, abbreviated as RM). Since this job requires coordination of the development community (and access to SVN), only committers to the project can be RM. However, there is no set RM. Any committer may perform a release at any time. In order to facilitate communication, it is deemed nice to alert the community with your planned release schedule before executing the release. A release should only be made when there is a plan to publicly release it. (A release should not be made only for private distribution. A private release is more suitable for that.)

Who may make a good candidate for an RM?

Someone with lots of time to kill. Being an RM is a very important job in our community because it takes a fair amount of time to produce a stable release.

If you feel lucky, a release could be distributed without testing, but our experience has shown that this leads to a higher number of dud releases. In general, our experience has shown that a well-coordinated release fares better than non-coordinated releases. For Qpid, we are yet to establish a quality bar for releases but it's on our list of things to do !

When do I know if it is a good time to release?

In the case of Qpid, we have identified (thus far) release cutoffs when we know that a) our codebase is fairly stable, b) we have made substantial improvements or additions since any previous release and c) we have significant change coming which would preclude a release for a significant period. M2 will be our second release from Qpid so this process is evolving.

What power does the RM yield?

Regarding what makes it into a release, the RM is the unquestioned authority. No one can contest what makes it into the release. The community will judge the release's quality after it has been issued, but the community can not force the RM to include a feature that they feel uncomfortable adding. Remember that

this document is only a guideline to the community and future RMs - each RM may run a release in a different way. If you don't like what an RM is doing, start preparing for your own competing release. Note that for Qpid we do tend to take votes for such items and follow the consensus.

How does an impending release affect development?

It can not. Let's repeat that: an impending release can not affect development of the project. It is the RM's responsibility to identify what changes should make it into the release. The RM may have an intermediate tag, so the RM can merge in or reject changes as they are committed to the repository's HEAD. For Qpid, we manage our releases using svn branches.

Committers may voluntarily refrain from committing patches if they wish to ease the burden on the RM, but they are under no obligation to do so. This is one reason why we recommend that the RMs have plenty of time on their hands - they may have to deal with a rapidly changing target. It's not an easy job.

How can an RM be confident in a release?

The RM may perform sanity checks on release candidates and should always ensure that (at least) the Qpid brokers being released can be started and connected to using test clients, before distributing. All maven available test classes should be run before releasing a distribution. The release candidate should pass all of the relevant tests before making it official. Note that we are currently in the process of defining a simple set of interop tests which ensure that our client/broker combinations can talk to one another.

How to do a release?

Once the tree has been suitably tested by the RM and any other interested parties, they should "roll" the release.

Key points:

Ensure that the RM's PGP/GPG key ?? - **How do we do the key bit for Qpid ??**

Create an official tag based on the candidate tree

Run the tools to build the appropriate binaries/source dists - **details tbc**

Copy the generated release tarballs and signatures to - ?

Email qpid-dev mailing list to inform them of the release

What can I call this release?

At this point, the release is an alpha. The Qpid Project has three classifications for its releases:

Should we adopt this ??

Alpha

Beta

General Availability (GA)

Alpha indicates that the release is not meant for mainstream usage or may have serious problems that prohibits its use. When a release is initially created, it automatically becomes alpha quality.

Beta indicates that at least three committers have voted positively for beta status and there were more positive than negative votes for beta designation.

This indicates that it is expected to compile and perform basic tasks. However, there may be problems with this release that prohibit its widespread adoption.

General Availability (GA) indicates that at least three committers have voted positively for GA status and that there were more positive than negative votes for GA designation. This release is recommended for production usage.

Who can vote?

Non-committers may cast a vote for a release's quality. In fact, this is extremely encouraged as it provides much-needed feedback to the community about the release's quality. However, only binding votes casted by committers count towards the designation.

Note that no one may veto a release. Releases may not receive a designation level if a problem is found that inhibits proper functionality. The group may (implicitly or explicitly) revoke all votes on a release if there is a problem. However, if there is a -1 vote for a particular designation and there are greater than 3 positive votes and more positive than negative votes (i.e. majority consensus), the appropriate designation is conferred upon the release.

How do we make it public?

Once the release has reached the highest-available designation (as deemed by the RM), the release can be moved to the Qpid distribution directory on apache.org. Approximately 24 to 48 hours after the files have been moved, a public announcement can be made. We wait this period so that the mirrors can

receive the new release before the announcement. An email can then be sent to the announcements lists (announce@apache.org, announce@httpd.apache.org). Drafts of the announcement are usually posted on the development list before sending the announcement to

let the community clarify any issues that we feel should be addressed in the announcement.

Should the announcement wait for binaries?

In short, no. The only files that are required for a public release are the source tarballs (.tar.Z, .tar.gz). Volunteers can provide the Win32 source distribution and binaries, and other esoteric binaries.

Note that the typical Win32 source distribution differs from the original tarball in that it has generated project files as well as the CRLF line endings required for that platform.

Oops. We found a problem.

At this point, the release has been created. No code changes can be made in this release. If a problem is found, it will have to be addressed in the next release or a patch can be made available. No changes can be made between alpha, beta, and GA status. The only difference is the file name that is downloaded

by the users. If an alpha tarball is created, but there was an error that can be resolved by re-rolling the tarball, it may be permissible to re-roll the release. But, the code itself may not change from designation to designation.

There are two courses of action:

Revoke the release and immediately create another one that has a fix to this problem. You can take the old release, apply the single patch, and start the voting process again. This is only recommended for critical problems found early on in the release cycle.

If the problem is less severe, place the patch to the problem in the Qpid patches directory **TBC**. A link to this directory should be included in the release notes with descriptions as to what problem each patch addresses.

Suggestions?

As always, if you have any suggestions or comments on our process, please feel free to email our developer mailing list (qpid-dev@incubator.apache.com) with your comments. We hope you found this document useful.

M2.1 Release process

This is what I (aidan) ran to make the Qpid release artifacts:

```
aidan@contemplation:~/hacking/qpid$ mkdir RC5-artifacts
aidan@contemplation:~/hacking/qpid$ svn co https://svn.apache.org/repos/asf/incubator/qpid/tags/M2.1/RC5/ qpid-M2.1-RC5
aidan@contemplation:~/hacking/qpid$ ln -s qpid-M2.1-RC5 qpid-1.0-incubating-M2.1
aidan@contemplation:~/hacking/qpid$ tar -hzcf RC5-artifacts/qpid-1.0-incubating-M2.1.tar.gz --exclude=.svn qpid-1.0-incubating-M2.1
aidan@contemplation:~/hacking/qpid$ rm qpid-1.0-incubating-M2.1
aidan@contemplation:~/hacking/qpid$ tar -zxf RC5-artifacts/qpid-1.0-incubating-M2.1.tar.gz
aidan@contemplation:~/hacking/qpid$ cd qpid-1.0-incubating-M2.1/cpp
aidan@contemplation:~/hacking/qpid/qpid-1.0-incubating-M2.1/cpp$ ./bootstrap && ./configure && make dist
aidan@contemplation:~/hacking/qpid/qpid-1.0-incubating-M2.1/cpp$ cp qpidc-M2.1.tar.gz ../../RC5-artifacts/
aidan@contemplation:~/hacking/qpid/qpid-1.0-incubating-M2.1/cpp$ cd ../java
aidan@contemplation:~/hacking/qpid/qpid-1.0-incubating-M2.1/java$ mvn -Pfastinstall && cd distribution/ && mvn assembly:assembly
aidan@contemplation:~/hacking/qpid/qpid-1.0-incubating-M2.1/java/distribution$ cp target/.gz target/.zip ../../RC5-artifacts/
aidan@contemplation:~/hacking/qpid/qpid-1.0-incubating-M2.1/java/distribution$ cd ../../dotnet/
aidan@contemplation:~/hacking/qpid/qpid-1.0-incubating-M2.1/dotnet$ sh ./build-framing && ./release mono-2.0 aidan@contemplation:
/hacking/qpid/qpid-1.0-incubating-M2.1/dotnet$ cp bin/mono-2.0/release/Qpid.NET-mono-2.0-2008414.zip ../../RC5-artifacts/
aidan@contemplation:~/hacking/qpid/qpid-1.0-incubating-M2.1/dotnet$ cd ../../
aidan@contemplation:~/hacking/qpid$ tar -zcf RC5-artifacts/qpid-1.0-incubating-M2.1/python-src.tar.gz qpid-1.0-incubating-M2.1/LICENSE
aidan@contemplation:~/hacking/qpid$ tar -zcf RC5-artifacts/qpid-1.0-incubating-M2.1/python-qpid-1.0-incubating-M2.1/specs
aidan@contemplation:~/hacking/qpid$ tar -zcf RC5-artifacts/qpid-1.0-incubating-M2.1/ruby-src.tar.gz qpid-1.0-incubating-M2.1/LICENSE
aidan@contemplation:~/hacking/qpid$ tar -zcf RC5-artifacts/qpid-1.0-incubating-M2.1/ruby-qpid-1.0-incubating-M2.1/specs
aidan@contemplation:~/hacking/qpid$ cd qpid-1.0-incubating-M2.1/java/
aidan@contemplation:~/hacking/qpid/qpid-1.0-incubating-M2.1/java$ mvn deploy -DaltDeploymentRepository=incubator::default::
file:///home/aidan/hacking/qpid/RC5-artifacts/maven -Pfastinstall
aidan@contemplation:~/hacking/qpid/qpid-1.0-incubating-M2.1/java$ cd ../../RC5-artifacts/
aidan@contemplation:~/hacking/qpid/RC5-artifacts$ rm java-src-console-unix
aidan@contemplation:~/hacking/qpid/RC5-artifacts$ sha1sum *.zip *.gz > SHA1SUM
aidan@contemplation:~/hacking/qpid/RC5-artifacts$ for i in `find . | egrep 'jar$|pom$|gz$|zip$|SHA1SUM'`; do gpg --sign --armor --detach $i; done;
```

and then rsync it up people.apache.org

0.5 Release process

The above steps performed by Aidan have been bundled up and provided as a release script.

Located in *trunk/qpid/bin* the *release.sh* script should make the RM's job easier.

```

$ ./release.sh --help
Usage: release.sh <svn-path> <svn-revision> <version> [options]

Options: Default : --prepare -all --sign
--help | -h : Show this help
--prepare : Download specified tree from svn
--clean-all : Remove build artefacts and downloaded svn tree
--clean : Remove built artefacts
--all | -a : Generate all artefacts
--source| -e : Generate the source artefact
--cpp | -c : Generate the CPP artefacts
--dotnet| -d : Generate the dotnet artefacts
--java | -j : Generate the java artefacts
--ruby | -r : Generate the ruby artefacts
--python| -p : Generate the python artefacts
--source| -e : Generate the source artefact
--sign | -s : Sign generated artefacts
--upload| -u : Upload the artifacts directory to people.apache.org as qpid-$VER

```

Copyright © 1999-2007, The Apache Software Foundation

RC Multi-Platform Testing

Any QPID release candidate should be tested on the following supported platforms:

- Linux 2.6+
- Windows XP 5+
- Solaris 8 (SunOS 5.8)+
- Cygwin v?

It shall be a quality gateway for any RC that it must run successfully (define tests here ?) on all platforms supported. All scripts will run successfully on these platforms (*.sh on all unix/linux style OSs and *.bat on Windows).

Multi-platform operability is a key attribute for QPID users and thus we must ensure that changes applied do not compromise this.

Qpid Ruby Documentation

Qpid supplies a ruby client for making AMQP connections to a compliant broker, require qpid.rb to get everything you need. tests/basic.rb has a simple example.

Qpid Testing

Testing Pages

- Interop Testing Specification
- Java Unit Tests with InVM Broker
- Performance, Reliability and Scaling
- Qpid JMX Management Console Testing Guide
- Testing Design - Java Broker CPU GC Monitoring

Interop Testing Specification

Qpid Interop Testing Spec. Working Copy.

| | | | |
|----------------------|---------------|---------------|---|
| Draft. | Rupert Smith. | 22nd Feb 2007 | Document started. |
| Working Copy. | Rupert Smith. | 6th Mar 2007 | Document updated from feedback to draft on qpid-dev list. Last requirement # used: 47 |
| Working Copy. | Rupert Smith. | 7th Mar 2007 | Senders and receivers to send reports to coordinator. Reply-to added to broadcast messages. Last requirement # used: 49 |
| Working Copy. | Rupert Smith. | 13th Mar 2007 | Added test case names. Last requirement # used: 52 |

| | | | |
|------------------------------------|---------------|----------------|--|
| Working Copy. | Rupert Smith. | 25th Sept 2007 | Added test cases for message size variation. Last requirement # used: 60 |
| Version 2 Work in Progress. | Rupert Smith | 22nd Nov 2007 | Test framework being expanded to cover functional and performance tests and a much wider variety of testing possibilities. |

Introduction:

The requirements in this specification use a common format, an example of which is given below:

| | |
|----------------------------------|---|
| RE-1. Sample Requirement. | A brief description of the requirement. |
|----------------------------------|---|

The requirements are numbered from 1.

Purpose:

- Test sending from and receiving by each of the clients in Qpid over both of the broker implementations.
- Enable testing of any JMS compliant product, by keeping a pure JMS sub-set of the testing framework separate. This only applies to Java messaging client implementations.
- Provide a parameter driven test framework, that can be used to generate many testing scenarios for different messaging modes.
- Allow functional testing of messaging at the product surface, through the standard interfaces/protocols (JMS, AMQP), so that the same test suite may be applied over different implementations.
- Allow tests to be posed in terms of abstract asynchronous messaging concepts that AMQP and JMS support, rather than at the level of direct interfaces. This allows the same tests to carry forward as standards and products evolve.
- Enable interoperability testing between any AMQP compliant components, not just those in Qpid.
- Allow performance testing to be carried out across a distributed set of edge nodes connected to a messaging broker.
- Make tests robust enough to run as part of an automated build. The scripts should pass or fail, not hang, wait forever, run out of memory or otherwise cause an automated build process to fail to complete.
- Be capable of running the full test suite on several machines in a hands free way. In particular C++ tests need to run on unix and .Net on windows, necessitating a multi-box solution for full interop testing.
- Be capable of running the same test cases across message topologies ranging from a single test node running in the same process as a broker, to many test nodes running on different machines, remotely connected to a broker.

Constraints:

| | | |
|---------------|----------------------------|---|
| IOP-1. | Operating System. | The test client scripts must run on Unix and Windows. If a test client implementation is only available on one of these platforms it only needs to run on its supported platform. |
| IOP-2. | Scripting Language. | Each test client must be startable from a Unix shell script. Tests run on Windows will use Cygwin to run these scripts. There is no need to support Windows .bat scripts. |

Functional Requirements:

Introduction.

These requirements describe the behaviour of test clients for testing between different client implementations of AMQP. Each client is expected to be a single program that is capable of sending test messages to other clients and receiving and responding to test messages received from other test clients. The clients are not to be run as separate programs for the sending and receiving parts for the sake of convenience in being able to run the clients as part of an automated build. The clients will listen for control messages broadcast by a master coordinator, to enlist them in tests, tell them which test to run, when to begin their tests, where to submit reports about the tests and when to shut down.

A centralized approach has been chosen, using a single coordinator, as test framework code which would otherwise have to be duplicated amongst all the clients will generally be put in the coordinator. The idea is to place as much logic as possible in the coordinator and as little as possible in the clients which means that code will only have to be written and maintained in one place. This code will include code for enlisting clients for tests, deciding which test case to run, and formatting and logging out the results. The alternative would be to have a de-centralized approach, where each client broadcasts the test enlist messages, finds out what other clients are available to talk to, chooses which tests to run and outputs the test results. One advantage of the centralized approach, is that the coordinator should know which clients are available, and therefore which clients cannot run particular tests, or fail completely to run particular tests, and should therefore be able to log out failures for clients that fail tests in a more reliable way, than if it were up to the clients to log their own failures and omissions.

Build tests out of a standardized construction block.

- Diagram: The test circuit.

Publisher/Receiver pair.

Each end of which is a Producer/Consumer unit.
M producers, N consumers, talking over Z destinations.

One of the stated aims of this specification is to "Allow tests to be posed in terms of abstract asynchronous messaging concepts that AMQP and JMS support, rather than at the level of direct interfaces". For example, we know that messages sent in a transaction, must not be delivered until the transaction is committed. This is true of AMQP as it is of JMS; as AMQP is intended to provide similar messaging semantics to JMS. The statement is also true, whether the messages are broadcast to many receivers or sent to just one.

The standard construction block for a test, is a test circuit. This consists of a publisher, and a receiver. The publisher and receiver may reside on the same machine, or may be distributed. Will use a standard set of properties to define the desired circuit topology.

Tests are always to be controlled from the publishing side only. The receiving end of the circuit is to be exposed to the test code through an interface, that abstracts as much as possible the receiving end of the test. The interface exposes a set of 'assertions' that may be applied to the receiving end of the test circuit.

In the case where the receiving end of the circuit resides on the same JVM, the assertions will call the receivers code locally. Where the receiving end is distributed across one or more machines, the assertions will be applied to a test report gathered from all of the receivers. Test code will be written to the assertions making as few assumptions as possible about the exact test topology.

A test circuit defines a test topology, M producers, N consumers, Z outgoing routes between them.

The publishing end of each test circuit always resides on a single JVM, even if M > 1. If publishers are to be distributed across many machines, the test framework itself provides the scaling by running the same test circuit many times in parallel. This means that it is possible to have an arbitrary number of message publishers across one or many machines, determined by the test setup.

The receiving half of the circuit may be local, in which case all messages come back to the same machine, or distributed in which case they may be received by many machines.

There are therefore two ways in which tests may be distributed across multiple nodes in a network; many test circuits may be distributed and run in parallel and/or the receiving ends of those circuits may be distributed or local.

Each node in the network can play up to 2 roles in any given test; publisher or receiver. It is possible to play both roles at once, but would like to have a 'single_role' flag, that can be set to ensure that test nodes taking one role, will not participate in the other for the duration of a test. For example, in the pub/sub test want one publisher and the remaining nodes to distribute the receiver role amongst themselves.

Probing for the available test topology.

- Diagram: The available topology.

When the test distribution framework starts up, it should broadcast an 'enlist' request on a known topic. All available nodes in the network to reply in order to make it known that they are available to carry out tests. For the requested test case, C test circuits are to be run in parallel. Each test defines its desired M by N topology for each circuit. The entire network may be available to run both roles, or the test case may have specified a limit on the number of publishing nodes and set the 'single_role' flag. If the number of publishing nodes exhausts the available network and the single role flag is on, then there are no nodes available to run the receiver roles, the test will fail with an error at this point. Suppose there are P nodes available to run the publisher roles, and R nodes available to run the receiver roles. The C test circuits will be divided up as evenly as possible amongst the P nodes. The C * N receivers will be divided up as evenly as possible amongst the R nodes.

A more concrete example. There are 10 test machines available. Want to run a pub/sub test with 2 publishers, publishing to 50 topics, with 250 subscribers, measuring total throughput. The distribution framework probes to find the ten machines. The test parameters specify a concurrency level of 2 circuits, limited to 2 nodes, with the single role flag set, which leaves 8 nodes to play the receiver role. The test parameters specify each circuit as having 25 topics, unique to the circuit, and 125 receivers. The total of 250 receivers are distributed amongst the 8 available nodes, 31 each, except for two of them which get 32. The test specifies a duration of 10 minutes, sending messages 500 bytes in size using test batches of 10000 messages, as fast as possible. The distribution framework sends a start signal to each of the publishers. The publishers run for 10000 messages. The publishers request a report from each receiver on their circuit. The receivers send back to the publishers a report on the number of messages received in the batch. The publishers assert that the correct number for the batch were indeed received, and log a time sample for the batch. This continues for 10 minutes. At the end of the 10 minutes, the publishers collate all of their timings, failures, errors into a log message. The distribution framework requests the test report from each publishing nodes, and these logs are combined together to produce a single log for the entire run. Some stats, such as total time taken, total messages through the system, total throughput are calculated and added as a summary to the log, along with a record of the requested and actual topology used to run the test.

- Diagram: The requested test applied onto the available topology.

Test Procedures.

A variety of different tests can be written against a standard test circuit, many of these will follow a common pattern. One of the aims of using a common test circuit configured by a number of test parameters, is to be able to automate the generation of all possible test cases that can be produced from the circuit combined with the common testing pattern, and an outline of a procedure for doing this is described here. The typical test sequence is described below:

A typical test sequence.

1. Initialize the test circuit from the default parameters, plus specific settings for the test.
2. Create the test circuit. The requested test parameters are applied to the available topology to produce a live circuit.
3. Send messages.
4. Request a status report.
5. Assert conditions on the publishing end of the circuit.
6. Assert conditions on the receiving end of the circuit.
7. Pass or fail the test.

The thorough test procedure.

The thorough test procedure uses the typical test sequence described above, but generates all of combinations of test parameters and corresponding assertions against the results.

The all_combinations function produces all combinations of test parameters described in Appendix A.

all_combinations : List<Properties>

The expected_results function, produces a list of assertions, given a set of test parameters. For example, mandatory && no_route -> assertions.add(producer.assertMessageReturned), assertions.add(receiver.assertMessageNotReceived).

expected_results: Properties -> List<Assertions>

For parameters : all_combinations
test_circuit = new TestCircuit(parameters).
test_circuit.start.

Send messages.
Request status.

For assertion : expected_results(parameters)
Assert(assertion).

Common Requirements.

| | | |
|----------------|---|--|
| IOP-3. | Directory Structure. | All scripts to start and stop brokers and run test clients will be placed in a directory structure underneath a top-level directory called 'interop' that sits at the top level of the Qpid project. |
| IOP-4. | Test Output Format. | Output in junit xml format (because a lot of automated build software understands this format). There doesn't seem to be a schema or DTD for this format but it is simple enough. See Appendix B for an example. |
| IOP-5. | Terminate On Timeout. | Each client will keep a timeout count. Every time it gets a message it will reset this count. If it does not hear from the broker at all for 60 seconds then it will assume that the broker has died or that the other test clients are failing to communicate with it, and will terminate. Test clients will only wait on this timeout when they are actually expecting messages, for example after enlisting to a test and expecting a role assignment message, or during a test when they are expecting to be sent a test message. If necessary, this timeout can be extended to a longer time period than 60 seconds, its purpose is to ensure eventual termination of all clients during a fully automated build. |
| IOP-6. | Default Virtual Host. | All test clients will use the default virtual host (no name) for all tests, unless overridden by test parameters for a particular test case, or by command line options when starting the client. |
| IOP-7. | Broadcast Control Topic. | All test clients will listen to control messages broadcast on the routing key 'iop.control' on the default virtual host on the default topic exchange. This control topic is used for communicating with the test coordinator client. |
| IOP-48. | Control Message Replies. | All control messages broadcast by the coordinator will include a reply to field. The coordinator will listen on the reply address for responses to its control messages. |
| IOP-8. | No Environment for Scripts. | In general, start up scripts should be intelligent enough to configure the environment variables that they need in order to run. It should be sufficient to have a path configured for the necessary run time tools (such as Java) when calling scripts. Environment variables, such as QPID_HOME, should be set by startup scripts themselves, figured out from their installation locations. |
| IOP-9. | Wait Until Background Process Started. | Scripts that start processes running in the background should not terminate until the process they are starting has successfully started. This is necessary for reliable testing, to ensure that subsequent scripts can be run, knowing that previous scripts have completed, with dependant processes in a known state. For example, it is important to start all test clients prior to starting the coordinator. |

Use Case 1. Starting a Broker.

Run the broker start script.

The script starts a broker running and tries to connect to it (or otherwise ping it) until it is verified to be running.
Once the broker is verified to be running the script terminates with no error code.

Failure path: The broker fails to start or does not appear to be running after a timeout has passed. The script fails with an error code.

| | | |
|----------------|----------------------------------|--|
| IOP-10. | Broker Start Script. | The Java and C++ brokers will define scripts that can start the broker running on the local machine, and these scripts will be located at interop/java/broker/start and interop/cpp/broker/start. The Java and C++ build processes will generate these scripts (or copy pre-defined ones to the output location) as part of their build processes. |
| IOP-11. | Broker Start Failure. | If a broker fails to start within 60 seconds its start script will timeout. Script will terminate with error code 1. |
| IOP-12. | Broker Start Successfull. | When the broker starts successfully the script will terminate with error code 0. |

Use Case 2. Stopping a Broker.

Run the broker stop script.

The script terminates the broker that was started with the start script if it is still running.

Failure path: The broker won't terminate. The script fails with an error code.

| | | |
|---------|-------------------------|---|
| IOP-13. | Broker Stop Script. | The Java and C++ brokers will define scripts that can stop the broker running on the local machine, and these scripts will be located at interop/java/broker/stop and interop/cpp/broker/stop. The Java and C++ build processes will generate these scripts (or copy pre-defined ones to the output location) as part of their build processes. |
| IOP-14. | Broker Stop Timeout. | If a broker fails to terminate within 60 seconds its stop script will timeout. Script will terminate with error code 1. |
| IOP-15. | Broker Stop Succesfull. | When the broker stops successfully the script will terminate with error code 0. |

Use Case 3. Starting a Test Client.

Run the client start script. The caller will pass in the address of the broker to connect to.

The script starts a client running.

The client starts running but waits for further instruction before running its tests.

The start script will terminate but leave the client running as a forked process.

Failure path: The client will not start, or fails to connect to the specified broker. The script will terminate with error code 1.

| | | |
|---------|----------------------------------|---|
| IOP-16. | Client Start Scripts. | For each client implementation, <client>, there will be a start script located at interop/<client>/client/start. The build processes for each client will generate these scripts and output them to this location as part of their build process. |
| IOP-17. | Client Start Timeout. | If the client fails to start and connect to the specified broker within 60 seconds the script will terminate with error code 1. |
| IOP-18. | Client Start Succesfull. | When the client starts successfully its script will terminate with error code 0. |
| IOP-19. | Client Start Broker and Port. | The -b <hostname> option will be used to instruct the start script to connect to the specified hostname. The -p <port> option will similarly allow the port to be specified. |
| IOP-20. | Client Virtual Host. | The default virtual host to connect to, may be overridden with the -v <virtual_host> command line option, which will be accepted by all test clients. |
| IOP-21. | Client Start General Parameters. | General parameters may be passed to the client start scripts using the syntax name=value. These name/value pairs may be used by specific test cases to override default test parameters. See Appendix C for a list of test parameters. |

Use Case 4. Starting the Coordinator.

- The requirements defined for Use Case 3, also apply to this use case.

Run the testall start script. The caller will pass in the address of the broker to connect to.

The script starts the coordinator client running.

The coordinator will manage the test procedure.

The script will terminate when the coordinator has completed.

Failure path: The coordinator will not start, or fails to connect to the broker. The script will terminate with error code 1.

| | | |
|---------|--------------------------|--|
| IOP-22. | Coordinator Test Script. | There will be a coordinator test script that kicks off the testing process once all clients have been started. It is to be located at interop/testall. It will start a coordinator test client that issues test invites, assigns roles, collects results and terminates test clients when all tests have been run. |
|---------|--------------------------|--|

Use Case 5. Overall Test Procedure.

Start a broker running using its start script as described by Use Case 1.

Call the start all clients script on each of the machines where there are clients that are to be tested. The caller will pass in address of the broker to connect to, and any additional parameters.

The start all script will scan for all start scripts located under interop/<client>/client/start and call each of them forwarding its command line arguments on the call. This performs Use Case 3 for each client.

Call the coordinator test client script. This is described as Use Case 4.

The coordinator test script will broadcast an invite message, with no test name on the control topic. The lack of a test name indicates that this is a compulsory invite, to which all clients must enlist.

Each client will respond with an enlist message. This message will contain the routing key on the default topic exchange to which the client has bound its private control queue.

The coordinator retains the list of available clients, and the addresses of their control queues.

The coordinator will broadcast an invite to a named test. This invite may also contain any parameters needed to configure the test, that are relevant to a clients choice to accept the invite or not.

All clients that are able to participate in this test will reply to the invite with enlist messages. Clients may opt to participate in the test depending on the test parameters, if desired.

The coordinator will send messages to assign roles to the sender and receivers private control topics. These messages will contain the test parameters and roles. The test parameters may also include additional parameters not in the original invite, for test parameters that are to be set on a per test instance basis.

The clients will respond with accept role messages.

The coordinator will wait until it has received acceptances from both roles.

The coordinator will issue a start message to the client with the sender role.

The sender client will send its test messages. Once the test has completed the sender will send a report message to the coordinator, giving details about the message that it sent.

The coordinator will wait until it receives a report message from the sender.

The coordinator will issue a status request message to the receiver role.

The receiver will reply with a report, giving details about the messages it has received.

The coordinator will wait until it receives a report message from the receiver.

The coordinator will compare the sender and receiver reports in order to decide whether the test passed or failed.

The coordinator will check its list of available clients and log out failures for any combinations of clients that were not tested because they did not enlist for the test.

Once all test cases are complete, the coordinator will broadcast a shutdown message.

All clients will terminate on receipt of the shutdown message.

The coordinator will terminate.

Terminate the broker using its stop script.

| | | |
|---------|--------------------------------------|--|
| IOP-23. | Start All Script. | There will be a start all clients script, located at interop/startall. The startall script finds all client starts scripts under interop/<client>/client/start and calls them. |
| IOP-24. | Start All Script Options Forwarding. | The start all script will take the same command line options as the client start scripts and will pass these command line options on to them. |
| IOP-25. | Invite Message. | For every test case the coordinator will broadcast an invite message on the control topic. This message will be identified by the header field, "CONTROL_TYPE", having the value, "INVITE". This message will also include the name of the test case and may also include some test parameters. (See IOP-48, for the reply to address.)

<div style="border: 1px solid black; padding: 10px;"><pre>"CONTROL_TYPE" , "INVITE" "TEST_NAME" , "<test_case>" ... optional test parameters.</pre></div> |
| IOP-26. | Initial Invite. | At the start of the test procedure the coordinator will broadcast a compulsory invite, to which all available clients must enlist, in order to declare their availability and to enable the coordinator to detect when there are clients that did not participate in some tests. The compulsory invite will be differentiated from an ordinary invite because it will have no "TEST_NAME" header field. |
| IOP-27. | Enlist Message. | Every test client that receives an invite message will respond by declaring its availability to run interop tests. The client will send an enlist message by replying to the invite message. The enlist message will be identified by the header field, "CONTROL_TYPE", having the value, "ENLIST". The client will declare the routing key on which it expects to be sent private control messages. The client will also declare a unique name by which it can be identified (see IOP-35). The declare available message will contain the following header fields with this information:

<div style="border: 1px solid black; padding: 10px;"><pre>"CONTROL_TYPE" , "ENLIST" "CLIENT_NAME" , "<client_name>" (see IOP-35 for rules about the client name). "CLIENT_PRIVATE_CONTROL_KEY" , "iop.control.<client_name>" (see IOP-36)</pre></div> |

| | | |
|---------|---|--|
| IOP-28. | Assign Role Message. | Having selected clients to participate in a particular test case, the coordinator will send those clients messages to assign the roles they will play in the test case, on the clients private control topics. Each test case has sender and receiver roles. This message will be identified by the header field, "CONTROL_TYPE", having the value, "ASSIGN_ROLE". The full test parameters will be included in this message, allowing tests to be configured on a per test instance basis. |
| | | <pre>"CONTROL_TYPE" , "ASSIGN_ROLE" ... full test parameters.</pre> |
| IOP-29. | Accept Role Message. | A client receiving an assign role message, will reply to it with an accept role message. This message also indicates that the client is ready to start the test. This message will be identifier by the header field, "CONTROL_TYPE", having the value, "ACCEPT_ROLE". |
| | | <pre>"CONTROL_TYPE" , "ACCEPT_ROLE"</pre> |
| IOP-30. | Start Message. | The coordinator will send a start message to begin the test procedure. All test clients will listen for this message on their private control topics. The start message will be identified by the header field, "CONTROL_TYPE", having the value, "START". |
| | | <pre>"CONTROL_TYPE" , "START"</pre> |
| IOP-31. | Report Message. | Once the test clients have completed a test case, they will send the coordinator a report about the actions they have performed. In the case of senders, this report will be sent once they have finished sending test messages. In the case of receiver, this report will be sent in response to a status request from the coordinator (see IOP-49). The report message will be identified by the header field, "CONTROL_TYPE", having the value, "REPORT". Its message body, or additional header fields will contain the report, specific to the test case being run. |
| | | <pre>"CONTROL_TYPE" , "REPORT" ... test specific parameters. Message body, Test case specific report.</pre> |
| IOP-49. | Status Request Message. | Once the coordinator has received the senders report, it will send a status request to the receiver, to request the receivers report. This message will be identified by the header field, "CONTROL_TYPE", having the value, "STATUS_REQUEST". |
| | | <pre>"CONTROL_TYPE" , "STATUS_REQUEST"</pre> |
| IOP-34. | Terminate Message. | The coordinator will wait for all test clients to complete their tests for all test cases at which time it will broadcast a terminate message to the control topic. The terminate message will be identified by the header field, "CONTROL_TOPIC", having the value, "TERMINATE". Upon receipt of this message the test clients will terminate. |
| | | <pre>"CONTROL_TYPE" , "TERMINATE"</pre> |
| IOP-35. | Client Name. | Each test client will provide a unique name for itself that reflects its implementation language and distinguishes it from the other clients. Clients may append an environment identifier onto this name to cater for the case where the same client is used multiple times in an interop test. For example, the same client might be run on two different operating systems, in order to check that it works correctly on both. Example names in this case might be "java-win" and "java-linux". |
| IOP-36. | Private Client Control Topic. | Each test client will listen for test control messages directed specifically to it on the default topic exchange. The routing key for these messages will consist of "iop.control." followed by the client name (see IOP-35). A topic exchange is used, rather than a direct exchange, to cater for the situation where multiple instances of a client are run in parallel and tests are to be scaled accross many clients (not currently in scope, see Waiting Room). It also allows a listener to be attached to the default topic exchange to listen to all control messages using a wildcard selector. |
| IOP-37. | Separate Connection for Control Topic. | Test clients should create open a separate connection to communicate with the control topics on the default topic exchange, to that which they use to perform tests. This is so that a channel level error that results in the closing of a connection during a test, may still allow a client to successfully send a failure report to the coordinator. |

Common Requirements for Test Cases.

Test cases that use these requirements mention them in the description of the test case.

| | | |
|---------|--------------------------------------|---|
| IOP-38. | Message Counts. | Whenever a test client receives a message from another test client it will increment the total count of messages received from that client. Test messages will contain the name of the sending client in the header field "CLIENT_NAME", and the count will be held against a combination of that name and the messages correlation id (see IOP-42). |
| IOP-39. | Message Count Reset. | Whenever a test client is beginning a new test case (when it accepts a role) it will reset its message counts to zero. |
| IOP-41. | Message Count Report Message. | Upon receipt of a status request message, a test client will reply with a report message. The report message will be identified by the header field "CONTROL_TYPE", having the value, "REPORT" (as described by IOP-31). In addition to this, the header field, "MESSAGE_COUNT" will contain the count of messages received since the last reset as a signed 32-bit integer.

<div style="border: 1px solid black; padding: 10px; width: fit-content;"> <pre>"CONTROL_TYPE" , "REPORT" "MESSAGE_COUNT" , <count> (signed 32 bit integer)</pre> </div> |
| IOP-42. | Correlation Id. | When sending test messages, clients will identify all messages using a unique correlation id for the test case instance. This will differentiate test messages in a situation where the same client is scaled up to run a test case many times in parallel (not in scope, see Waiting Room). |
| IOP-43. | Test Connections. | Test clients will create connections to send test messages on when they are assigned roles. In many cases this will consist of creating a single connection, and a producer or consumer for the test routing key or queue. In some tests, which simulate the activity of many message receivers, multiple connections may be opened. |

Test Case 1. Dummy Run.

The sending client will not send any test messages at all. It will send a report message on the control topic, declaring that the test has passed.

The purpose of this test case is to check that clients can interoperate successfully with the test coordinator and participate in the sequencing of the tests.

| | | |
|---------|--------------------------|---|
| IOP-50. | Test Case 1 Name. | The "TEST_NAME" field in the test invite (IOP-25) will be "TC1_DummyRun" for this test. |
|---------|--------------------------|---|

Test Case 2. Basic P2P Test.

- This test case uses requirements IOP-38 to 43 inclusive.

The sending client creates a fresh correlation id, and the entire test case conversation uses this id.

The sending client will send the required number of test messages to the test routing key on the default direct exchange.

The sending client will send a message count report to the coordinator.

In response to a status request from the coordinator, the receiving client will reply with a message count report.

The coordinator will compare the messages received to the messages sent and pass or fail the test accordingly.

| | | |
|---------|--|---|
| IOP-44. | Basic P2P Setup. | Prior to assigning roles, the coordinator will bind a queue to the default direct exchange with a routing key, the same as the queue name. It will create a fresh queue and key for every test case instance. |
| IOP-45. | Basic P2P Assign Role Parameters. | In addition to the invite message format defined in IOP-26, the basic p2p test invite will also include the following parameters.

<div style="border: 1px solid black; padding: 10px; width: fit-content;"> <pre>"P2P_QUEUE_AND_KEY_NAME" , "<name>" "P2P_NUM_MESSAGES" , <count> (signed 32 bit int), P2P_NUM_MESSAGES property.</pre> </div> |
| IOP-51. | Test Case 2 Name. | The "TEST_NAME" field in the test invite (IOP-25) will be "TC2_BasicP2P" for this test. |

Test Case 3. Basic Pub/Sub Test.

- This test case uses requirements IOP-38 to 43 inclusive.

The sending client creates a fresh correlation id, and the entire test case conversation uses this id.

The sending client will send the required number of test messages to the test routing key on the default topic exchange.

The sending client will send a message count report to the coordinator.

In response to a status request from the coordinator, the receiving client will reply with a message count report. This number will be the

number of messages sent multiplied by the number of receivers being simulated by the receiving client.
The coordinator will compare the messages received to the messages sent and pass or fail the test accordingly.

| | | |
|----------------|---|--|
| IOP-46. | Basic Pub/Sub Setup. | Prior to assigning roles, the coordinator will choose a routing key for the test. It will create a fresh key for every test case instance. |
| IOP-47. | Basic Pub/Sub Invite Parameters. | In addition to the invite message format defined in IOP-26, the basic pub/sub test invite will also include the following parameters. <pre>"PUBSUB_KEY", "<key>"
 "PUBSUB_NUM_RECEIVERS", <count> (signed 32 bit
int), PUBSUB_NUM_RECEIVERS property.
 "PUBSUB_NUM_MESSAGES", <count> (signed 32 bit
int), PUBSUB_NUM_MESSAGES property.</pre> |
| IOP-52. | Test Case 3 Name. | The "TEST_NAME" field in the test invite (IOP-25) will be "TC3_BasicPubSub" for this test. |

Test Case 4. P2P Test with Different Message Sizes.

- This test case uses requirements IOP-38 to 43 inclusive.

The sending client creates a fresh correlation id, and the entire test case conversation uses this id.

The sending client will send the required number of test messages to the test routing key on the default direct exchange.

The sending client will send a message count report to the coordinator.

In response to a status request from the coordinator, the receiving client will reply with a message count report.

The coordinator will compare the messages received to the messages sent and pass or fail the test accordingly.

The above test cycle will be repeated for each message size to test.

| | | |
|----------------|--|--|
| IOP-53. | P2P Message Size Test Setup. | Prior to assigning roles, the coordinator will bind a queue to the default direct exchange with a routing key, the same as the queue name. It will create a fresh queue and key for every test case instance. |
| IOP-54. | P2P Message Size Test Assign Role Parameters. | In addition to the invite message format defined in IOP-26, the basic p2p test invite will also include the following parameters. <pre>"P2P_QUEUE_AND_KEY_NAME", "<name>"
 "P2P_NUM_MESSAGES", <count> (signed 32 bit
int), P2P_NUM_MESSAGES property.
 "messageSize", <count> (signed 32-bit
int).</pre> |
| IOP-55. | P2P Message Size Test Sizes | The following values for the message size parameter will be tested: 0K, 63K, 64K, 65K, 127K, 128K, 129K, 255K, 256K, 257K. |
| IOP-56. | Test Case 4 Name. | The "TEST_NAME" field in the test invite (IOP-25) will be "TC4_P2PMessageSize" for this test. |

Test Case 5. Pub/Sub Test with Different Message Sizes.

- This test case uses requirements IOP-38 to 43 inclusive.

The sending client creates a fresh correlation id, and the entire test case conversation uses this id.

The sending client will send the required number of test messages to the test routing key on the default topic exchange.

The sending client will send a message count report to the coordinator.

In response to a status request from the coordinator, the receiving client will reply with a message count report. This number will be the number of messages sent multiplied by the number of receivers being simulated by the receiving client.

The coordinator will compare the messages received to the messages sent and pass or fail the test accordingly.

The above test cycle will be repeated for each message size to test.

| | | |
|----------------|---|--|
| IOP-57. | Pub/Sub Message Size Test Setup. | Prior to assigning roles, the coordinator will choose a routing key for the test. It will create a fresh key for every test case instance. |
|----------------|---|--|

| | | |
|---------|---|---|
| IOP-58. | Pub/Sub Message Size Test Invite Parameters. | In addition to the invite message format defined in IOP-26, the basic pub/sub test invite will also include the following parameters.

<pre>"PUBSUB_KEY" , <key> "PUBSUB_NUM_RECEIVERS" , <count> (signed 32 bit int), PUBSUB_NUM_RECEIVERS property. "PUBSUB_NUM_MESSAGES" , <count> (signed 32 bit int), PUBSUB_NUM_MESSAGES property. "messageSize" , <count> (signed 32-bit int).</pre> |
| IOP-59. | P2P Message Size Test Sizes | The following values for the message size parameter will be tested: 0K, 63K, 64K, 65K, 127K, 128K, 129K, 255K, 256K, 257K. |
| IOP-60. | Test Case 5 Name. | The "TEST_NAME" field in the test invite (IOP-25) will be "TC5_PubSubMessageSize" for this test. |

Waiting Room:

Contains ideas for possible future directions relating to this spec.

Command processor. Test cases to be written using a command language (perhaps in XML) on top of a common client API. Interpreter for this to be implemented using each client library. Test cases need only be written once and can be run by the interpreters. Command language rich enough to exercise the whole AMQP protocol. May not handle client specific edge cases. Good for ensuring test consistency, but may take a fair amount of time to do.

How I anticipate this being run as part of a fully automated build. Will try to get a free licence for Anthill Pro 3 as they offer free licences for open source projects. Viewtier Parabuild is another possibility. Anthill Pro runs a central build server that does all its work through build agents that can run on many boxes. It also lets you define build workflows. I imagine running a Unix agent to build the c++, java and python stuff, and a Windows agent for the .net stuff. Will define a workflow that starts a broker on the unix box, then starts all clients built on the unix and windows boxes in parallel, then runs the entire test procedure across all clients, then terminates the broker on the unix box. The agents send back the test results to the central server.

Full testing of field tables. Make sure that every possible data type is tested and confirmed to encode and decode correctly between all client implementations.

Testing more of the protocol. Add tests to more fully exercise the complete AMQP protocol.

Allow scaling of test clients. Each test client should only be run once (in each environment) and they create unique names for themselves. Tests are only run between pairs of single clients, with a single sender and number of receivers defined by the test case (often 1). Clients listen for control messages on topics, and use correlation id's in all tests messages to differentiate themselves, were multiple senders to be active. This has been done deliberately to allow for future expansion of the test framework to allow scaling up of the tests by starting more clients in parallel on the same environment. To do this each client might also create a sequence number, to uniquely identify itself, as the client names will no longer be unique. Reports from senders will include client name, sequence number and correlation id. Status requests to receivers may specify client name, sequence number and correlation id to get specific reports, or omit correlation id or sequence number to get a bulk report of all messages with a particular client.

More sophisticated reporting. Message count reports are fairly minimal. Might also put an entire list of messages send/recieved in a report, in order to check that there were no omissions or duplicates.

Appendix A, General Notes:

Brokers that need to be interop tested: C++ and Java

Clients that need to be interop tested: C++ , Java, Java 1.4 retrotranslation, C++, .Net 2.0, .Net 1.1, (Mono?), Python, Ruby.

Appendix B, Example of XML Format for Test Ouput:

I don't think there is a DTD or schema for this but the XML output from JUnit looks like the example below. This is a convenient choice for the output format from these test results even if the code does not actually use JUnit (or cppunit or nunit) internally, because automated build servers generally understand and are able to produce test reports from it.

Example:

```

<?xml version="1.0" encoding="UTF-8" ?>
<testsuite errors="0" skipped="0" tests="18" time="0.02" failures="0"
name="org.apache.qpid.framing.BasicContentHeaderPropertiesTest">
  <properties>
    <property value="Java(TM) 2 Runtime Environment, Standard Edition" name="java.runtime.name"/>
    ... (there were lots of properties).
  </properties>
  <testcase time="0.02" name="testRejectedExecution"/>
  ... (there were lots of test cases).
</testsuite>

```

Appendix C, Test Parameters.

| | Possible Values | Default Value |
|-------------------------------------|--|-----------------|
| Connection properties. | | |
| broker | tcp, vm | tcp://localhost |
| vhost | | <empty> |
| username | | guest |
| password | | guest |
| Topology properties. | | |
| max_publishing_node | | 1 |
| single_role | true, false | true |
| Circuit properties. | | |
| num_publishers | | 1 |
| num_consumers | | 1 |
| num_destinations | | 1 |
| base_out_route_name | | ping |
| base_in_route_name | | pong |
| bind_out_route | true, false | true |
| bind_in_route | true, false | false |
| consumer_out_active | true, false | true |
| consumer_in_active | true, false | false |
| JMS flags and options. | | |
| transactional | true, false | false |
| persistent | true, false | false |
| no_local | true, false | false |
| ack_mode | tx, auto, client, dups_ok, no_ack, pre_ack | auto |
| AMQP/Qpid flags and options. | | |
| exclusive | true, false | false |
| immediate | true, false | false |
| mandatory | true, false | false |
| durable | true, false | false |
| prefetch_size | | |
| header_fields | | |
| Standard test parameters. | | |
| message_size | no_body, one_body, multi_body | one_body |

| | | |
|------------------|--|------------|
| num_messages | | 100 |
| outgoing_rate | | |
| inbound_rate | | |
| timeout | | 30 seconds |
| tx_batch_size | | 100 |
| max_pending_data | | |

Total combinations over all test parameters: $4 * 48 * 16 * 3 = 9216$ combinations.

Defaults give an in-VM broker, 1:1 P2P topology, no tx, auto ack, no flags, publisher -> receiver route configured, no return route.

Appendix D, Command line options.

IOP-21 states that general parameters can be passed on the command line using name=value syntax. The coordinator understands the following parameters, and will use them to override the default values for the tests. Individual test cases refer to the command line parameter that they take their test parameters from.

| Parameter | Default |
|----------------------|---------|
| P2P_NUM_MESSAGES | 50 |
| PUBSUB_NUM_RECEIVERS | 5 |
| PUBSUB_NUM_MESSAGES | 10 |

Appendix E, Clock Synchronization Algorithm.

On connection/initialization of the framework, synch clocks between all nodes in the available topology. For in vm tests, the clock delta and error will automatically be zero. For throughput measurements, the overall test times will be long enough that the error does not need to be particularly small. For latency measurements, want to get accurate clock synchronization. This should not be too hard to achieve over a quiet local network.

After determining the list of clients available to conduct tests against, the Coordinator synchronizes the clocks of each in turn. The synchronization is done against one client at a time, at a fairly low messaging rate over the Qpid broker. If needed, a more accurate mechanism, using something like NTP over UDP could be used. Ensure the clock synchronization is captured by an interface, to allow better solutions to be added at a later date. Here is a simple algorithm to get started with:

1. Coordinator tells client to synchronize its clock with the coordinators time.
2. Client stamps current local time on a "time request" message and sends to Coordinator.
3. Upon receipt by Coordinator, Coordinator stamps Coordinator-time and returns.
4. Upon receipt by Client, Client subtracts current time from sent time and divides by two to compute latency. It subtracts current time from Coordinator time to determine Client-Coordinator time delta and adds in the half-latency to get the correct clock delta.
5. The first result should immediately be used to update the clock since it will get the local clock into at least the right ballpark.
6. The Client repeats steps 1 through 3, 25 or more times, pausing a few tens of milliseconds each time.
7. The results of the packet receipts are accumulated and sorted in lowest-latency to highest-latency order. The median latency is determined by picking the mid-point sample from this ordered list.
8. All samples above approximately 1 standard-deviation from the median are discarded and the remaining samples are averaged using an arithmetic mean.

The above algorithm includes broker latency, two network hops each way, plus possible effects of buffering/resends on the TCP protocol. A fairly easy improvement on it might be:

1. Coordinator tells client to synchronize its clock with the coordinators time, provides a port/address to synchronize against.
2. Clients sends UDP packets to the Coordinators address and performs the same procedure as outlined above.

Appendix F, Deleted Requirements:

Put deleted requirements here, in case they can be re-used.

| | | |
|-------------|--|---|
| IOP. | Client Start Messages Per Test. | The -m <num_messages> option will be used to tell the client how many messages to send per test. |
| IOP. | Client Number of Receivers. | For topic testing each client will simulate the behaviour of many clients listening to the same topic. The number of receivers per test client for topic tests will be specified by the -r <num_receivers> command line option. |
| IOP. | Client Default P2P Test Direct Key. | Each test client will listen for test messages on the default direct exchange. The routing key for these messages will consist of the client name (see IOP-35) followed by ".direct". |
| IOP. | Client Default Pub/Sub Test Direct Key. | Each test client will listen for test messages on the default topic exchange. The routing key for these messages will consist of the client name (see IOP-35) followed by ".topic". |

| | | |
|------|---------------------------------------|---|
| IOP. | Test Done Message. | Once a test client has completed its role, it will send the coordinator a test done message on the control topic. This message will be identified by the header field, "CONTROL_TYPE", having the value, "TEST_DONE". The client will also post its name in the "CLIENT_NAME" header field. |
| | | "CONTROL_TYPE" ,
"TEST_DONE" |
| IOP. | End Role Message. | Once the coordinator receives a report for a test case, it will send end role messages to the private control topics of all clients participating in the test case. This message will be identified by the header field, "CONTROL_TYPE", having the value, "END_ROLE". |
| | | "CONTROL_TYPE" ,
"END_ROLE" |
| IOP. | Client Status Request Message. | When a test client has completed sending test messages it may request the count of actual messages received from the test client to which it sent the messages. The status request message will be sent to the receiving test clients individual control topic. This message will be identified by the header field, "CONTROL_TYPE", having the value, "STATUS_REQUEST", and will contain the name of the sending client in the header field "CLIENT_NAME". |
| | | "CONTROL_TYPE" ,
"STATUS_REQUEST"
"CLIENT_NAME" ,
"<client_name>" |

Java Unit Tests with InVM Broker

Sample Code

Here is a template for correctly creating a unit test that will use the InVM Broker.

```

@Before
public void initialSetup() throws Exception
{
    createVMBroker();

    //Any other setup code

}

public void createVMBroker()
{
    try
    {
        TransportConnection.createVMBroker(1);
        // Multiple Brokers can be created by passing in a different port. The above is
connected to with the url vm://:1
    }
    catch (AMQVMBrokerCreationException e)
    {
        Assert.fail("Unable to create broker: " + e);
    }
}

@After
public void stopVmBroker()
{
    // If you created a connection in the @Before be sure to close it before you kill the
broker or
    // it will attempt failover, and recreate the broker.
    TransportConnection.killVMBroker(1);
    //Remember to kill any other brokers you create
}

@Test
public void yourTest code() throws Exception
{
    //If you do your connection setup here
    Connection con = new AMQConnection("vm://:1", "guest", "guest", "consumer1", "test");

    // Be sure to close it before you end the test
    con.close();
}

```

Performance, Reliability and Scaling

Qpid has a substantial server performance test suite based around the [junit-toolkit](#) library.

The test cases fall into the following categories:

[Throughput](#) - straight line 0-600,000 messages/second in how long?

[Latency](#) - how quickly can I get a few messages?

[Reliability](#) - burn in, stress and soak tests

Building and running the tests

The performance tests live in `qpid/java/perftests`. When they are built the scripts are generated and placed into `qpid/java/build/bin/perftests`. These are detailed in the relevant list of test cases above. There are 3 convenience scripts which correspond to the categories to run all the throughput, latency or reliability tests.

Test case implementation

All test cases utilise one of a couple of test classes, with a different set of command line parameters.

The following parameters may be passed to the test runner to control its operation:

| Parameter | Meaning |
|-------------------------|--|
| <code>-c pattern</code> | The number of tests to run concurrently. |

| | |
|-------------|--|
| -r num | The number of times to repeat each test. |
| -d duration | The length of time to run the tests for. |
| -t name | The name of the test case to execute. |
| -s pattern | The size parameter to run tests with. |
| -o dir | The name of the directory to output test timings to. |
| --csv | Output test results in CSV format. |
| --xml | Output test results in XML format. |
| -v | Verbose mode. |

Here are some examples:

| | |
|--|--|
| -c [10:20:30:40:50] | Runs the test with 10,20,...,50 threads. |
| -s [1:100]:samples=10 | Runs the test with ten different size parameters evenly spaced between 1 and 100. |
| -s [1:1000000]:samples=10:exp | Runs the test with ten different size parameters exponentially spaced between 1 and 1000000. |
| -r 10 | Runs each test ten times. |
| -d 10H | Runs the test repeatedly for 10 hours. |
| -d 1M, -r 10 | Runs the test repeatedly for 1 minute but only takes a timing sample every 10 test runs. |
| -r 10, -c [1:5:10:50], -s [100:1000:10000] | Runs 12 test cycles (4 concurrency samples * 3 size sample), with 10 repeats each. In total the test will be run 199 times (3 + 15 + 30 + 150) |

The test runner also accepts name=value properties on the end of the command line, and these are passed to the test code as parameters to control the operation of the test. The following properties may be set:

| Parameter | Default | Comments |
|------------------|----------------------|---|
| messageSize | 0 | Message size in bytes. Not including any headers. |
| destinationName | ping | The root name to use to generate destination names to ping. |
| persistent | false | Determines whether persistent delivery is used. |
| transacted | false | Determines whether messages are sent/received in transactions. |
| broker | tcp://localhost:5672 | Determines the broker to connect to. |
| virtualHost | test | Determines the virtual host to send all ping over. |
| rate | 0 | The maximum rate (in hertz) to send messages at. 0 means no limit. |
| verbose | false | The verbose flag for debugging. Prints to console on every message. |
| pubsub | false | Whether to ping topics or queues. Uses p2p by default. |
| failAfterCommit | false | Whether to prompt user to kill broker after a commit batch. |
| failBeforeCommit | false | Whether to prompt user to kill broker before a commit batch. |
| failAfterSend | false | Whether to prompt user to kill broker after a send. |
| failBeforeSend | false | Whether to prompt user to kill broker before a send. |
| failOnce | true | Whether to prompt for failover only once. |
| username | guest | The username to access the broker with. |
| password | guest | The password to access the broker with. |
| selector | null | Not used. Defines a message selector to filter pings with. |
| destinationCount | 1 | The number of destinations to send pings to. |
| numConsumers | 1 | The number of consumers on each destination. |
| timeout | 30000 | In milliseconds. The timeout to stop waiting for replies. |
| commitBatchSize | 1 | The number of messages per transaction in transactional mode. |
| uniqueDests | true | Whether each receivers only listens to one ping destination or all. |

| | | |
|----------------|----------|--|
| durableDest | false | Whether or not durable destinations are used. |
| ackMode | AUTO_ACK | The message acknowledgement mode. Possible values are:

0 - SESSION_TRANSACTED
1 - AUTO_ACKNOWLEDGE
2 - CLIENT_ACKNOWLEDGE
3 - DUPS_OK_ACKNOWLEDGE
257 - NO_ACKNOWLEDGE
258 - PRE_ACKNOWLEDGE |
| consTransacted | false | Whether or not consumers use transactions. Defaults to the same value as the 'transacted' option if not separately defined. |
| consAckMode | AUTO_ACK | The message acknowledgement mode for consumers. Defaults to the same value as 'ackMode' if not separately defined. |
| maxPending | 0 | The maximum size in bytes, of messages sent but not yet received. Limits the volume of messages currently buffered on the client or broker. Can help scale test clients by limiting amount of buffered data to avoid out of memory errors. |

Added for 0.7

| Parameter | Default | Comments |
|--------------------|---------|--|
| numConsumers | <int> | Augmented Allow a value of 0, meaning no consumers. |
| consumeOnly | boolean | Disable all message sending. Message counts are used by consumers as validation. |
| preFill | int | Message count to preFill the destination with before the test start. |
| delayBeforeConsume | int | Delay in ms to wait after the preFill has occurred before the test starts. |

Test case output

Test cases output data in csv format. The extractThroughputResults.sh script can be used to interpret the data and output the average throughput rate.

Latency

- Job Queueing 1:10, medium sized messages
- Low volume, small messages auto-ack
- Medium volume, small messages, no-ack
- Scaling tests

Job Queueing 1:10, medium sized messages

The first set of benchmark tests, TQBL-AA-Qpid-02*.sh and PQBL-AA-Qpid-02*.sh, set up 10 consumers and send 1000 messages of 5120 bytes at a capped rate limit. The final number in the test name is the rate cap, divided by 1000.

Low volume, small messages auto-ack

The second set of benchmark tests, TTBL-AA-Qpid-03* and PTBL-AA-*.sh, set up 50 consumers and send 1000 messages of 256 bytes at a capped rate. The Rate is the final number in the test name, multiplied by 1000. The [PT]TBL-AA-Qpid-04*.sh scripts are identical except that they send messages of 5120 bytes.

Medium volume, small messages, no-ack

The [PT]TBL-NA-Qpid-05*.sh and [PT]TBL-NA-Qpid-05*.sh scripts are identical to the second case above, but use No-Ack and send messages at a rate of the last number in the test name multiplied by 400.

Scaling tests

PQCL-Qpid-01 and PQCL-Qpid-02 set up an increasing number of consumers (between 1 and 30) and send messages of 256 bytes at a rate of 600 (PQCL-Qpid-01) or 100 messages (PQCL-Qpid-02) per second.

PTCL-Qpid-01 (consumed in a transaction) and PTCL-Qpid-02 (consumed with AutoAck) set up an increasing number of consumers

(between 1 and 30) and send messages of 256 bytes at a rate of 1 message per second.

Reliability

- Short running timing tests
- Longer running tests

Short running timing tests

[PT][QT]R-Qpid-01 (Transactions) and [PT][QT]R-Qpid-02 (AutoAck) send messages of 256 bytes for 1 minute to 16 consumers

Longer running tests

[PT][QT]R-Qpid-0[3-8]*.sh send messages of 256 bytes to 16 consumers using the specified Acknowledgement mode for an hour.

Throughput

- Consumer scaling tests
- Message size scaling tests
- Sustained throughput tests for different acknowledgement modes

Consumer scaling tests

[PT][TQ]CT-Qpid-01 (transacted consumer) and [PT][TQ]CT-Qpid-02 (AutoAck consumer) send messages of 256 bytes to destinations which have between 1 and 30 consumers subscribed.

Message size scaling tests

The [PT][TQ]M-Qpid-* .sh and [PT][TQ]M-Qpid-* tests send messages of varying size from 8 producers to 8 consumers. In [PT][TQ]M-Qpid-01, the consumer uses a transaction. In [PT][TQ]M-Qpid-02, it uses autoack. Messages vary in size from 512 bytes to 1MiB.

Sustained throughput tests for different acknowledgement modes

The [PT][TQ]BT-* tests send messages of 256 bytes to 16 consumers using AutoAck, transactions or NoAck (NoAck is not tested for persistent messages) as fast as possible.

Qpid JMX Management Console Testing Guide

The guide can be found below in wiki form, or downloaded as a file: ([DOC](#)) ([PDF](#))

Introduction
General Test Configuration & Startup
Server configuration
Console configuration
SSL configuration
JMXMP configuration
Console Startup
Server Management Connections
ConfigurationManagement MBean
LoggingManagement MBean
ServerInformation MBean
UserManagement MBean
VirtualHostManager MBean
Queue Management
Notifications
Exchange Management
Connection Management

Introduction

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 details procedures and expected outcomes for performing functional testing of the console.

General Test Configuration & Startup

Server configuration

For the purposes of the console testing, the server should initially be configured as detailed below. To assist, example server configuration files are provided (these serve as example and may need updated when testing a different version of server). The can be downloaded along with a message sending utility [here](#)

VirtualHosts: 'localhost', 'development', and 'test'

minimumAlertRepeatGap = 30sec, maximumMessageCount = 89, & maximumMessageAge = 10sec in 'test' VirtualHost

Queues in 'test' VirtualHost, bound to 'amq.direct' Exchange: ping, queue, ping_1

Username & Passwords: admin:admin

Management access rights: admin=admin

Console configuration

When the console is started for the first time on a machine, it creates the file *qpiddmc_navigation.ini* in the *.qpiddmc* subfolder of the current '*home*' directory. This file stores the Qpid server addresses and MBean Favourites which are added to the consoles connection tree, in order that they may be persisted between sessions.

When Queue attributes are selected in the Queues selection screen these are also saved in this folder, in the file *qpiddmc_queue_attributes.ini*. Ideally, these files and the containing folder should be removed before testing begins to ensure this functionality still works fully.

Typical '*home*' directories are C:\Documents and Settings\<userid> for Windows XP, C:\Users\<userid> for Windows Vista + 7, and /home/<userid>/on Linux.

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/jss/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 *qpiddmc.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.

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.

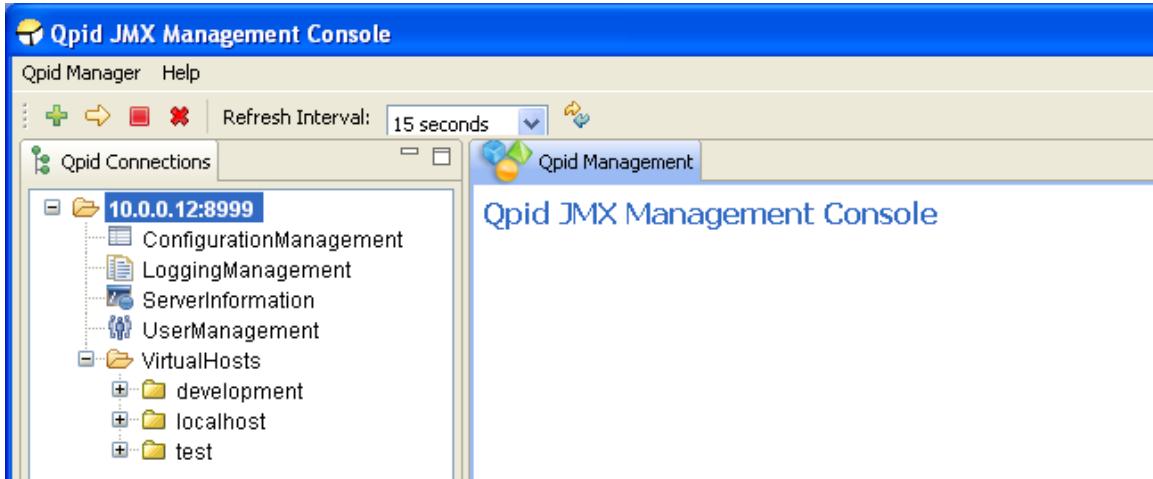
You can download the JMX Remote API 1.0.1_04 Reference Implementation from the Sun website [here](#). The included *jmxremote-1_0_1-bin/lib/jmxremote_optional.jar* file must be added to the *plugins/jmxremote.sasl_1.0.1* folder of the console release (again, in Mac OS X you will need to select 'Show 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.

Console Startup

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

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

Server Management Connections

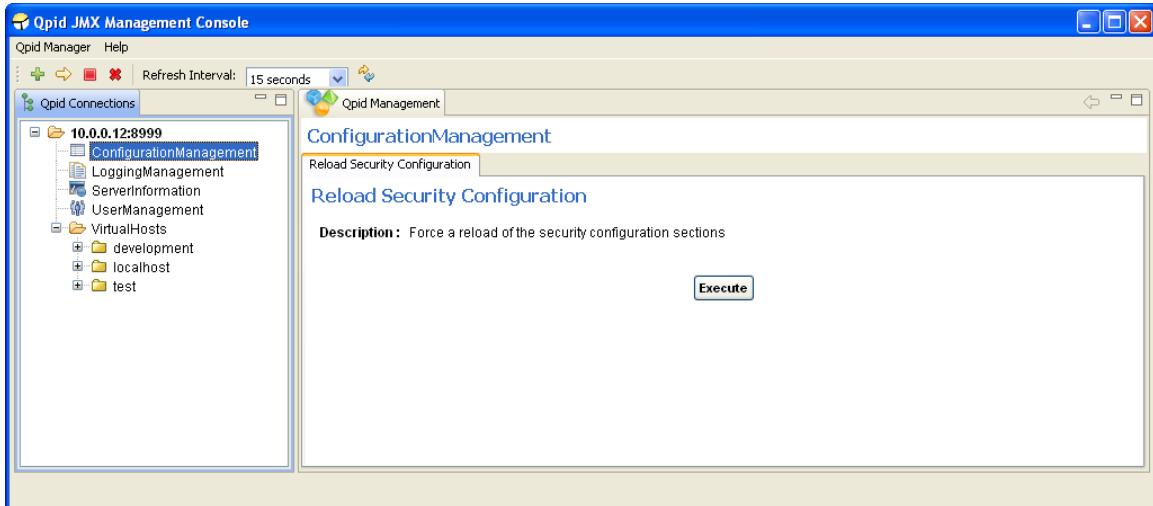


| Test ID | Test Steps | Expected Result |
|-----------|--|---|
| LOGIN-0 | Start the console (described above) | The GUI opens |
| LOGIN-1 | Click the <i>New Connection</i> icon | The <i>New Connection</i> dialog opens |
| LOGIN-1.1 | Enter hostname=<hostname>, port= 8999, Username=admin, Password=admin and then click Connect. | The node <hostname>:8999 will be added in the Qpid Connections pane at the left-hand side of the console window, and expanded to show the <i>ConfigurationManagement</i> , <i>UserManagement</i> , <i>ServerInformation</i> , and <i>LoggingManagement</i> MBeans as well as a VirtualHosts folder containing child folders <i>development</i> , <i>localhost</i> , and <i>test</i> . |
| LOGIN-2 | Select the <hostname>:8999 node in the Qpid Connections tree and then click the Disconnect button in the toolbar | The connection is closed and the <hostname>:8999 tree node collapses to a single entry. |
| LOGIN-3 | Select the <hostname>:8999 node in the Qpid Connections tree and then click the "Reconnect" icon | The <i>Reconnect</i> dialog opens |
| LOGIN-3.1 | Enter Username=admin, Password=admin and then click Connect. | The server node will be expanded to show the <i>ConfigurationManagement</i> , <i>UserManagement</i> , <i>ServerInformation</i> , and <i>LoggingManagement</i> MBeans as well as a VirtualHosts folder containing child folders <i>development</i> , <i>localhost</i> , and <i>test</i> . |

ConfigurationManagement MBean

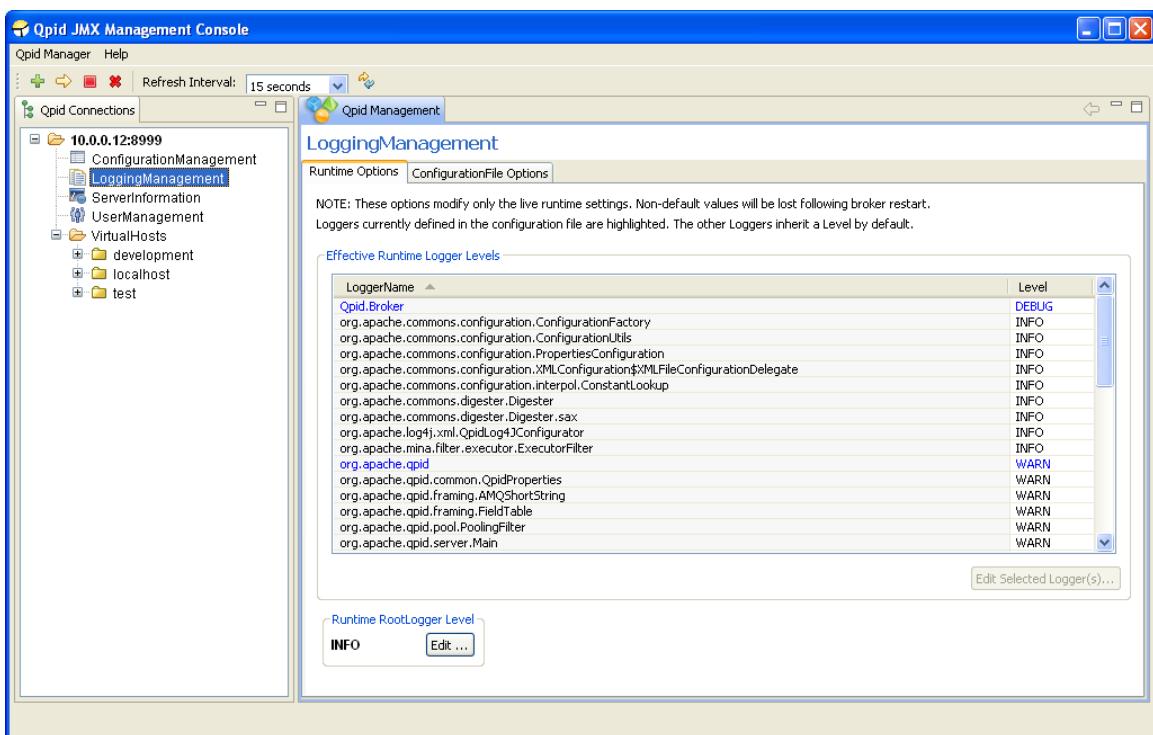
Pre-Requisite: Connect to a server as described in test LOGIN -0 or LOGIN -3 above.

| Test ID | Test Steps | Expected Result |
|----------|---|---|
| CONF-0 | Select the <i>ConfigurationManagement</i> node for the server in the Qpid Connections tree. | The <i>ConfigurationManagement</i> MBean is opened in the MBean view, showing the <i>reloadSecurityConfiguration</i> operation. |
| CONF-1 | Modify the server configuration file, updating a VirtualHost <i>security</i> sub-section, adding a <i>firewall</i> configuration entry to deny AMQP access from a certain IP address. | N/A |
| CONF-1.1 | Press the Execute button and confirm the prompt to carry out the <i>reloadSecurityConfiguration</i> operation | The updated security behaviour is applied and an Operation Successful dialog is shown. |
| CONF-1.2 | Attempt an AMQP connection from the blocked IP address | The connection fails. |



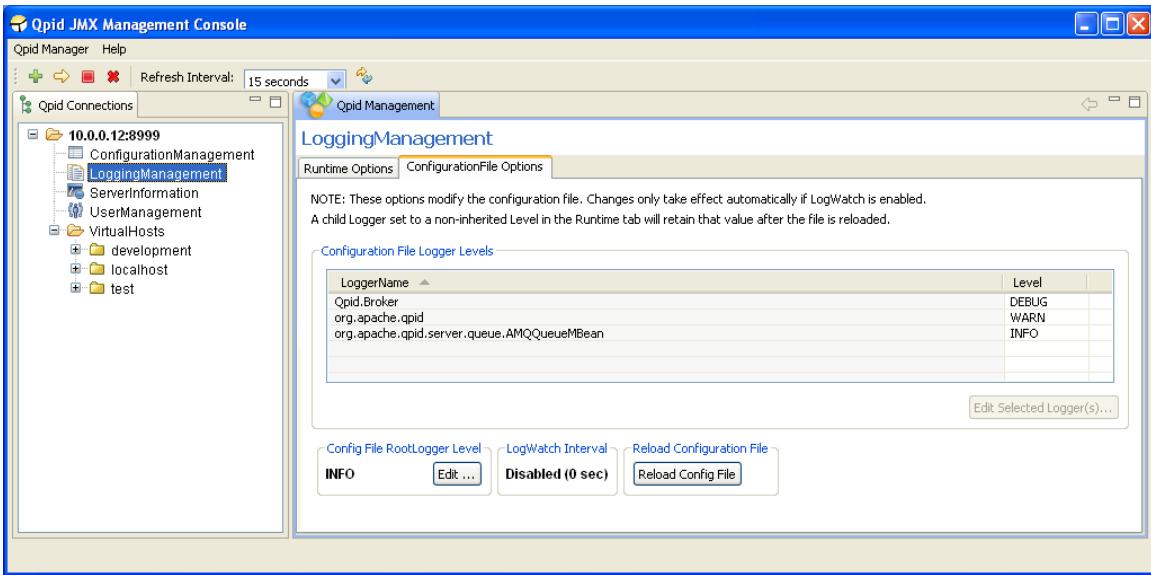
LoggingManagement MBean

Pre-Requisite: Connect to a server as described in test LOGIN -0 or LOGIN -3 above.



| Test ID | Test Steps | Expected Result |
|---------|--|---|
| LOG-0 | Select the <i>LoggingManagement</i> node for the server in the Qpid Connections tree. | The <i>LoggingManagement</i> MBean is opened in the MBean view, showing the <i>Runtime Options</i> tab. |
| LOG-1 | Double click the <i>org.apache.qpid</i> LoggerName | The <i>Set Runtime Logger Level</i> dialog opens. |
| LOG-1.1 | Select a Level of ERROR from the combo box and click ok. | The <i>org.apache.qpid</i> Logger and all those below it beginning with <i>org.apache.qpid</i> (except <i>org.apache.qpid.server.queue.AMQQueueMBean</i>) are now shown at ERROR Level in the table. |
| LOG-2 | Press the <i>Edit</i> button in the <i>Runtime RootLogger Level</i> area. | The <i>Set Runtime RootLogger Level</i> dialog opens. |
| LOG-2.1 | Select a Level of WARN from the combo box and click ok. | The Level is updated to WARN, plus any Logger (except <i>qpid.message</i>) without a highlighted blue Logger as a prefix inherits from the RootLogger and will also change to WARN Level in the table. |
| LOG-3 | Select the <i>org.apache.qpid.server.Main</i> LoggerName and press the <i>Edit Selected Logger</i> button. | The <i>Set Runtime Logger Level</i> dialog opens. |

| | | |
|---------|---|---|
| LOG-3.1 | Select a Level of WARN from the combo box and click ok. | The <i>org.apache.qpid.server.Main</i> Logger is now shown at WARN Level in the table, all others remain unchanged. |
|---------|---|---|



| | | |
|---------|---|--|
| LOG-4 | Select the <i>ConfigurationFile Options</i> tab in the <i>LoggingManagement</i> MBean view. | The <i>ConfigurationFile Options</i> tab opens to show the Loggers defined in the configuration file and their Level |
| LOG-5 | Double click the <i>org.apache.qpid</i> LoggerName | The <i>Set ConfigFile Logger Level</i> dialog opens. |
| LOG-5.1 | Select a Level of INFO from the combo box and click ok. | The <i>org.apache.qpid</i> Logger is now shown at INFO Level in the table |

The following tests are not supported by the original version of the LoggingManagement MBean.

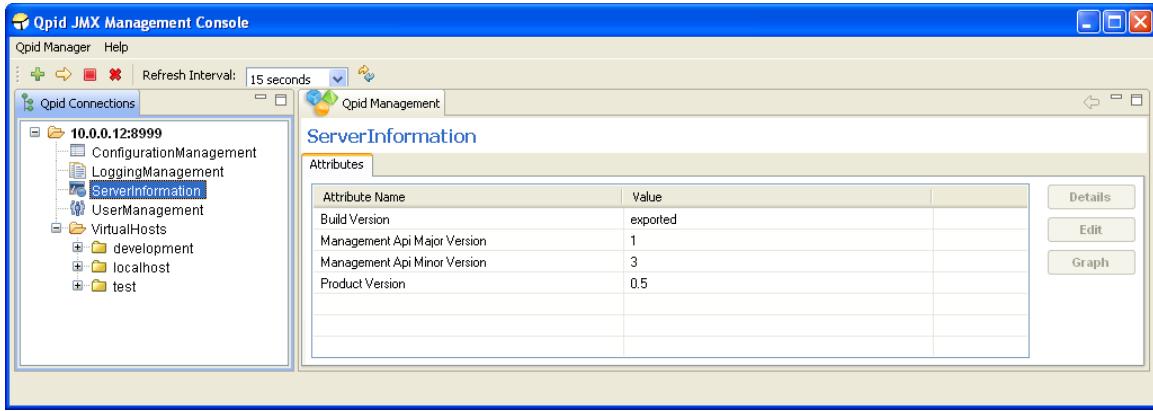
| Test ID | Test Steps | Expected Result |
|---------|--|--|
| LOG-6 | Press the <i>Reload Config File</i> button and confirm the prompt to carry out the action. | The configuration file should be reloaded (success is indicated by lack of error prompts, and a note in the status bar at lower left). |
| LOG-6.1 | Select the <i>Runtime Options</i> tab in the <i>LoggingManagement</i> MBean view. | The <i>Runtime Options</i> tab opens to show the effective Levels of all active Loggers.

The <i>org.apache.qpid</i> Logger and all its children are set to INFO Level, except <i>org.apache.qpid.server.Main</i> which has retained the previously set Runtime Level of WARN.

The <i>Runtime RootLogger Level</i> has returned to INFO, as has the level of all the children Loggers inheriting from it. |
| LOG-7 | Select the <i>org.apache.qpid.server.Main</i> LoggerName and press the <i>Edit Selected Logger</i> button. | The <i>Set Runtime Logger Level</i> dialog opens. |
| LOG-7.1 | Select a Level of INHERITED from the combo box and click ok. | The <i>org.apache.qpid.server.Main</i> Logger is now shown at INFO Level in the table like its parent <i>org.apache.qpid</i> as it once again inherits its Level instead of having its own defined. |

ServerInformation MBean

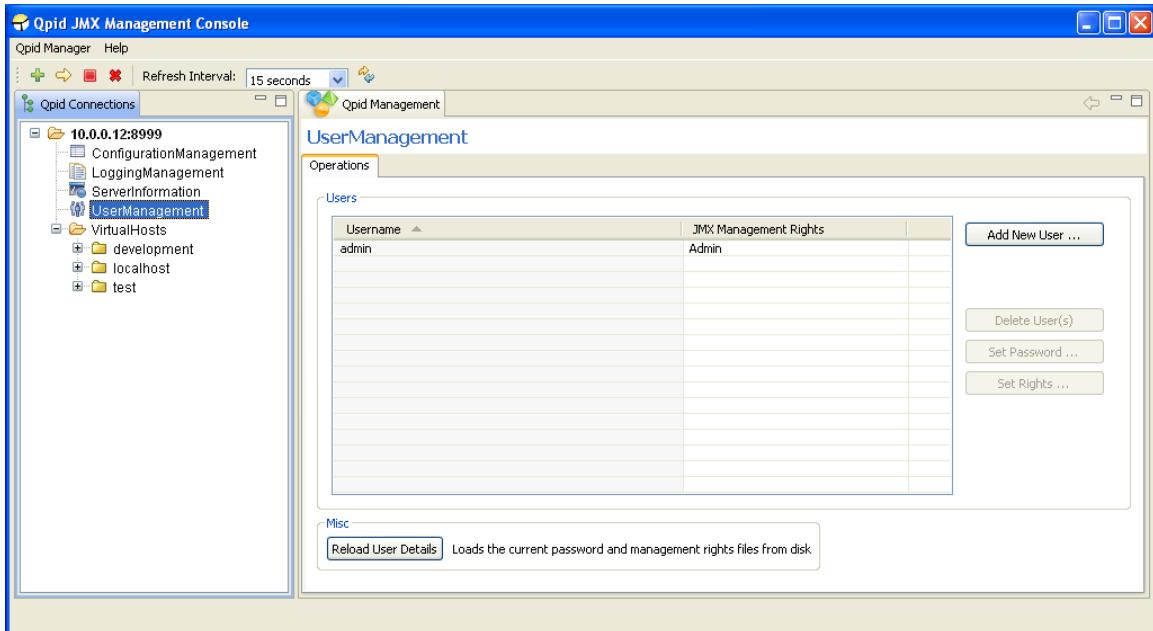
Pre-Requisite: Connect to a server as described in test LOGIN -0 or LOGIN -3 above.



| Test ID | Test Steps | Expected Result |
|---------|---|---|
| INFO-0 | Select the <i>ServerInformation</i> node for the server in the Qpid Connections tree. | The <i>ServerInformation</i> MBean is opened in the MBean view, showing the Attributes tab, displaying version information about the server |

UserManagement MBean

Pre-Requisite: Connect to a server as described in test LOGIN -0 or LOGIN -3 above.

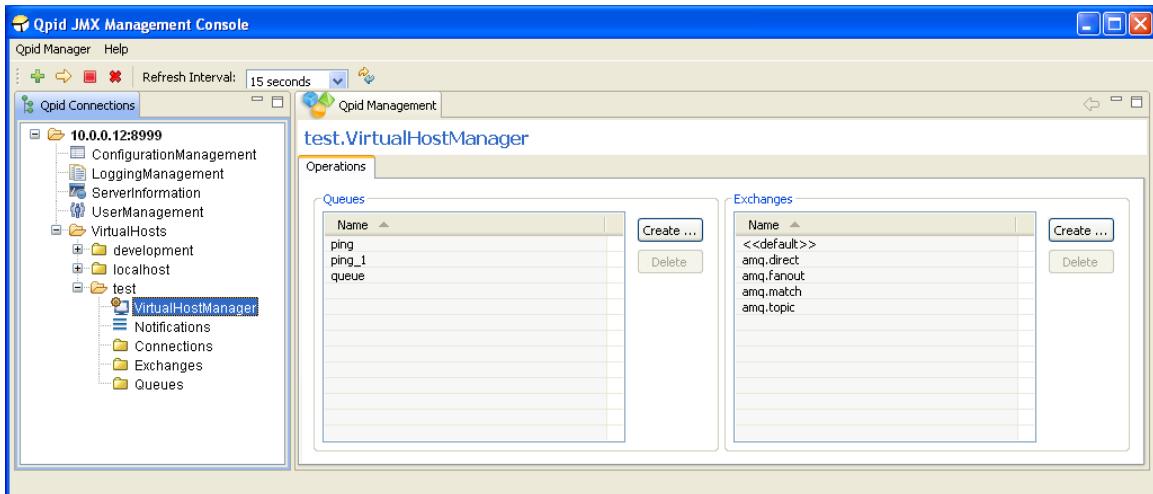


| Test ID | Test Steps | Expected Result |
|----------|---|--|
| USER-0 | Select the <i>UserManagement</i> node in the Qpid Connections tree. | The <i>LoggingManagement</i> MBean is opened in the MBean view |
| USER-1 | Click the <i>Add New User</i> button | The <i>Add New User</i> dialog opens |
| USER-1.1 | Enter Username=guest1, Password=guest1 and select Read Only access rights, then click ok. | <i>guest1</i> will be added to the password and management rights files with indicated password and rights, and be added to the table with <i>Read Only</i> access rights.
NOTE: Verify that the password file and access rights files configured in the server configuration were updated. |
| USER-2 | Select guest1 in the Users table and click the <i>Set Rights</i> button, | The <i>Set Rights</i> dialog opens |
| USER-2.1 | Select Admin rights and press ok. | <i>guest1</i> rights in the management rights files will be changed to admin, and it will be displayed in the table with <i>Admin</i> rights.
NOTE: Verify that the access rights file configured in the server configuration was updated. |

| | | |
|---------------|--|--|
| USER-3 | Select guest1 in the Users table and click the Set Password button, | The <i>Set Password</i> dialog opens |
| USER-3.1 | Enter Password=newpass and press ok. | <i>guest1</i> will be altered to have password <i>newpass</i> in the password file. There will be no visible change in the table.
NOTE: Verify that the password file configured in the server configuration was updated. |
| USER-4 | Select guest1 in the Users table and click the Delete Users(S) button, then validate the operation when prompted for confirmation. | <i>guest1</i> will be removed from the password file and rights file and disappear from the table.
NOTE: Verify that the password file and access rights files configured in the server configuration were updated. |
| USER-5 | Repeat USER-1 &1.1 | As USER-1.1 |
| USER-6 | Repeat USER-1 & 1.1 with Username=client1, Password=client1 and select No Access rights. | As USER-1.1 but with the new credentials, and the rights file will not be modified. |
| USER-7 | Alter the password file on disk to add user1:user1, and alter the access rights file on disk to add user1=readwrite. | No change. The password and rights files are only read once at startup by the server, until instructed to reload them via JMX. |
| USER-8.1 | Press the <i>Reload User Data</i> button | <i>user1</i> will be added to the server and shown in the table with <i>Read & Write</i> access rights.
(Older servers only reload the rights file, and so no change in the table will be visible in this case) |
| USER-8 | Select the <hostname>:8999 node in the Qpid Connections tree and then click the Disconnect button in the toolbar | The connection is closed and the <hostname>:8999 tree node collapses to a single entry. |
| USER-8.1 | Select the <hostname>:8999 node in the Qpid Connections tree and then click the "Reconnect" icon | The <i>Reconnect</i> dialog opens |
| USER-8.2 | Enter Username=user1, Password=user1 and then click Connect. | The server node will be expanded to show the <i>ServerInformation</i> MBean as well as a <i>VirtualHosts</i> folder containing child folders <i>development</i> , <i>localhost</i> , and <i>test</i> .
The <i>ConfigurationManagement</i> , <i>LoggingManagement</i> , and <i>UserManagement</i> MBeans will not be shown as only admin-level users have access to these. |
| USER-9 | Select the <hostname>:8999 node in the Qpid Connections tree and then click the Disconnect button in the toolbar | The connection is closed and the <hostname>:8999 tree node collapses to a single entry. |
| USER-9.1 | Select the <hostname>:8999 node in the Qpid Connections tree and then click the "Reconnect" icon | The <i>Reconnect</i> dialog opens |
| USER-9.2 | Enter Username=client1, Password=client1 and click Connect. | The connection attempt fails, as user <i>client1</i> has no management access rights. |

VirtualHostManager MBean

Pre-Requisite: Connect to a server as described in test LOGIN -0 or LOGIN -3 above.



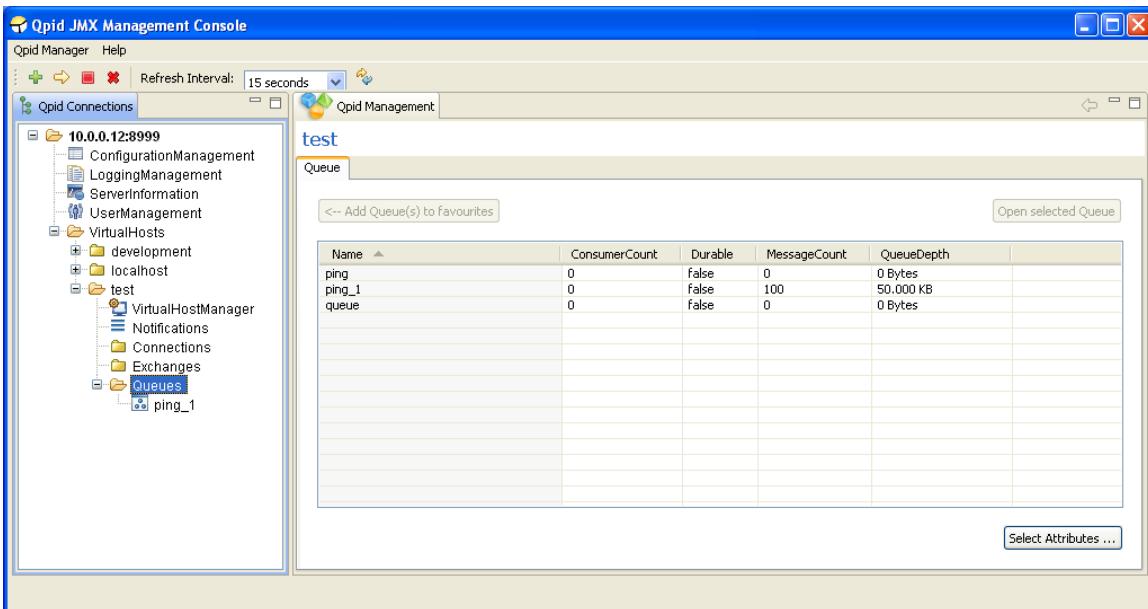
| Test ID | Test Steps | Expected Result |
|-----------|---|---|
| VHOST-0 | Select the <i>VirtualHostManager</i> node for the <i>test</i> VirtualHost in the Qpid Connections tree. | The <i>VirtualHostManager</i> MBean is opened in the MBean view |
| VHOST-1 | Double-click ping_1 in the Queues table. | The ping_1 mbean is opened in the MBean view. |
| VHOST-1.1 | Press the back arrow button at the top right corner of the view. | The VirtualHostManager MBean is opened in the MBean view |

The following tests are based on the use of user with *Admin* or *Read & Write* management access rights. *Read Only* level management rights do not permit a user to perform actions that modify the server state, such as creating or deleting Queues and Exchanges. Attempting such operations will be met by an Access Denied security warning at the point of remote execution.

| | | |
|-----------|--|---|
| VHOST-2 | Press the Create button in the Queues group | The Create Queue dialog opens |
| VHOST-2.1 | Enter Name=newQueue and then press OK. | newQueue is created and shown in the Queues table. |
| VHOST-3 | Select newQueue in the Queues table and press the Delete button in the Queues group. | The Delete Queue(s) dialog opens, listing newQueue to be deleted. |
| VHOST-3.1 | Press the OK button. | newQueue is removed from the server and disappears from the Queues table. |
| VHOST-4 | Press the Create button in the Exchange group | The Create Exchange dialog opens |
| VHOST-4.1 | Enter Name=newExchange and select type=direct then press OK. | newExchange is created and shown in the Exchanges table. |
| VHOST-5 | Select newExchange in the Exchange table and press the Delete button in the Exchanges group. | The Delete Exchange(s) dialog opens, listing newExchange to be deleted. |
| VHOST-5.1 | Press the OK button. | newExchange is removed from the server and disappears from the Exchanges table._- |

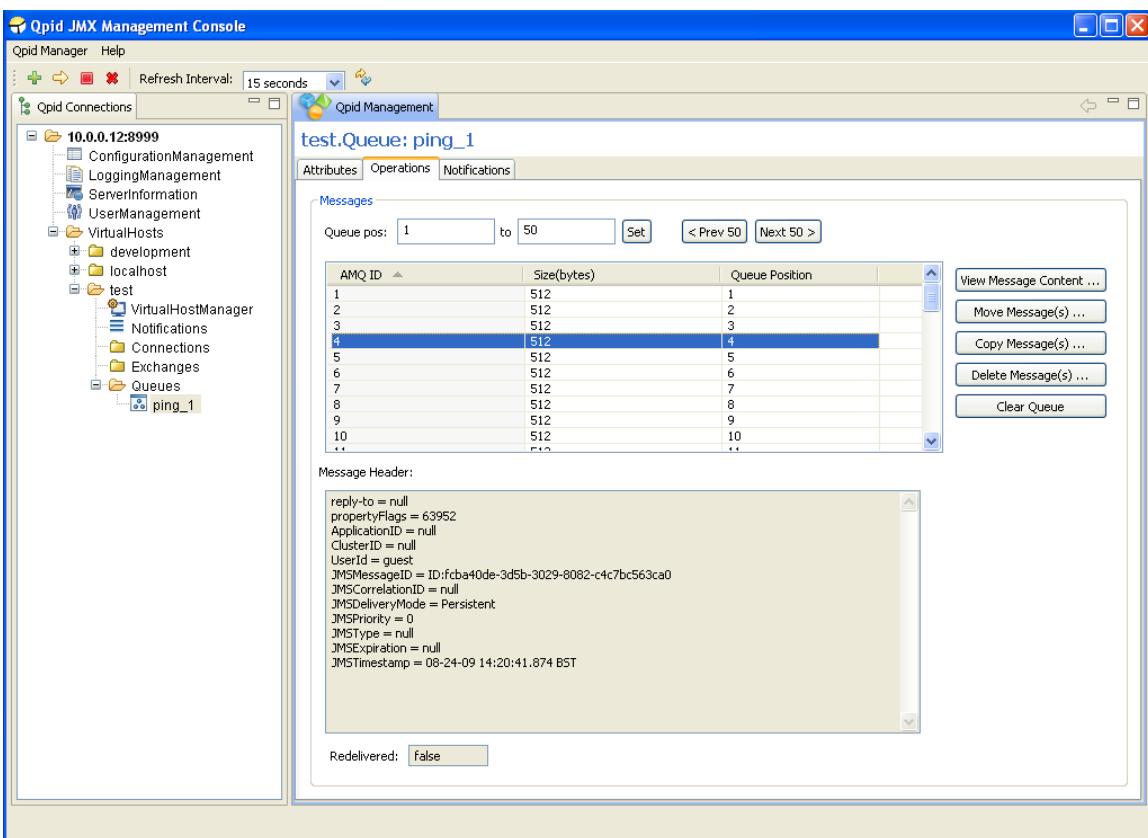
Queue Management

Pre-Requisite: Connect to a server as described in test LOGIN -0 or LOGIN -3 above. Delete the console *qpidmc_queue_attributes.ini* settings file as directed in the initial Console Configuration sub-section.



| Test ID | Test Steps | Expected Result |
|-----------|--|--|
| QUEUE-0 | Select the Queues node for the <i>test</i> VirtualHost in the Qpid Connections tree. | The Queue selection screen is opened in the MBean view, listing Queues ping,ping_1, and queue in the table. |
| QUEUE-1 | Press the SelectAttributes button. | The Select Attributes dialog opens. |
| QUEUE-1.1 | Ensure the following attributes re ConsumerCount, Durable, MessageCount, QueueDepth and then press OK. | The table updates to show additional columns for the new Attributes. All queues have additional attribute values of 0, false, 0, and 0bytes. |

| | | |
|----------------|---|---|
| QUEUE-2 | Run the <i>ping_sender.sh</i> utility script, but do not press a key to exit the script when the sending is complete. | 100 messages of size 512bytes are sent to the <i>ping_1</i> queue. After a refresh interval elapses, the table updates to show <i>ping_1</i> having 100 messages and a queue depth of 50.000 KB, as well as a new queue named TempQueue<etc>. |
| QUEUE-2.1 | Press a key in the script shell to make it exit. | After another refresh interval elapses, the table will update to show the TempQueue has been deleted. |
| QUEUE-3 | Select queue <i>ping_1</i> in the table and click the <i>Add Queue(s) to favourites</i> button, then click the + icon at the Queues node to expand the Queues node. | <i>ping_1</i> has been added as a child node of Queues. |
| QUEUE-3.1 | Select the new <i>ping_1</i> node in the Qpid Connections tree. | The <i>ping_1</i> queue MBean opens in the MBean View, showing the <i>Attributes</i> tab. |
| QUEUE-3.2 | Select the <i>Operations</i> tab in the <i>ping_1</i> MBean view. | The <i>Operations</i> tab opens, with the details of the first 50 messages on <i>ping_1</i> visible in the table. |



| | | |
|----------------|---|---|
| QUEUE-4 | Select the message with AMQ ID 4 in the table. | The message will be highlighted, and its Header details and Redelivered status shown in the lower sections. |
| QUEUE-4.1 | Press the <i>View Message Content</i> button (or double-click the entry in the table) | The result window opens, showing the AMQ Message ID, Content (repeated -message payload statements), Encoding, andMimeType. |
| QUEUE-4.2 | Close the result dialog. | The result window closes. |
| QUEUE-5 | Press the <i>Next 50 ></i> button to advance the viewed message positions. | The range will change to 51 - 100 and the table will update to show messages in positions 51-100 (which at this time possess AMQ IDs 51-100) |
| QUEUE-5.1 | Press the <i>Next 50 ></i> button to advance the viewed message positions. | The range will change to 101 - 150 and the table will update to show messages in positions 101 - 150, which do not exist at this time (only 100 messages were placed on the queue) and so the table is now empty. |
| QUEUE-6 | Enter 11 in the left <i>Queue Pos</i> box, and 20 in the right <i>Queue Pos</i> box, then press the <i>Set</i> button | The table will update to show messages in positions 11-20 (which at this time possess AMQ IDs 11-20). The <i>< Prev 50</i> and <i>Next 50 ></i> buttons have updated to have 10 as their step size. |
| QUEUE-7 | Select the <i>Notifications</i> tab in the <i>ping_1</i> MBean view. | The <i>Notifications</i> tab opens. |

| | | |
|-----------|---|---|
| QUEUE-7.1 | Press the <i>Subscribe</i> button | The <i>Subscribe</i> button becomes disabled, and the <i>Unsubscribe</i> button becomes enabled. After at most 30seconds, Notifications should start being received asserting that <i>ping_1</i> has exceeded its <i>MaximumMessageCount</i> (89 allowed, 100 present) and contains messages over the <i>MaximumMessageAge</i> (10sec allowed, arbitrary actual age over 10sec depending on time taken to execute previous tests) |
| QUEUE-7.2 | Select the <i>Operations</i> tab in the <i>ping_1</i> MBean view. | The <i>Operations</i> tab opens, with the details of message positions 11-20 on <i>ping_1</i> in the table. |

The following tests are based on the use of user with *Admin* or *Read & Write* management access rights. *Read Only* level management rights do not permit a user to perform actions that modify the server state, such as moving, deleting, or copying messages and clearing the queue. Attempting such operations will be met by an Access Denied security warning at the point of remote execution. **NOTE: Copying and deleting messages is only supported on newer servers. If testing older servers, substitute the Copy test (QUEUE-9) with another Move and then skip the Delete test (QUEUE-10).**

| Test ID | Test Steps | Expected Result |
|-----------|---|--|
| QUEUE-8 | Select the messages with AMQ IDs 11,13-14,19-20 (Note that since 0.6 the IDs now start at 2 and so the IDs here will be n+1.)and press the <i>Move Message(s)</i> button_._ | The <i>Move Messages</i> dialog opens, requesting destination queue and confirmation of moving messages with AMQ ID 11,13-14,19-20. |
| QUEUE-8.1 | Press OK (using the <i>ping</i> destination already selected). | The messages with AMQ IDs 11,13-14,19-20 are moved and disappear from the table, which now shows messages AMQ ID 12,15-18, and 21-25. |
| QUEUE-9 | Select the messages with AMQ IDs 12,15-18 and press the <i>Copy Message(s)</i> button_._ | The <i>Copy Messages</i> dialog opens, requesting destination queue and confirmation of moving messages with AMQ IDs 12,15-18. |
| QUEUE-9.1 | Select the <i>queue</i> as the destination queue Press OK. | The messages are copied, as indicated in the status bar at the bottom left of the application, and they continue to be present on the <i>ping_1</i> queue. |
| QUEUE-10 | The messages with AMQ IDs 12,15-18 are still selected. Press the <i>Delete Message(s)</i> button and confirm the prompt_._ | The messages with AMQ IDs 12,15-18 are deleted and disappear from the table, which now shows messages with AMQ ID 21-30 in positions 11-20. |
| QUEUE-11 | Press the <i>Clear Queue</i> button and confirm the prompt. | The queue is cleared of the 90 remaining (and unacquired) messages and the table becomes empty. Note: The number of deleted unacquired messages is reported in the status bar only for newer brokers |

The following test sequence can be undertaken with a user of any level of management access rights, but depend on completing the previous tests that required *Admin* or *Read & Write* management access rights.

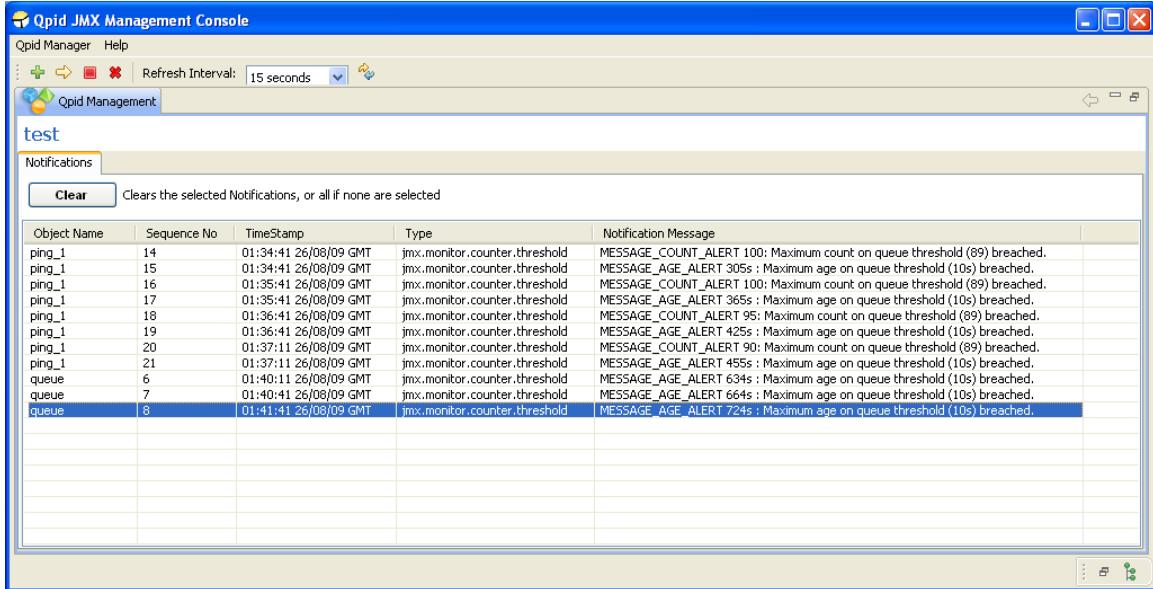
| | | |
|------------|---|---|
| QUEUE-12 | Select the <i>Queues</i> node for the <i>test</i> VirtualHost in the Qpid Connections tree. | The <i>Queue</i> selection screen is opened in the MBean view, listing Queues <i>ping</i> , <i>ping_1</i> , and <i>queue</i> in the table with the attributes <i>ConsumerCount</i> , <i>Durable</i> , <i>MessageCount</i> , and <i>_QueueDepth</i> . These have values for <i>ping</i> , <i>ping_1</i> , and <i>queue</i> :
0, false, 5, 2.500 KB
0, false, 0, 0 bytes
0, false, 5, 2.500 KB |
| QUEUE-12.1 | Double-click the entry in the table for the <i>ping</i> queue | The <i>ping</i> queue MBean opens in the MBean View, showing the Attributes tab. The MessageCount should be 5, the QueueDepth 2560(bytes), MaximumMessageAge 10000 (ms), and MaximumMessageCount 89. |
| QUEUE-12.2 | Select the <i>Operations</i> tab in the <i>ping</i> MBean view. | The <i>Operations</i> tab opens, the table shows the 5 messages on the queue that we moved across in QUEUE-8, AMQ IDs 11,13-14,19-20 |
| QUEUE-12.3 | Press the back arrow button at the top right corner of the view. | The <i>Queue</i> selection screen is opened in the MBean view. |
| QUEUE-13 | Double-click the entry in the table for the <i>queue</i> queue | The <i>queue</i> queue MBean opens in the MBean View, showing the Attributes tab. The MessageCount should be 5, the QueueDepth 2560(bytes), MaximumMessageAge 10000 (ms), and MaximumMessageCount 89. |
| QUEUE-13.1 | Select the <i>Operations</i> tab in the <i>queue</i> MBean view. | The <i>Operations</i> tab opens, the table shows the 5 messages on the queue that we copied across in QUEUE-9, AMQ IDs 12,15-18. |
| QUEUE-14 | Select the <i>Notifications</i> tab in the <i>queue</i> MBean view. | The <i>Notifications</i> tab opens. |

| | | |
|------------|----------------------------|---|
| QUEUE-14.1 | Press the Subscribe button | The Subscribe button becomes disabled, and the Unsubscribe button becomes enabled. After at most 30seconds, Notifications should start being received asserting that <i>queue</i> contains messages older than the MaximumMessageAge (10sec allowed, arbitrary actual age over 10sec depending on time taken to execute previous tests) |
|------------|----------------------------|---|

Notifications

Pre-Requisites: Connect to a server as described in test LOGIN -0 or LOGIN -3 above. Complete the Queue Management testing.

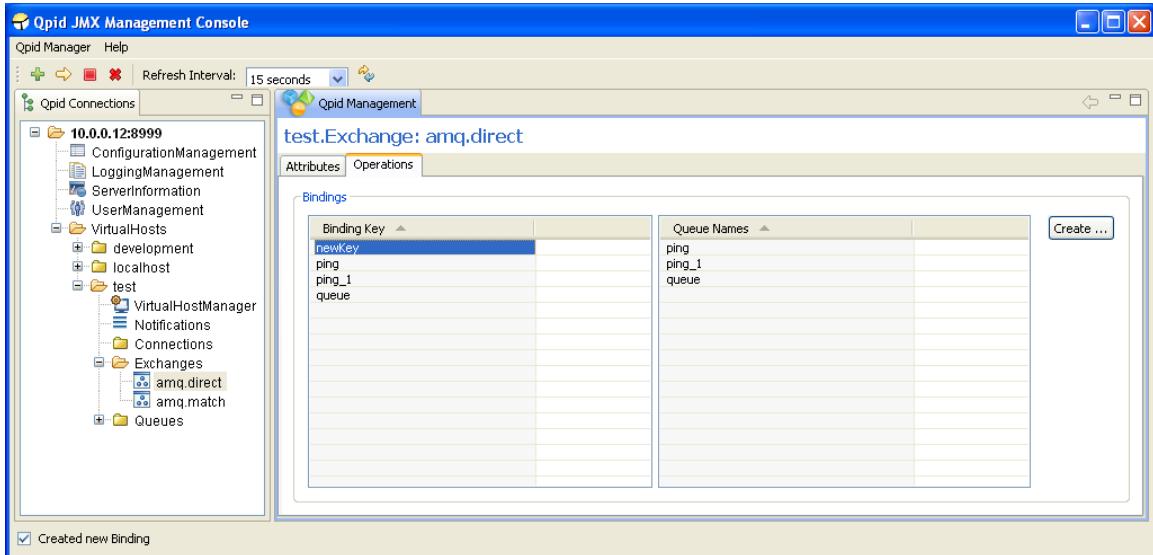
The following test sequence can be undertaken with a user of any level of management access rights, but depend on completing the previous Queue Management tests that at certain points required *Admin* or *Read & Write* management access rights.



| Test ID | Test Steps | Expected Result |
|---------|--|--|
| NOTIF-0 | Select the <i>Notifications</i> node for the <i>test</i> VirtualHost in the Qpid Connections tree. | The VirtualHost <i>Notifications</i> screen is opened.
Notifications received for <i>ping_1</i> are present indicating it holds messages over the MaximumMessageAge of 10sec, and is over MaximumMessageCount (initially at 100, then possibly at 95 if an alert interval fell between the completion of tests QUEUE-8 and QUEUE-9, then possibly at 90 if an alert interval fell between the completion of tests QUEUE-9 and QUEUE-10, all versus an allowed 89 messages).
There will also be Notifications received at the end for <i>queue</i> , indicating it has messages over the MaximumMessageAge of 10 seconds. |
| NOTIF-1 | Select a group (any, but not all) of the Notifications and press the <i>Clear</i> button. | The selected Notifications are removed from the table and no further Notifications are selected in the table. |
| NOTIF-2 | Ensuring no notifications are selected in the table, press the <i>Clear</i> button, then validate the confirmation to proceed with clearing all Notifications from MBeans in the VirtualHost | All remaining Notifications from MBeans in the VirtualHost are removed from the table. |
| NOTIF-3 | Wait at most 30 seconds | Additional Notifications should be received for <i>queue</i> , indicating it has messages over the MaximumMessageAge of 10 seconds. |

Exchange Management

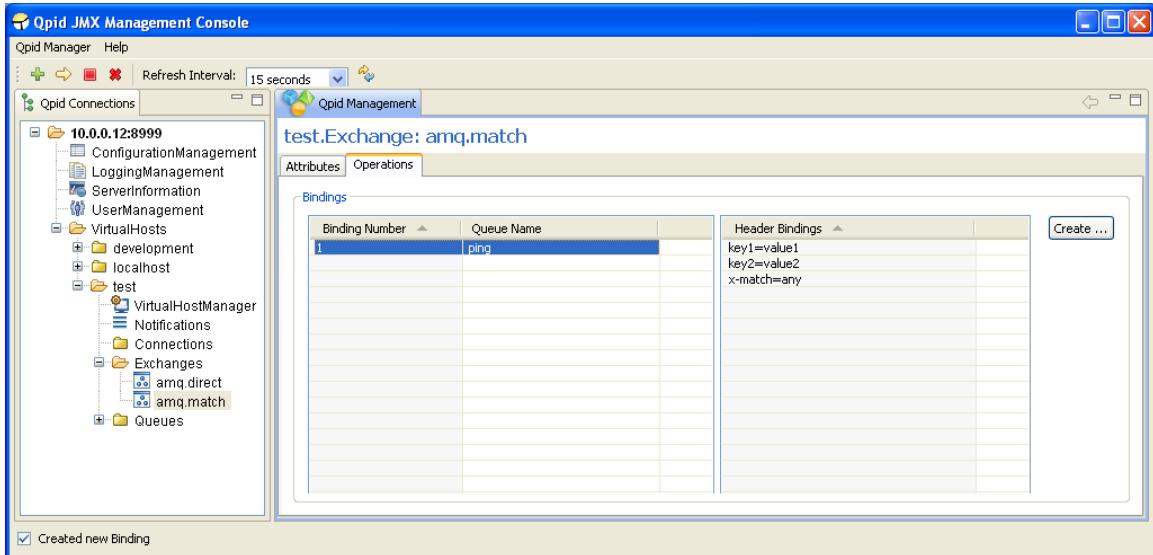
Pre-Requisites: Connect to a server as described in test LOGIN -0 or LOGIN -3 above.



| Test ID | Test Steps | Expected Result |
|---------------|---|--|
| EXCH-0 | Select the <i>Exchanges</i> node for the <i>test</i> VirtualHost in the Qpid Connections tree. | The <i>Exchanges</i> selection screen is opened in the MBean view, listing Exchanges <>, <i>amq.direct</i> , <i>amq.fanout</i> , <i>amq.match</i> , and <i>amq.topic</i> in the table. |
| EXCH-0.1 | Select exchanges <i>amq.direct</i> and <i>amq.match</i> in the table and click the <i>Add Exchange(s) to favourites</i> button, then click the + icon at the Exchanges node to expand the Exchanges node. | <i>amq.direct</i> and <i>amq.match</i> have been added as a child nodes of Exchanges. |
| EXCH-1 | Select the new <i>amq.direct</i> node in the Qpid Connections tree. | The <i>amq.direct</i> exchange MBean opens in the MBean View, showing the <i>Attributes</i> tab. |
| EXCH-1.1 | Select the <i>Operations</i> tab in the <i>amq.direct</i> MBean view. | The <i>Operations</i> tab opens |
| EXCH-2 | Select the <i>ping</i> entry in the Binding Key table. | The Queue Names table updates and shows the <i>ping</i> queue is associated with the selected binding. |

The following tests are based on the use of user with *Admin* or *Read & Write* management access rights. *Read Only* level management rights do not permit a user to perform actions that modify the server state, such as creating bindings. Attempting such operations will be met by an Access Denied security warning at the point of remote execution.

| | | |
|---------------|--|--|
| EXCH-3 | Press the <i>Create</i> button at the right hand side of the Bindings group. | The <i>Create New Binding</i> dialog opens. |
| EXCH-3.1 | enter Binding= <i>newKey</i> and press OK. | The new binding is created and <i>newKey</i> appears in the Binding Key table. |
| EXCH-3.2 | Select the new <i>newKey</i> entry in the Binding Key table. | The Queue Names table updates and shows the <i>ping</i> queue is associated with the <i>newKey</i> binding. |
| EXCH-3.3 | Press the <i>Create</i> button at the right hand side of the Bindings group. When the dialog loads, enter Binding= <i>newKey</i> and select queue <i>ping_1</i> then press OK. | The new binding is created, and the selection in the table is cleared. |
| EXCH-3.4 | Select the <i>newKey</i> entry in the Binding Key table. | The Queue Names table updates and shows the <i>ping</i> and <i>ping_1</i> queues are now both associated with the <i>newKey</i> binding. |

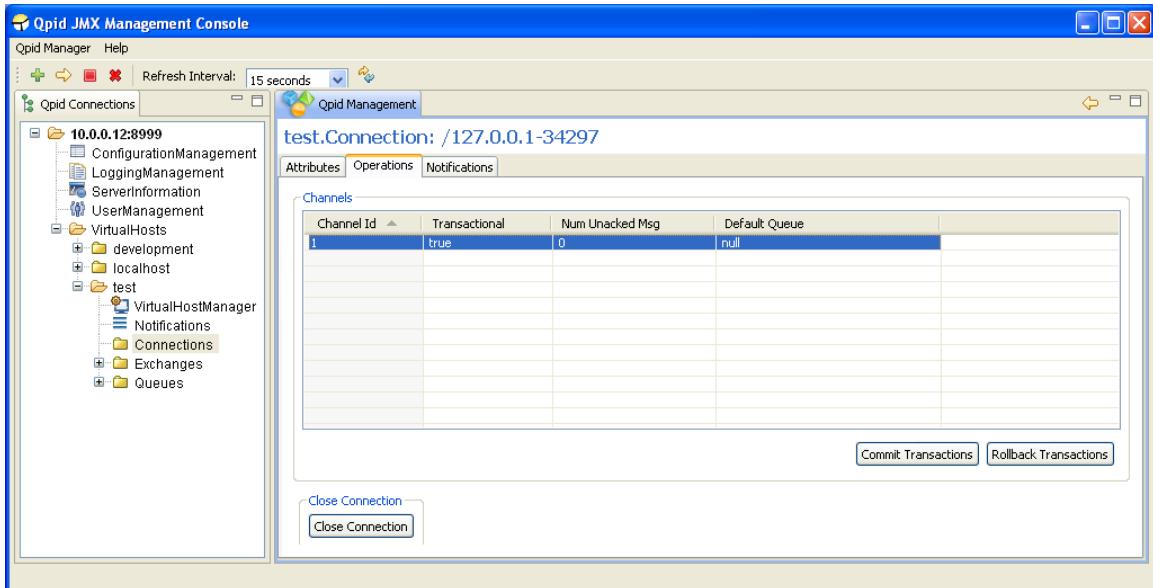


| | | |
|----------------|--|--|
| EXCH-4 | Select the <i>amq.match</i> node in the Qpid Connections tree. | The <i>amq.match</i> exchange MBean opens in the MBean View, showing the <i>Attributes</i> tab. |
| EXCH-4.1 | Select the <i>Operations</i> tab in the <i>amq.match</i> MBean view. | The <i>Operations</i> tab opens. The <i>Binding Number</i> and <i>QueueName</i> table is empty. |
| EXCH -5 | Press the <i>Create</i> button at the right hand side of the Bindings group. | The <i>Create New Binding</i> dialog opens. |
| EXCH-5.1 | Select a value of 'all' for the x-match key. Then enter a new key called key1, with a value of value1, and a new key called key2 with a value of value2. Press the <i>Add additional field</i> button, and a new empty row should be added at the bottom, giving 3 empty rows. Press OK. | The binding is created and an entry added to the table with <i>Binding Number</i> 1 and <i>Queue Name</i> ping. |
| EXCH-5.2 | Select the binding entry in the table. | The Header Bindings table should update to list the keys and values entered in the dialog:

<i>key1=value1
key2=value2
x-match=any</i> |

Connection Management

Pre-Requisites: Connect to a server as described in test LOGIN-0 or LOGIN -3 above.



| Test ID | Test Steps | Expected Result |
|---------------|---|--|
| CONN-0 | Select the <i>Connections</i> node for the <i>test</i> VirtualHost in the Qpid Connections tree. | The <i>Connections</i> selection screen is opened in the MBean view, with no entries in the table. |
| CONN-0.1 | Run the <i>ping_sender.sh</i> utility script, but do not press a key to exit the script when the sending is complete. | After a refresh interval, two connections from the IP address of the machine <i>ping_sender.sh</i> was run will appear in the table |
| CONN-1 | Double-click the first connection entry in the table. | The connection MBean opens in the MBean View, showing the <i>Attributes</i> tab. |
| CONN-1.1 | Select the <i>Operations</i> tab in the connection MBean view. | The <i>Operations</i> tab opens, listing channel 1 in the table, which is Transactional, has 0 unacked messages, and no (null) default queue. |
| CONN-2 | Select the channel entry in the table. | The <i>Commit</i> and <i>Rollback Transactions</i> buttons activate as the selected channel is transactional. |
| CONN-3 | Press the <i>Close Connection</i> button and confirm the action. | The connection is closed, the main view is cleared and a dialog opens to inform the user that the open MBean was unregistered from the server. |

Qpid Management Console Testing (Old UI)

- Background
- Test Platforms
- Introduction
- Installation and Testing
- General Test and Setup Information
- Administration
- Virtual Host Management
- Connections Management
- Queue Management
- Exchange Management
- Trouble Shooting

Background

The Qpid Management Console is user graphical interface for managing and monitoring the Qpid remotely.

This specification is primarily concerned with detailing the functional specifications for Management Console. The tests cover all the functionalities supported by the console.

Test Platforms

The management console currently supports Windows, Linux, Solaris and Mac OS X.

Introduction

This Management Console functional test specifications are categorized into following categories:

1. Administration
2. Virtual Host Management
3. Connections Management
4. Queue Management
5. Exchange Management

Installation and Testing

To install the management console, it can be [] or []

There are three config files that should be used in this test which are attached to this page along with the sendAndWaitClient script.

General Test and Setup Information

When the Qpid Management Console is started for the first time on a machine, it creates a configuration file (*qpidManagementConsole.ini*) in user home directory (C:\Documents and Settings\<userid> for windows). This ini file stores the Qpid server address, which are added to the console.

Following setup is needed for Qpid Management Console test

1. Start the Qpid Broker with following configuration.
 - a. Virtual Host "Development" and "test"
 - b. Virtual Host "test" having following Queues - ping, queue, ping_1
 - c. Virtual Host "test" having following Exchanges - amq.direct (type=direct), amq.topic(type=topic), amq.headers(type=headers) and amq.fanout(type=fanout)
2. Qpid has following user configuration

a. Username=admin, Password=admin (Admin permissions)

User permissions are set in the jmxremote.access file.

Administration

This section details the test cases for administration of management console users. The setup described in the set up information section is required for these tests.

| Test ID | Test steps | Expected result |
|---------|--|--|
| AD-0 | Start the Qpid Management Console with script qpidmc.sh | The GUI starts up with "Qpid Connections" on left hand side. |
| AD-1 | Click the "New Connection" icon. | New Connection details pop up opens |
| AD-1.1 | Enter the host=<hostname>, port= 8999
Username=admin, Password=admin and click Connect. | The Qpid Connections node "<hostname>:8999" will be added. |
| AD-2 | Expand all the child node of "<hostname>:8999" | Following child nodes should be listed.
org.apache.qpid ->
UserManagements->UserManagement |
| AD-3 | Select the UserManagement | <p>Following admin operations are listed as tabs on "Qpid Management" page-</p> <ul style="list-style-type: none"> View Users Set Rights Reload Data Set Password Delete user Create User <p>View Users tab will be selected and following users will be listed, which can be browsed using first/next/previous/last buttons</p> |
| AD-4 | Select the tab "Create User" and enter the user details-
Username=user
Password=password
Read and Write access.
Click Execute. | Operation successful dialog pops up. |
| AD-4.1 | Select the tab "Create User" and enter the user details-
Username=guest
Password=password
Read access.
Click Execute. | Operation successful dialog pops up. |
| AD-4.2 | Select the tab "Create User" and enter the user details-
Username=newuser
Password=password
Read access.
Click Execute. | Operation successful dialog pops up. |
| AD-4.3 | Select the "Reload Data" tab and execute. | Operation successful dialog pops up. |
| AD-4.4 | Select the "View Users" tab and check if the following users are listed - user, guest, newuser. | Users are listed with same permissions as assigned while creating users. |
| AD-5 | Select "Delete User" tab and enter username=newuser. Click execute | Operation successful dialog pops up. |
| AD-5.1 | Select the "Reload Data" tab and execute. | Operation successful dialog pops up. |
| AD-5.2 | Select the "View Users" tab and check if the user "newuser" is listed | User "newuser" is not listed. |
| AD-6 | Select the "Set Rights" tab, enter the username=user and select the "Read" access. Click Execute | Operation successful dialog pops up. |
| AD-6.1 | Select the "Set Password" tab, enter the username=user and password=newpassword.
Click execute. | Operation successful dialog pops up. |
| AD-6.2 | Start another session of Management Console using qpidmc.sh and connect to <hostname>:8999 using username=user and password=newpassword | User logs in. |
| AD-6.3 | Select the VirtualHosts->test->VirtualHostManager. Now enter details on the "Create New Queue" tab and click execute. | "Access Denied" message popup. |
| AD-6.4 | Select the UserManagements->UserManagement | "Access Denied" error pops up. |

| | | |
|--------|--|----|
| AD-6.5 | Disconnect this new session of Management Console. | NA |
|--------|--|----|

Virtual Host Management

This section details the test cases for managing and monitoring a Virtual Host. The setup described in the setup information section is required for these tests.

| Test Id | Test Steps | Expected Result |
|---------|--|--|
| VH-0 | Start the Qpid Management Console with script qpidmc.sh | The GUI starts up with "Qpid Connections" on left hand side. |
| VH-1 | Click the "New Connection" icon. | New Connection details pop up opens |
| VH-1.1 | Enter the host=<hostname>, port= 8999 Username=user, Password=password and click Connect. | The Qpid Connections node "<hostname>:8999" will be added. |
| VH-2 | Expand all the child node of "<hostname>:8999" | Following child nodes should be listed.
org.apache.qpid -> VirtualHosts -> test and development
test->VirtualHostManager, Connections, Exchanges and Queues
development-> VirtualHostManager, Connections, Exchanges and Queues |
| VH-2 | Click on test->VirtualHostManager | Qpid Management view on right hand side displays following operations as tabs-
Create New Queue
Create New Exchange
Delete Queue
Unregister Exchange |
| VH-4 | On the "Create New Queue" tab enter Queue Name=newQueue and click Execute | A confirmation dialog pops up. |
| VH-4.1 | Click yes on the confirmation box | A pop up displays that the operation is successful. |
| VH-4.2 | Click on the test->Queues | Qpid Management page displays the "Queue" tab and following queues will be listed- ping, queue and newQueue |
| VH-5 | Click on test->VirtualHostManager and then select the "Create New Exchange" tab | The "Qpid Management" page displays following the data fields – Name, Type and Durable. |
| VH-5.1 | Enter the Exchange Name=newExchange, select the Exchange Type as "direct" and click Execute. | A confirmation dialog pops up. |
| VH-5.2 | Click yes on the confirmation box | A pop up displays that the operation is successful. |
| VH-5.3 | Click on the test->Exchanges | Qpid Management page displays the "Exchange" tab and following exchange will be listed- amq.direct, amq.fanout, amq.match, amq.topic and newExchange |
| VH-6 | Click on test->VirtualHostManager and then select the "Delete Queue" tab | The "Qpid Management" page displayed following Queues in the list- queue, ping, newQueue |
| VH-6.1 | Select the newQueue from the list and click Execute | A confirmation dialog pops up. |
| VH-6.2 | Click yes on the confirmation box | A pop up displays that the operation is successful. |
| VH-6.3 | Click on the test->Queues | Qpid Management page displays the "Queue" tab and following queues will be listed- ping and queue. The newQueue should not be in the list. |
| VH-7 | Click on test->VirtualHostManager and then select the "Unregister Exchange" tab | The "Qpid Management" page displayed following Exchange in the list- amq.direct, amq.fanout, amq.topic, amq.headers and newExchange |
| VH-7.1 | Select the "newExchange" from the list and click Execute | A confirmation dialog pops up. |
| VH-7.2 | Click yes on the confirmation box | A pop up displays that the operation is successful. |
| VH-7.3 | Click on the test->Exchanges | Qpid Management page displays the "Exchange" tab and following exchange will be listed- amq.direct, amq.fanout, amq.match and amq.topic. The newExchange should not be in the list. |
| VH-8 | Select the Qpid server node <hostname>:8999 and click the "Disconnect" icon from the tool bar. | The Qpid connection is disconnected and the server nodes disappear. |

Connections Management

This section details the test cases for managing and monitoring a Qpid connection. The setup described in Setup information section is required for these tests.

Precondition: A Qpid server (e.g. <hostname>:8999) is added to the Management Console as described in the "Virtual Host Management" section.

| Test Id | Test Steps | Expected Result |
|---------|--|--|
| CM-0 | Select the Qpid Server node "<hostname>:8999" on the Qpid Connections page and click the icon for reconnect from the tool bar. | The GUI starts up with "Qpid Connections" on left hand side. |
| CM-1 | Start a consumer application and select the test->Connections node. | On the right hand page the connection will appear in the list. |
| CM-2 | Select the connection and "Add to Navigation" | The connection node is added to the left hand side page. |
| CM-3 | Select the connection node. | <p>The Qpid Management page displays following connection attributes-
Authorized Id, Client Id, Last IO Time, Maximum Number of Channels, Remote Address and Version</p> <p>Also following tabs are displayed-
Rollback Transactions
Commit Transactions
Close Connection
Channels</p> |
| CM-4 | Select Channels tab | Qpid Management page displays channel details. |
| CM-5 | Select Close Connection tab | A confirmation dialog pops up. |
| CM-5.1 | Click yes on the confirmation box | A pop up displays that the operation is successful and the connection node from left hand side disappears. |
| CM-6 | Select the Qpid server node <hostname>:8999 and click the "Disconnect" icon from the tool bar. | The Qpid connection is disconnected and the server nodes disappear. |

Queue Management

This section details the test cases for managing and monitoring a Qpid Queue. The setup described in Setup information section is required for these tests.

Precondition: A Qpid server (e.g. <hostname>:8999) is added to the Management Console as described in the "Virtual Host Management" section.

| Test Id | Test Steps | Expected Result |
|---------|--|---|
| QM-0 | Select the Qpid Server node "<hostname>:8999" on the Qpid Connections page and click the icon for reconnect from the tool bar. | The GUI starts up with "Qpid Connections" on left hand side. |
| QM-1 | Select test->Queues and add Queue ping_1 to the navigation | Queue ping_1 gets added under test-Queues |
| QM-2 | Select ping_1 node on left hand side | <p>Following Queue properties are displayed in a table on "Qpid Management" page-</p> <ul style="list-style-type: none"> Active Consumer Count Auto Delete Consumer Count Durable Maximum Message Age Maximum Message Count Maximum Message Size Maximum Queue Depth Message Count Name Owner Queue Depth Received Message Count <p>Following operations will be displayed as tabs-</p> <ul style="list-style-type: none"> Clear Queue Delete Message From Top View Messages View Message Content Move Messages |

| | | |
|---------|---|--|
| QM-3 | Run a client application to send 100 messages of size 512bytes each to queue "ping_1" and refresh the attributes tab on management console | Following attributes value will be updated.
Message Count=100
Queue Depth=50 (in kb)
Received Message Count=100 |
| QM-4 | Select the "Maximum Message Count" attribute and click "Edit Attribute" | Attribute window pops up with Attribute Name, Description and value. |
| QM-4.1 | Update the value to 100 | The attribute value gets updated. |
| QM-5 | Select the Notifications tab and select the notification type and subscribe. | Subscribe button gets disabled. |
| QM-6 | Run the same client application again and sent 100 messages of size 512 bytes to the "ping_1" and refresh the attributes tab on management console | Following attributes value will be updated.
Message Count=200
Queue Depth=100 (in kb)
Received Message Count=200 |
| QM-6.1 | Select the Notifications tab. | One MESSAGE_COUNT_ALERT will be listed. |
| QM-6.2 | Clear the notifications. Click the Clear button | The notifications list will be empty. |
| QM-6.3 | Select the notifications type and Unsubscribe. | Subscribe button will be enabled and Unsubscribe button will be disabled. |
| QM-7 | Select the "Delete Message From Top" tab. Click Execute and confirm. | Operation successful dialog will popup. |
| QM-8 | Select Attributes tab. | Following attributes value will be updated.
Message Count=199
Queue Depth=99 (in kb) |
| QM-9 | Select the "Clear Queue" tab. Click Execute and confirm. | Operation successful dialog will popup. |
| QM-10 | Select Attributes tab. | Following attributes value will be updated.
Message Count=0
Queue Depth=0 (in kb) |
| QM-11 | Run the same client application again and sent 100 messages of size 512 bytes to the "ping_1" and refresh the attributes tab on management console. | Following attributes value will be updated.
Message Count=100
Queue Depth=50 (in kb)
Received Message Count=300 |
| QM-12 | Select the "View Messages" tab and enter this data-
from Index=1
to index=100 and Execute | The results window pops up with message header attributes details of 100 messages on the Queue. |
| QM-13 | Close the results window and select "View Message Content" tab. Enter the message id =250 and Execute. | The results window pops up with following fields-
Message Id
Content
Encoding
Mime Type |
| QM-14 | Select "Move Messages" tab. | The Qpid Management page shows these fields-
From Message Id
To Message Id
Queue (with list of all other queues) |
| QM-14.1 | Enter the following data-
From Message Id = 200
To Message Id = 150 and Select Queue="queue". Now Execute and confirm. | Operation successful dialog will popup. |
| QM-14.2 | Select test-Queues and add queue to the navigation and select "queue" | The queue attributes will be displayed with these values-
Queue depth=25
Message Count=50
Received Message Count=50 |

Exchange Management

This section details the test cases for managing and monitoring a Qpid Exchanges. The setup described in Setup information section is required for these tests.

Precondition: A Qpid server (e.g. <hostname>:8999) is added to the Management Console as described in the "Virtual Host Management" section.

| Test Id | Test Steps | Expected Result |
|---------|------------|-----------------|
|---------|------------|-----------------|

| | | |
|--------|---|---|
| EM-0 | Select the Qpid Server node "<hostname>:8999" on the Qpid Connections page and click the icon for reconnect from the tool bar. | The GUI starts up with "Qpid Connections" on left hand side. |
| EM-1 | Select test->Exchanges and add these exchanges to the Navigation page-
amq.direct
amq.fanout
amq.topic
amq.match | Those exchanges get listed under the node test->Exchanges |
| EM-2 | Select "amq.direct" node | The exchange attributes get displayed on "Qpid Management" page and following tabs-
Create New Binding
Bindings |
| EM-3 | Select the "Create New Bindings" tab | The available queues (ping, queue) will be listed. |
| EM-3.1 | Select the Queue "ping" and enter a binding "newBinding". Click "Execute" and confirm the operation. | Operation successful dialog will popup. |
| EM-3.2 | Select the "Bindings" tab | Queue "ping" with binding "newBinding" will be listed on "Queue Management" page. |
| EM-4 | Select the node "amq.match", which is headers exchange | Result same as in step EM-2 |
| EM-4.1 | Select the "Create New Binding" tab | The available queues (ping, queue) will be listed and there will be text fields to enter binding as key=value pair |
| EM-4.2 | Select a Queue "ping" and enter these bindings-
Name=key1 value=value1
Name=key2 value=value2

Now click "Execute" and confirm the operation. | Operation successful dialog will popup. |
| EM-4.3 | Select the "Bindings" tab | Queue "ping" with bindings "key1=value1" and "key2=value2" will be listed on "Queue Management" page. |
| | | |

Trouble Shooting

Q. Why the text fields are not visible on my machine but are visible on other machine?

A. Please check if the windows theme being used in Control Panel->Display is not the default one.

Q. How can I get the exception stack trace for sending to development team for debugging?

A. In start-up scripts qpidmc.sh or qpidmc.bat, update the parameter value:

-Declipse.consoleLog=true

Testing Design - Java Broker CPU GC Monitoring

Java Broker CPU/GC Monitoring

When testing the Java broker with the perftest suite of tests one of the problems is that we only gather the result of the tests. If the numbers are better than last time great. However, investigating how the broker handled the load is probably a good thing to do. If CPU usage jumped 100% for only 10% performance benefit we should look at why. Similarly if we spend a lot of time in GC we should check what extra garbage we have started creating.

So as a starter lets enable verbose:gc on the tests runs and monitor the CPU usage additional monitoring can be added to the suite but the first step is to have the ability to gather the data.

- Data collection
 - JVM GC Logging
 - CPU Monitoring
- Test Execution
 - Broker Monitoring Setup
 - GC Gathering
 - CPU Usage Gathering
 - Log Data Processing
 - GC
 - CPU
- Result Presentation

Data collection

Current goals are to focus on the Java broker by collecting CPU usage (measured via top -p), and GC data as written by the JVM with verbose gc logging. This same information could be gathered for the client JVM however it would require modification to the existing scripts to log the data.

This data can then be graphed for later review. Eventually some form of automated analysis could be performed from an automated testing platform which could identify problematic commits earlier in our release cycle.

JVM GC Logging

Start the vm with verbose gc being logged to its own file. The file contains log entries in seconds since VM start up, this makes correlating that time with other output, such as CPU, difficult. As The gc file is created very close to the VM startup the file access time can be taken as its creation time. This time can then be used as a start point for the offsets of each log entry. On a linux box the access time can be seen via 'ls -tu' and is shown in minutes or 'stat' which shows the additional seconds field.

CPU Monitoring

Running 'top' in batch mode and fixed on the JVM process with -p will provide a sample of the CPU usage of the process which can then be aligned to the GC data using the same file creation time method. If more data points are required the frequency of top updates can be modified on the command line. An initial period of 0.5 seconds should give good coverage when compared to the GC output.

Test Execution

Integrating with our existing test cases rather than requiring a rewrite of the tests is desirable as we can focus on the collection of data rather than updating the existing test cases.

Broker Monitoring Setup

This section details how we can go about gathering GC and CPU data.

GC Gathering

The JVM has a number of extra GC options which allow us to gather the data very easily. Setting QPID_OPTS to the following value will create a gc.log file with GC details.

```
export QPID_OPTS="-Xloggc:gc.log -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps"
```

CPU Usage Gathering

Gathering CPU usage statistics can be done via 'top' running in batch mode. A simple script can be created to provide the a monitoring rate and a PID to monitor.

```
top -d <CPU_MONITOR_RATE> -Sbcp <PID> > broker_cpu.log
```

This script will create entries in the broker_cpu.log of the following format:

```
top - 05:16:32 up 24 days,  1:04,  6 users,  load average: 0.08, 0.03, 0.31
Tasks:  1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.8% us,  0.3% sy,  0.0% ni, 98.8% id,  0.0% wa,  0.0% hi,  0.1% si
Mem:  4040220k total, 2814272k used, 1225948k free,  194860k buffers
Swap: 16386292k total,        0k used, 16386292k free,  2270140k cached

      PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM     TIME+ COMMAND
27774 ritchiem  16    0 1254m  64m 8564 S    0  1.6  0:01.17 java -server -DPNAM
```

Log Data Processing

GC

The GC log file has a number of types of entries:

Successful Full GC

```
0.503: [Full GC (System) 0.503: [CMS: 0K->1454K(63872K), 0.0617910 secs] 8321K->1454K(83008K),
[CMS Perm : 10933K->10925K(21248K)], 0.0619320 secs]
```

ParNew run

```
9.351: [GC 9.351: [ParNew: 19119K->2112K(19136K), 0.0141410 secs] 50757K->42135K(83008K),
0.0142560 secs]
```

The CMS phases

```
9.366: [GC [1 CMS-initial-mark: 40023K(63872K)] 42272K(83008K), 0.0016310 secs]
9.367: [CMS-concurrent-mark-start]
9.407: [CMS-concurrent-mark: 0.040/0.040 secs]
9.407: [CMS-concurrent-preclean-start]
9.408: [CMS-concurrent-preclean: 0.001/0.001 secs]
9.408: [CMS-concurrent-abortable-preclean-start]
10.495: [CMS-concurrent-abortable-preclean: 0.113/1.088 secs]
10.498: [CMS-concurrent-sweep-start]
10.508: [CMS-concurrent-sweep: 0.010/0.010 secs]
10.509: [CMS-concurrent-reset-start]
10.517: [CMS-concurrent-reset: 0.008/0.008 secs]
```

Failed Full GC

```
357.779: [Full GC 357.779: [CMS357.885: [CMS-concurrent-abortable-preclean: 0.199/0.990 secs]
(concurrent mode failure): 961425K->13192K(962444K), 0.2641230 secs] 963545K->13192K(981580K),
[CMS Perm : 17808K->17777K(29804K)], 0.2649910 secs]
```

YG Rescan

```
10.496: [GC[YG occupancy: 2151 K (19136 K)]10.496: [Rescan (parallel) , 0.0016840 secs]10.497:
[weak refs processing, 0.0007330 secs] [1 CMS-remark: 259108K(260864K)] 261259K(280000K),
0.0025130 secs]
```

These entries can allow us to generate a graph of memory usage as the application runs. At a first approach using the 'ParNew run' entry we can graph the heap usage (50757K->42135K) and the current maximum heap size (83008K).

Timing

As the log entries are timestamped in seconds since VM startup if we want to correlate these values with the CPU or client log then we must convert them to real times. This can be done by looking at the 'Access' time of the gc.log. When retrieving the log file (or after starting the JVM) we can gather this information using 'stat':

Output from stat

```
File: `2009-05-22-1016/broker-results/logging/gc.log'
Size: 1183063          Blocks: 2320          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 363245        Links: 1
Access: (0644/-rw-r--r--)  Uid: (  500/ritchiem)  Gid: (  500/ritchiem)
Access: 2009-05-29 11:24:00.000000000 +0100
Modify: 2009-05-22 13:42:22.000000000 +0100
Change: 2009-05-22 13:42:22.000000000 +0100
```

As long as the file has not been opened for reading then the 'Access' time will be the time that the file was created. We can then use this as the base for the offsets in the log file.

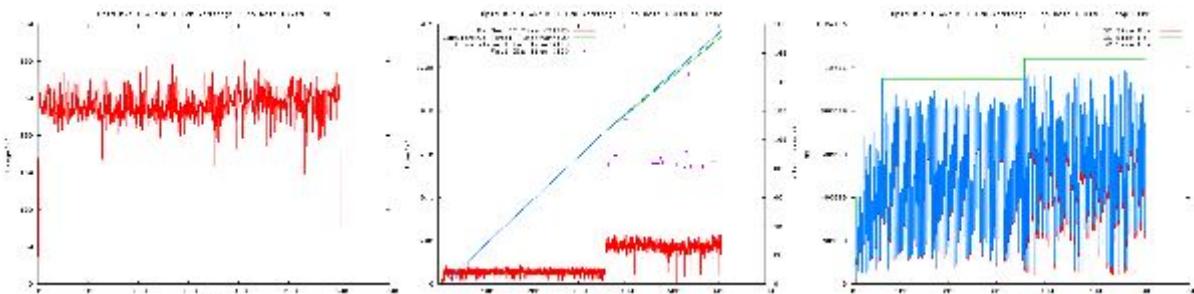
CPU

Processing the CPU data can be done in a similar way to the GC log file. As we know how often the top command will run we can use the stat output to give us a base entry for the log and then increment each log entry by the logging interval.

There is a risk here that top does not accurately log at the specified rate. However if logging is performed at sub second intervals the effort in extracting the time data from top and calculating the millisecond value for the log entry is not believed to be worth the effort.

Result Presentation

The results of the GC and CPU can be put through gnuplot and graphed here is an example of what it might look like:



Source Repository

Web Browsing of SVN

To browse via the web use the ViewVC interface:

<http://svn.apache.org/viewvc/qpid/trunk/qpid>

Or to browse the source tree directly:

<https://svn.apache.org/repos/asf/qpid/trunk/qpid>

Checking out from SVN

The source code can be checked out anonymous over HTTP by doing:

```
svn co http://svn.apache.org/repos/asf/qpid/trunk
```

Commiters can check out the code over HTTPS:

```
svn co https://svn.apache.org/repos/asf/qpid/trunk
```

Read only GIT repo

A read only GIT repo is available:

It can be cloned with

```
git clone git://git.apache.org/qpid.git qpid
```

or

```
git clone http://git.apache.org/qpid.git qpid
```

and then git pull will fetch updates.

If you have commit access it is also possible to commit back with git svn dcommit by following the instructions on the [GitAtApache](#) page.

Setting up your subversion client

When adding files to subversion, it's important that your subversion client is properly setup to the appropriate subversion properties are set. The client can do it automatically by modifying the auto-props section of the subversion config file. Use the contents of:

```
http://svn.apache.org/repos/asf/qpid/trunk/etc/svn-auto-props
```

Mailing Lists

Qpid Mailing lists

There are a number of lists listed below. Note when sending subscription emails it is best to have a value in the subject and body even if it is only 'subscribe'. This will help the email get past the spam filters.

Qpid User List

The user's list is for discussions that relate to use or questions on Qpid. If you have questions about how a feature works, suggestions on additional requirements, or general questions about Qpid please use this list

- To subscribe to the user's list send an e-mail with subject 'subscribe' to users-subscribe@qpid.apache.org

- To remove yourself from the user's list send an e-mail with subject 'unsubscribe' to users-unsubscribe@qpid.apache.org
- The user's mailing list is archived. You can view the archive at http://mail-archives.apache.org/mod_mbox/qpid-users/
Nabble archive <http://n2.nabble.com/Apache-Qpid-users-f2158936.html>

Qpid Developer List

The developer's list is for discussions that relate to the on going development of Qpid. If you have questions about how a feature is being developed, suggestions on how to implement a new feature, or requests for a new feature this is the list to use.

- To subscribe to the developer's list send an e-mail with subject 'subscribe' to dev-subscribe@qpid.apache.org
- To remove yourself from the developer's list send an e-mail with subject 'unsubscribe' to dev-unsubscribe@qpid.apache.org
- The developer's mailing list is archived. You can view the archive at http://mail-archives.apache.org/mod_mbox/qpid-dev/
- Nabble archive <http://www.nabble.com/Qpid-Developers-f16694.html>
<http://n2.nabble.com/Apache-Qpid-developers-f2158895.html>

Qpid Commits List

The commits list is for receiving notifications about code being committed to the Qpid repository. The traffic on this list is automatically generated by Subversion. You should not post messages to this list.

- To subscribe to the commits list send an e-mail with subject 'subscribe' to commits-subscribe@qpid.apache.org
- To remove yourself from the commits list send an e-mail with subject 'unsubscribe' to commits-unsubscribe@qpid.apache.org
- The commits mailing list is archived. You can view the archive at http://mail-archives.apache.org/mod_mbox/qpid-commits/

Useful Links

Purpose

Links to related projects/useful stuff for developers