Operational Logging - Status Update - Technical Specification

Technical Specification

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.

Psuedo-Code Example

```
// logActor is retreived 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>]
```

AMQMinaProtocolSession

```
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.

AMQMinaProtocolSession

```
_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 repective model objects	