Operational Logging - Status Update - Test Specification

Test Specification

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

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 *BRK* ID 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 *BRK* ID 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 / 5672

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 *BRK* ID 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 *BRK* ID 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 *BRK* ID 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 *BRK* ID 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.

testManagementStartupRMIRegistry

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'

testManagementStartupRMIRegistryCustom

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 Registgry : 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.

testManagementStartupRMIConnectorServer

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'

testManagementStartupRMIConnectorServerCustom

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

testManagementShutdownRMIRegistry

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.

testManagementShutdownRMIConnectorServer

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 *VHT* ID 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 *VHT* ID 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 logged 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 explict QueueDelete request must result in a *QUE-1002* Deleted message being logged. This can be done via an explict 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 logged 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 DurableTopicSubiber
   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.