JoramMQ 1.18
MQTT Server
User Manual

Last modified Sep 27, 2023 *release 1.18*

Content

Illustration index

1 Introduction

The MQTT protocol has been invented in 1999, and has become an OASIS standard in 2014. MQTT is a lightweight event and message oriented protocol allowing devices to asynchronously and efficiently communicate across constrained networks to remote systems. MQTT is now becoming one of the standard protocols for the Internet of Things.

MQTT relies on a messaging server following the hub and spoke model of Message Oriented Middleware (MOM). Every MQTT client, data processing application or device, producer or consumer, needs to connect to a central server before communicating with other MQTT clients. The server accepts published messages and delivers them to the interested consumers according to a Publish/Subscribe interaction pattern.

This document is the user manual of the MQTT server included in the JoramMQ Enterprise offering. MQTT versions 3.1 and 3.1.1 are fully implemented with the following features:

- MQTT QoS levels 0, 1 and 2
- message priority
- full subscription: discard old messages or reject new messages
- dynamic topics
- shared subscription support for MQTT (load balancing for MQTT clients sharing subscriptions)
- default access control based on SSL/TLS and user authentication with password
- plugin interface to customize the access control module
- clustered and distributed topics
- WebSocket connector (secure or not)
- MQTT bridge with another MQTT server
- required $SYS topics as specified at https://github.com/mqtt/mqtt.github.io/wiki/SYS-Topics
- interoperability with JMS 2.0

2 Installation

2.1 Requirements

2.1.1 Java version

Since JoramMQ 1.11 Java 8 or later is required.

This is due to the upgrade of the sshd-core bundle for security reasons. However if you use Java 7 JoramMQ will work but you will get error messages on startup and the jorammq-admin commands (SSH administration, see chapter 10) will not work. You can avoid these error messages by removing the sshd-core and jorammq-ssh bundles from the list of started bundles (file conf/felix.properties).

```
felix.auto.start.1= \
 …
 file:./bundle/sshd-core.jar \
 file:./bundle/jorammq-ssh.jar \
 …
```

2.1.2 Available memory (RAM)

By default, the maximum amount of memory used by a JoramMQ server is 2 GB. This is specified by the property JVM_PROPERTIES in the file 'bin/jorammq-server'. Therefore at least 2GB should be available. If not, then the maximum should be lowered accordingly.

2.2 Release

The following archive is provided:

```
jorammq-mqtt-<version>.zip
```

The distribution is expanded in the directory jorammq-mqtt-**<version>**. The directory layout of a JoramMQ installation is as follows:

Directory	Content
bin	Commands, e.g. start, stop, update
bundle	OSGi bundles used by JoramMQ
conf	Configuration files and resources
doc	JoramMQ MQTT user manual

javadoc	Java documentation of the API provided by JoramMQ to implement plugins, e.g. for customizing the access control module
lib	Java libraries used by JoramMQ (bootstrap and other dependencies)
license	Product license

2.3 Environment variables

The environment variable JORAMMQ_MQTT_HOME has to be assigned with the absolute path of the installation directory. For example, on Linux (bash shell) it would be:

```
export JORAMMQ_MQTT_HOME=<absolute path>/joramMQ-mqtt-<version>
```

This directory contains the static data resulting of the installation of JoramMQ (bin, bundles, conf, doc and lib directories), as well as the dynamic persistence data after startup (data and log directories).

The environment variables JORAMMQ_DATA_DIR and JORAMMQ_STORAGE_DIR allow to move the persistent data outside the installation directory:

- JORAMMQ_DATA_DIR allows to fix the location of the data directory, it normally contains all the execution data. By default, *<JORAMMQ_MQTT_HOME>*/data.
- JORAMMQ_STORAGE_DIR allows to fix the path of base directory for JoramMQ transactional persistence. By default: *<JORAMMQ_MQTT_HOME>*/joramMQ.

When the leveldb library is used, which is the rule except in very specific configurations, then the path to the JoramMQ storage directory must be made of true ASCII characters. Please take care of this constraint when choosing the installation directory of JoramMQ.

The environment variable JMX_PORT defines the JMX listening port. If not defined (default) the JMX remote connection is not configured.

The environment variable JORAMMQ_TMP_DIR allows to configure the directory to use for temporary files. By default JoramMQ uses the OS temporary directory (“/tmp” or “/var/tmp” on Linux OS, “%USERPROFILE%\AppData\Local\Temp” on Windows). Make sure to regularly clean this directory.

The environment variable JORAMMQ_JAVA_OPTS allows to fix specific Java options when launching JoramMQ. For example, you can configure a verbose output of SSL handshaking by setting JORAMMQ_JAVA_OPTS to "-Djavax.net.debug=ssl:handshake:verbose".

These variables are used in the setenv scripts.

2.4 Start a server

Launch the following command:

```
joramMQ-server
```

Initial traces look like:

```
The JORAMMQ_MQTT_HOME environment variable is not defined.
Use JORAMMQ_MQTT_HOME="…\jorammq-mqtt-<version>"
Launching JoramMQ MQTT server 0 data_dir:…\jorammq-mqtt-<version>\data
28/02/2023 08:22:29.011 com.scalagent.jorammq.mqtt.adapter.MqttAdapter WARN:
Unlimited JoramMQ MQTT server version <version> for ScalAgent D.T.
=====================================================
Unlimited JoramMQ MQTT server version <version>
for ScalAgent D.T.
=====================================================
28/02/2023 08:22:29.039 com.scalagent.jorammq.mqtt.adapter.KeyStoreStatus WARN:
Checking MQTT adapter key store .\conf\keystore: found no valid X509 trusted
certificate
AgentServer#0 started: OK
```

The following directories are created after the first start:

Directory	Content
data	All the working and temporary files used by JoramMQ, especially for the persistent data and the transaction logs. See section 2.3 to change the location of this directory.
log	Log files

2.5 Stop a server

A server is stopped with the command bin/jorammq-admin:

```
jorammq-admin -stop
```

In some situations the server may not terminate quickly, you can then use the command below:

```
jorammq-admin -exec "exit 0"
```

2.6 Restart a server

Call the same command as the one used to start the server:

```
jorammq-server
```

If the server has been killed (e.g. kill -9) an error may be raised by the server because the lock file has not been deleted:

```
TxLogTransaction.init : TxLog.init(): Either TxLog is already running, or you have
to remove the lock file: /home/sadt/jorammq-1.3.0/data/jorammq/s0/lock
```

In that case, remove the lock file manually:

```
rm ./data/jorammq/s0/lock
```

If you need to avoid creating the lock file, then you can set the following property in conf/jorammq.xml:

```
<property name="Transaction.UseLockFile" value="false"/>
```

2.7 Reset a server

If you want to restart from a clean state, you can delete the directory data/:

```
rm -rf ./data/
```

2.8 Upgrading the license key of a server

To upgrade a server license, you must replace the old key with the new one in the configuration file "conf/jorammq.xml"[1]. Then to update the configuration, follow the procedure described in section 3.13.1.

2.9 Upgrading a server

When upgrading a server, the data store of the new version may not have the same format as the data store of the old version. Therefore, all the persistent MQTT messages should be first retrieved from the old server, before moving to the new one[2].

The OSGi Felix platform keeps a cache of the libraries used. When upgrading versions it is therefore important to delete the data/felix directory before restarting the server.

Since version 1.13 the compatibility issues between versions are detailed in chapter 27.

2.10 Installing JoramMQ as a Linux service

JoramMQ will soon provide a service installer for Linux and Windows, meanwhile you will find instructions for installing it with LSB (see 20.3.1) or systemd (see 20.3.2) in HA chapter.

2.11 Issue with jleveldb component

JoramMQ uses LevelDB as backend persistent storage. LevelDB is a fast key-value storage library written at Google that provides an ordered mapping from string keys to string values.

1 The key is contained in the value of the "com.scalagent.jorammq.key" property.
2 There is currently no automatic tool to upgrade the data store of an old server. If you need one, please contact us (contact@scalagent.com).

JoramMQ uses either a port of LevelDB to Java or a JNI wrapping to native LevelDB C++ libraries. This second solution is preferred for performance reasons but it is currently only available in some environments: Linux (x86, x86-64, aarch64), OS/X (x86, x86-64) and Windows (x86, x86-64).

However even in these environments there may be an error when initializing LevelDB, this error results in the message “Cannot use LevelDB with JNI. Using LevelDB Java version instead” when starting JoramMQ.

JoramMQ remains fully functional but it may be more efficient to correct this error:

1. Linux / Windows: In most cases this problem results from access rights problems on the temporary directory used by Java (write and execute rights).

    1. Try setting the java temporary directory location used by JoramMQ defining the OS environment variable JORAMMQ_TMP_DIR.

    2. On linux this problem is often linked to the “noexec” attribute when mounting the /tmp directory, you can remount /tmp without noexec.

2. Windows: You need to install the “Microsoft Visual C++ 2010 Redistributable”, that solves the issue on Windows machines.

3 Configuration

3.1 Resources

Configuration files are located in the directory conf/. The following table lists the configuration files, gives the content, the way to edit the file and to update the server. The content of those files is described in dedicated sections of this document starting from section 4.

File	Content	Edition	Update
acl.xml	Access control rules for client identifiers and topic publish/subscribe	Modify a copy of the file and move the copy to conf/	Automatically done by the server
admin-cert.pem	Example of certificate (X509 format) used by the secure shell	Directly modify the file	Stop and restart the server
bridge.xml	Bridges to other MQTT servers	Directly modify the file	Stop and restart the server
cmdLog.properties	Logging properties for commands 'jorammq-admin' and 'jorammq-password'	Directly modify the file	Not applicable as not used by the server
felix.properties	OSGi platform Felix properties	Directly modify the file	Stop and restart the server
jorammq.xml	Server properties	Directly modify the file	Stop the server Call bin/jorammq-admin -update Restart the server
keystore	Data (keys, certificates) to authenticate the local server to its peers and to authenticate the remote MQTT clients to the local server	Use the Java command 'keytool'	Stop and restart the server
log.properties	Logging configuration properties for the server	Directly modify the file	Stop and restart the server[3]
monitor.xml	JMX monitoring data published to MQTT topics	Modify a copy of the file and move the copy to conf/	Automatically done by the server

[3] Logger's levels can now be dynamically configured through AgentServer's Mbean.

passwd.properties	List of the user names and passwords	Use the command 'bin/passwd'	Automatically done by the server

3.2 OSGi configuration: felix.properties

The felix.properties file configures the OSGi[4] part of JoramMQ. It includes notably the felix.auto.start.1 property which defines the list of OSGi bundles (jar files) to start when the JoramMQ broker is launched. Several optional services of JoramMQ requires changing this property in order to execute the appropriate service bundle. This is done by copying the “file:...” lines required by the service in this list, removing the leading comment and adding a trailing ‘\’[5].

Be careful to add the line at the proper place in the list. The bundles are started in the order of the list, and this order may be important to the right execution of the services.

🛠 **Warning :** If you modify the list of bundles you need to stop the server, delete Felix bundle cache 'data/felix/', and restart the server[6].

3.3 Logging configuration: log.properties

See chapter 26.

3.4 Basic optimizations

Many parameters allow you to optimize JoramMQ and adapt it to your use. Most of these parameters are accessible from the “jorammq.xml” file (“conf” directory of your installation), their default value corresponds to normal use.

These parameters are described in detail in Chapter 6, among them:

- Transaction.FileSize (section 6.1.3) and Transaction.MinFileCount (section 6.1.4) allow to control the disk space used by transactional logs (by default 10 x 10Mb).
- Transaction.SyncOnWrite (section 6.1.8) force a disk synchronization for each write operation. This ensures that no messages are lost even in the event of a system crash.
- Transaction.AsyncWrite (section 6.1.9) allow to factorize transaction operations and to reduce their global cost. This has a very positive impact on scalability when the number of connections is high, but it increases the latency and can reduces the throughput for a particular connection. Setting this parameter to false reduces memory usage. Transaction.BufferSize (section 6.1.10) and Transaction.BufferTimeout (section 6.1.11) allow to finely configure this mechanism.
- Transaction.UseNioFileChannel (section 6.1.13) determines the use of NIO. Normally should be true on Linux, and false on Windows. The impact of this parameter on performance is significant.

[4] OSGi provides a modular system and a service platform for the Java programming language that implements a complete and dynamic component model (see https://www.osgi.org).
[5] The "\" character at the end of the above line indicates that the property value continues on the next line; It must then be the last character of the line.
[6] It resets only the code cache and there is no data lost.

- maxMessageTableSize (section 6.3) determines the maximum amount of memory allowed to keep the enqueued MQTT messages. Beyond that, messages are swapped to disk.

3.5 Default listening ports

The following table lists the ports which are used by default.

Port	Usage	Configuration file	Activated by default	Available protocols
1883	MQTT standard port	conf/jorammq.xml	Yes	MQTT/TCP MQTT/SSL
1884	MQTT for a large number of clients	conf/jorammq.xml	Yes	MQTT/TCP MQTT/SSL
3333	JMX administration provided by Java (JVM)	bin/jorammq-server	Yes	TCP SSL
7050	Jolokia	conf/felix.properties	No	Only HTTP
8989	JoramMQ Rest API	conf/felix.properties	YES	Only HTTP
9001	MQTT/WebSocket	conf/jorammq.xml	Yes	MQTT/WebSocket/TCP MQTT/WebSocket/TLS
18090	Secure shell used to monitor and control the server	conf/felix.properties	Yes	SSH
17500	TCP entry point for JMS/Joram	conf/jorammq.xml	Yes	TCP SSL
17600	JNDI registry	conf/jorammq.xml	Yes	Only TCP
17700	Used in a cluster for the server-to-server communication	conf/jorammq.xml	No	TCP SSL

Some ports can be secured but by default access are not secured. If security is required then the ports that cannot be secured should be disabled (Jolokia, JNDI).

🛠 **Warning :** Dropbox uses port 17500, 17600 and 17603.

3.6 Limiting resources usage

By default there is no limit to the number of connections or subscriptions. However if you host JoramMQ on a server where resources are limited (CPU, memory or network bandwidth) it can be crucial to control the bandwidth or the maximum TCP connections in order to save resources and avoid abuse of malicious clients.

All limit properties reside in the jorammq.xml configuration file and they can be adjusted at starting. Currently you can apply a global limit to the number of active connections and subscriptions. There is an additional property allowing to throw an alert if the number of connections (resp. subscriptions) exceeds a percentage of the limit.

Property	Value
com.scalagent.jorammq.maxNbOfConnections	Limit the number of connections (0 to ignore), the value must be less than the value registered in the license key.
com.scalagent.jorammq.maxNbOfSubscriptions	Limit the number of subscriptions (0 to ignore), the value must be less than the value registered in the license key.
com.scalagent.jorammq.thresholdPercentage	Percentage alert threshold (0 to ignore),

Memory (RAM)

The maximum amount of memory assigned to the MQTT server is configured by the property JVM_PROPERTIES in the file bin/jorammq-server. By default 2 GB is assigned to the server.

```
JVM_PROPERTIES=-server -Xmx2G
```

3.7 Message cleaning

When clients are not connected to JoramMQ for a significant time then many messages can be stored by the server. These old messages may be outdated and do not need to be delivered, JoramMQ allows you to regularly clean these messages and avoid to deliver them.

You can configure this mechanism from two points of view:

- First, you can specify a TTL (Time-To-Live) for each message published on a specified topic or subtree.
- Second, you can specify a TTL for each message should be delivered to a given client.

The cleaning mechanism may be configured by various properties:

- The property “com.scalagent.jorammq.mqtt.clean.messages.period” allows to define the activation period of a regular task that cleans all outdated messages. This value is in seconds, by default 0 (no activation). Regardless of the periodic activation of this task, the cleanup operation is activated on each client context when it connects.
- The property “com.scalagent.jorammq.mqtt.clean.default.ttl” allows to define a default time-to-live for all messages. This value is in seconds, by default 0 (no TTL).
- The property “com.scalagent.jorammq.mqtt.clean.topics” allows to define a comma separated list of topics and their associated TTL. A topic in the list can be specified specifically (“/A/B” for example define a specific rule) or generically by a topic filter (using the “#” and “+”). If several rules apply to a topic then the specific one define its TTL.

- The property “com.scalagent.jorammq.mqtt.clean.clients” allows to define a comma separated list of client and their associated TTL. A client in the list can be specified specifically or through a regular expression (see class java.util.regexp.Pattern). If several rules apply to a topic then the specific one define its TTL.

If there is no rule for a given message (nor about the topic, nor about the client) the default TTL is used. If several rules apply to a message then the more restrictive one define its expiration time. If one of the applicable rules defines a negative TTL for a message, that message will never be garbaged.

For example:

```
<!-- Period to run the messages cleaning task (unit: seconds) -->
<property name="com.scalagent.jorammq.mqtt.clean.messages.period" value="3600" />

<!-- Set default TTL for messages (unit: seconds) -->
<property name="com.scalagent.jorammq.mqtt.clean.default.ttl" value="240" />

<!-- Set the message TTL for some topics (unit: seconds) -->
<property name="com.scalagent.jorammq.mqtt.clean.topics"
          value="topic/sensors/sensor1=60, topic/sensors/sensor2=60,
topic/sensors/+=30" />

<!-- Set specific TTL for durable clients (unit: seconds) -->
<property name="com.scalagent.jorammq.mqtt.clean.clients"
          value="sensor[0-9]*=120, admin=-1, client1=240" />
```

🛠 **Warning :** Since JoramMQ 1.13 the cleaning task period is no longer defined by com.scalagent.jorammq.mqtt.clean.period property.

3.8 Session cleaning

Durable sessions are normally kept without time limit by the server. An explicit mechanism can be activated through SSH commands to clean these old sessions (see section 10.4.4). JoramMQ also allows you to regularly clean these sessions.

The cleaning mechanism may be configured by various properties:

- The property “com.scalagent.jorammq.mqtt.clean.sessions.period” allows to define the activation period of a regular task that cleans old sessions. This value is in seconds, by default 0 (no activation).

- The property “com.scalagent.jorammq.mqtt.clean.sessions.conf” allows to define the file containing the properties pair defining the expiration period of sessions. This file can be either in a simple line-oriented format or in XML format (filename endings by “.xml”). By default the configuration file is “./conf/cleanSessions.properties”.

- The name part of each property defines a filter on the clientId of the sessions. The value part defines the expiration period in seconds for these sessions. All sessions whose 'clientID' matches the regular expression[7] of the filter, and which are inactive for at least the number of seconds indicated by the expiration period, are removed. If the duration is 0, all inactive sessions whose 'client ID' matches the regular expression are deleted.

For example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">

<properties>
  <comment>XML Configuration file for MQTT sessions cleaning</comment>

  <entry key="nodemcu\d\d">3600</entry>
  <entry key="user.*">86400</entry>
</properties>
```

3.9 Monitoring of client contexts

Since version 1.15 of JoramMQ it is possible to automatically monitor client contexts to detect when they store too many messages[8]. Multiples indicators can be monitored :

- ToDeliverMessageCount: Number of messages waiting to be delivered.
- NotAckedSize: Total payload size of the unacknowledged messages.
- DisconnectionTime: duration in seconds since the last time the client was connected.

For each of these properties you can decide on an alert threshold and an error threshold. When the threshold is exceeded, the corresponding message is recorded in the log file. If the threshold is less than or equal to 0 it is not checked.

The properties used to set the thresholds are:

- com.scalagent.jorammq.warnMaxToDeliverMessageCount (default 0).
- com.scalagent.jorammq.errorMaxToDeliverMessageCount (default 0).
- com.scalagent.jorammq.warnMaxNotAckedSize (default 0).
- com.scalagent.jorammq.errorMaxNotAckedSize (default 0).
- com.scalagent.jorammq.warnDisconnectionTime (default 0).
- com.scalagent.jorammq.errorDisconnectionTime (default 0).

[7] See documentation of java.util.regex.Pattern Java class.
[8] It is common to have ghost client contexts with subscriptions but no longer any consumption activity.

Another property allows to configure the activation period in seconds, the checking task is not activated if less than or equal to zero (default):

- com.scalagent.jorammq.checkClientContext.period

🛠 **Warning :** The values of these properties are modified in the “jorammq.xml” configuration file.

3.10 Dynamic allocation of client identifier

Since MQTT v5 the Server allows a client to supply a ClientID that has a length of zero bytes. The Server treats this as a special case and assign a unique ClientID to that Client.

The identifiers allocated by the server then consist of a prefix and an index. The prefix is by default “JMQDCId”, its value can be modified using an environment variable defined in “conf/jorammq.xml”:

- com.scalagent.jorammq.mqtt.assignedClientIdPrefix

3.11 Topic alias

Since MQTT v5 client and server can define Topic alias that can be used as a substitute for topic names. When connecting, the client and the server exchange the maximum number of aliases to be used. The maximum number of aliases used by server is defined by an environment variable defined in “conf/jorammq.xml”:

- com.scalagent.jorammq.mqtt.maxTopicAlias

Its default value is 5.

3.12 Java garbage collection (GC)

Java's default GC option is -XX:+UseParallelGC. Another GC can be configured in 'bin/jorammq-server', for example:

```
JVM_PROPERTIES=-server -Xmx2G -XX:+UseParallelOldGC
```

3.13 Configuration updates

This section explains how the file 'conf/jorammq.xml' has to be updated. The updating procedure depends on whether the server is standalone, clustered, and/or replicated for high availability (HA).

3.13.1 Standalone server

A standalone server needs to be stopped and restarted to take into account configuration updates.

Since JoramMQ 1.15, by default the modifications of the configuration are taken into account each time JoramMQ is started. Thus, the steps for modifying the configuration are:

- Modify the file 'conf/jorammq.xml'
- Stop the server

```
jorammq-admin -stop
```

- Restart the server

```
jorammq-server
```

Old behavior

If you want to keep the old behavior and makes the updates explicitly, you must modify the UPDATE_CONF_AUTO variable at the beginning of the server launch script (bin/jorammq-server or bin/jorammq-server.bat).

Any value other than "OK" will require the explicit update of the configuration.

In this case when the file 'conf/jorammq.xml' is modified then the command 'jorammq-admin -update' needs to be called before the server is restarted, otherwise the server still uses the old configuration which is stored in binary format in the directory “data/”.

The steps to update the configuration are:

- Modify the file 'conf/jorammq.xml'
- Stop the server

```
jorammq-admin -stop
```

- Update the configuration

```
jorammq-admin -update
```

- Restart the server

```
jorammq-server
```

3.13.2 Clustered and/or replicated (HA) server

Like a standalone server, a server which is either in a cluster, or replicated (HA), or both (in a HA cluster), also needs to be stopped and restarted to take into account configuration updates. Moreover, there may have been some changes in the configuration saved in the data store that are not reflected in the XML configuration, i.e. the file 'conf/jorammq.xml'. Those changes may happen in two cases:

1. Clustered servers (see section 18): if the cluster is modified
2. Replicated servers for HA (see section 20): if the XML configuration on a different node (that used to be the primary node) has been updated

In both cases, the XML configuration needs to be synchronized with the configuration saved in the data store before being modified.

The steps to update the configuration are:

- Synchronize the file 'conf/jorammq.xml'

```
jorammq-admin -sync
```

- Modify the file 'conf/jorammq.xml'
- Stop the server

```
jorammq-admin -stop
```

- Update the configuration

```
jorammq-admin -update
```

- Restart the server

```
jorammq-server
```

4 MQTT access control

A default access control module is provided by JoramMQ. At MQTT Connect, a simple authentication mechanism checks that the MQTT user name is registered and that the password is correct.

You can customize this basic behaviour by implementing a new access control module (see section 4.3).

4.1 Create an MQTT user

4.1.1 Set the password

Call the command bin/jorammq-passwd[9] as follows:

```
jorammq-passwd [username] [password]
```

This command populates the file conf/passwd.properties. Passwords are translated with the message digest algorithm SHA-512 before being stored in the file.

4.1.2 Specify the access control rules

Create an element 'user' in conf/acl.xml. For example the user named “guest” would be defined as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<acl xmlns="http://www.scalagent.com/schema/MqttAclSchema">
  <user name="guest" >
```

Create a child element 'clientId' for every prefix allowed for the MQTT client identifiers. In this example, two prefixes are allowed, “g0-” and “g1-”:

```
    <clientId prefix="g0-" />
    <clientId prefix="g1-" />
```

Create a child element 'topic' for every accepted topics. Topics are specified by a topic filter and an access mode that can be:

- 'r' for reading: the user is allowed to subscribe;
- 'w' for writing; the user is allowed to publish;
- 'rw' for both; the user is allowed to subscribe and publish;

For example:

```
    <topic filter="a" mode="r" />
```

[9] By default a unique user 'guest' with password 'guest' is declared in this file.

```
    <topic filter="b/a" mode="w" />
    <topic filter="c/#" mode="rw" />
    <topic filter="d/+/a" mode="rw" />
  </user>
```

🛠 **Warning :** By default a new user does not have any authorization, the definition of a rule in the file conf/acl.xml is thus obligatory to allow the connection.

4.1.3 Specify generic access control rules

To factorize the definition of your security policy, you can define access control rules that apply to more than one user.

Start by creating a "generic" element in the configuration file "conf/acl.xml". The user attribute allows to define a regular expression specifying the users to whom the rule applies. In our example below, it applies to all users whose names begin with “nodemcu” followed by two digits.

```
  <generic user="nodemcu\d\d" >
```

Create a child element 'clientId' for each prefix allowed for the MQTT client identifiers. To define this prefix, you can use the $ {user} tag replaced by the actual user name when the rule is applied. In the example below, a unique prefix is allowed, it matches the username followed by '-':

```
    <clientId prefix="${user}-" />
```

Create a child element 'topic' for each accepted topic. Topics are specified in the same manner as for the normal access control rule. As for the 'clientId' element, you can use the ${user} tag. For example:

```
    <topic filter="home/${user}/data/#" mode="w" />
    <topic filter="home/${user}/ctrl/#" mode="r" />
  </user>
```

4.2 Configuration

Access control is enabled in conf/jorammq.xml by the following property.

```
<property name="com.scalagent.jorammq.mqtt.adapter.accessControlEnabled"
    value="true" />
```

Default value is false (access control disabled). The access control can be enabled (resp. disabled) for each MQTT connector using the connector’s properties (see 5.3.11).

Two other properties give the path of the password file and the path of the file containing the access control rules:

```
<property name="com.scalagent.jorammq.mqtt.accesscontrol.password.file"
    value="./conf/passwd.properties" />
```

```
<property name="com.scalagent.jorammq.mqtt.accesscontrol.acl.file"
    value="./conf/acl.xml" />
```

If the password file location is changed, it should also be changed in the command bin/jorammq-passwd.

A property allows to change the checking period (in ms) of the configurations files:

```
<property name="com.scalagent.jorammq.mqtt.accesscontrol.update.period"
    value="30000" />
```

🛠 **Warning :** In the default configuration the 'health-check' module uses the guest user. If you remove this user, or change the access control rules you have to take care to avoid periodic exceptions (see chapter 12).

4.3 Customize the access control module

The way to customize the access control is described in sections 21.1 and 21.3.

4.4 IP Access restriction policy

JoramMQ provides additional security measure allowing to restrict access from certain IP's. You can use this function by creating a list of IP addresses to which the access will be allowed. Access from IP's which are not in the list will be denied. By default if there is no list defined all IP addresses are allowed.

You can define a specific list for each connector or a global list to restrain access for all connectors. Each list is built as a comma-separated list of IP addresses[10] and is declared as a property in the jorammq.xml configuration file:

- com.scalagent.jorammq.mqtt.ipallowed for the gobal list,
- com.scalagent.jorammq.mqtt.<connector>.ipallowed for a list dedicated to a particular connector where the connector name is formatted as “<protocol>-<port>”.

If a specific list is declared for a connector, the global list, if it exists, is ignored.

There is no filtering for access from the server itself (loopback or local IP addresses).

Example:

```
<!-- List of IP addresses that will be accepted by all connectors. -->
<property name="com.scalagent.jorammq.mqtt.ipallowed"
          value="192.168.1.0/24, 192.168.10.1, aurore.scalagent.com" />

<!-- List of IP addresses that will be accepted by TCP-1883 connector. -->
<property name="com.scalagent.jorammq.mqtt.tcp-1883.ipallowed"
          value="216.58.208.206, 54.239.26.128" />
```

[10] “a.b.c.d” or “a.b.c.d/m” where m specifies the number of significant bits in the address.

🛠 **Warning :** Currently this mechanism only works with IPv4 address.

4.4.1 Configuration updates

When the file 'conf/jorammq.xml' is modified then a standalone server needs to be stopped and restarted (see 3.13) to take into account configuration updates:

- bin/jorammq-admin -stop
- bin/jorammq-admin -update
- bin/jorammq-server

5 MQTT connectors

Several MQTT connectors can be created depending on the various QoS levels you need to be provided to the MQTT clients.

5.1 Creation

A connector is created by passing a URI as a parameter of the MQTT service declared for a particular JoramMQ server.

The URI is used to specify the name of the protocol below the MQTT layer, the listening address and port, and some QoS properties (declared with their short names, see 5.3).

By default, 3 connectors are created[11]:

```
<service class="com.scalagent.jorammq.mqtt.adapter.MqttAdapter" args=
 "tcp://0.0.0.0:1883
  tcp://0.0.0.0:1884?prod.win=1024&amp;client.mx.inflight=128&amp;snd.buf=1024
&amp;rcv.buf=1024
  ws://0.0.0.0:9001 " />
```

The available secure protocol is TLS/SSL. It can be activated for TCP/IP and Web Socket connections.

5.2 Available protocols

The available protocols under MQTT are listed in the following table:

Protocol scheme	Specifies
tcp	MQTT over TCP/IP.
ssl	MQTT over SSL
sslv*<version number>*	MQTT over the specified version of SSL
tls	MQTT over TLS
tlsv*<version number>*	MQTT over the specified version of TLS
ws	MQTT over Web Socket
wss	MQTT over secure Web Socket
invm	InVM connector allowing efficient communication from embedded components (MQTT-SN gateway for example).

[11] Thanks to the small TCP read/write buffer size (1024 bytes) the connector listening to port 1884 can handle many MQTT clients while consuming a low level of memory.

5.2.1 InVM connector

The InVM connector allows to use the MQTT adapter through an internal protocol to the JVM. This connector is still experimental.

```
invm://0.0.0.0:0
```

Properties

- com.scalagent.jorammq.mqtt.adapter.InVMTransport.queue.capacity
  - maximum size of the connector's message queue.
  - default value = 100

5.2.2 TLS connector

The TLS connector provides a secured access to the broker. The standard port used for MQTT over TLS is 8883, registered as secure-mqtt by IANA.

```
tls://0.0.0.0:8883
```

Configuration

All the parameters used by a TCP connector can be used by the TLS connector. In addition the TLS connector accepts parameters related to the configuration of the security policy.

JoramMQ uses the javax.net.ssl package to implement the TLS connector. This Java API requires a keystore identifying the broker, and a truststore holding certificates of trusted authorities. JoramMQ uses a single store for both roles (cf 5.3.21 MQTT server Keystore).

Minimal configuration

The broker keystore must at least contain a valid certificate associated with its private key (PKCS12 entry).

With such a configuration the MQTT client can authenticate the JoramMQ broker, to some extent. The MQTT client must have access to a store holding the certification chain of the broker certificate, so that the TLS layer accepts the certificate provided by the broker.

Extended broker authentication

As we said before, the minimal configuration enables the client to authenticate the broker. However some SSL APIs enable it to ignore the value of the CN field of the certificate. This is the case with the Paho Java API[12]. In that case you may configure all your brokers with the same key store. All brokers then use the same certificate, and the clients actually recognize the certificate as a valid one. However they are not able to authenticate each individual broker.

If your security policy requires your JoramMQ brokers to be individually authenticated, be sure to use an MQTT client which checks the CN field of the broker certificate. This is the case of the Paho Java client by default. In that case the CN field of the broker certificate must match the name that the client can retrieve from the IP address of the broker.

[12]when the client uses the call MqttConnectOptions.setHttpsHostnameVerificationEnabled(false)

Client authentication

The TLS protocol allows the broker to authenticate the client, after the client has authenticated the broker. This step is optional and controlled by the property clientAuth (cf 5.3.12 Client authentication).

In that case the client must have access to a keystore holding a certificate with its associated private key, in addition to the truststore required to authenticate the broker. On the other side, the keystore of the broker must hold the certification chain of the client certificate.

The broker does not check the value of the CN field of the client certificate during the SSL handshake. It is then possible to use the same certificate for all the clients. If your security policy requires the broker to authenticate each individual client, then you must write a security plugin (cf 21.3 Access controlAccess control) where you will be able to read the client certificate and apply any treatment of your choice.

Health-check component

The health-check component (cf 12 MQTT connector health-check) connects to the declared connectors of the JoramMQ broker as a standard external client. Its default configuration reuses the keystore of the broker, meaning that it retrieves the key/certificate of the broker itself, which should be accepted by the broker[13].

Debug

Properly configuring SSL security may be arduous. No clear message explains the reasons why a connection does not establish. In that case you may activate the Java SSL debugging using the Java option “-Djavax.net.debug=all”, and detailed logs about the negotiation of the SSL session will be issued onto the stderr channel.

The simplest way to activate this function is to set and export the JORAMMQ_JAVA_OPTS environment variable before starting the broker.

```
$ export JORAMMQ_JAVA_OPTS="-Djavax.net.debug=ssl:handshake:verbose"
$ bin/jorammq-server
```

5.3 QoS properties

Every QoS property can be assigned either globally for all the connectors of a server, or specifically for a given connector, directly in the URI and identified by a short name. When a specific value is set for a given connector, the global value is overloaded.

5.3.1 Send and Receive buffer size

Receive buffer size

Size of the receiving buffer in bytes.

[13]In versions previous to the 1.15.0 the health-check client required a proper CN field in the broker certificate.

```
<property name="com.scalagent.jorammq.mqtt.adapter.receiveBufferSize"
    value="32768" />
```

Short name:

```
rcv.buf
```

Send buffer size

Size of the sending buffer in bytes.

```
<property name="com.scalagent.jorammq.mqtt.adapter.sendBufferSize"
    value="32768" />
```

Short name:

```
snd.buf
```

MQTT over Web Socket

MQTT over WebSocket connector is built on top of Jetty, receiveBufferSize allow to configure the maximum message size of the underlying transport. It can help to avoid error messages with big messages:

```
Frame discarded. Binary aggregation disabed for ..
```

5.3.2 Producer window size

Maximum number of message payload bytes that can be accepted for a given MQTT publisher client before doing flow control.

The number of payload bytes is incremented when a Publish message is received from the publisher and decremented when the message is enqueued for delivery.

Above the limit, messages are not accepted any more from the MQTT connection (flow control is enabled). As soon as the number of payload bytes goes below half the limit, messages are accepted (flow control is disabled).

```
<property name="com.scalagent.jorammq.mqtt.adapter.maxProducerWindowSize"
    value="2048576" />
```

Short name:

```
prod.win
```

5.3.3 Consumer window size

Maximum number of message payload bytes allowed to be in-flight for a given MQTT subscriber client before doing flow control.

This limit is only valid for enqueued messages, i.e. messages whose QoS level is either 1, 2 or 0 if the Queued QoS 0 flag is true.

The number of payload bytes is incremented when a Publish message is delivered to the subscriber and decremented when the message is dequeued:

- after acknowledgement for QoS levels 1 and 2;
- after the message has been sent through the connection for QoS level 0.

Above the limit, messages are not sent any more to the MQTT connection (flow control is enabled). As soon as the number of payload bytes goes below half the limit, messages are sent (flow control is disabled).

```
<property name="com.scalagent.jorammq.mqtt.adapter.maxConsumerWindowSize"
    value="2048576" />
```

Short name:

```
cons.win
```

5.3.4 Keep Alive timer

The Keep Alive timer, measured in seconds, defines the maximum time interval between messages received from a MQTT client. The Keep Alive timer values is fixed by MQTT when establishing the session. It is an unsigned short value, a value of zero (0) means that the client is never disconnected (no timeout).

Server KeepAlive minimum

The minimum value for the Keep Alive timer. If the value of the Keep Alive timer sent by the client is less than this value, then it is set to this minimum value. By default, set to 0 (no timeout).

```
<property name="com.scalagent.jorammq.mqtt.serverKeepAlive.min" value="0" />
```

Server KeepAlive Maximum

The maximum value for the Keep Alive timer. If the value of the Keep Alive timer sent by the client is greater than this value, then it is set to this maximum value. By default, set to 65535.

```
<property name="com.scalagent.jorammq.mqtt.serverKeepAlive.max" value="65535" />
```

5.3.5 Queued QoS 0 flag

Flag enabling to queue messages whose QoS level is 0 (at delivery):

- true means that QoS level 0 messages are enqueued;
- false means that they are either delivered immediately or dropped if the QoS0 consumer window is full (see 5.3.6)

```
<property name="com.scalagent.jorammq.mqtt.adapter.queuedQos0" value="false" />
```

Short name:

```
q0
```

Default value is false.

5.3.6 QoS 0 consumer window size

Maximum number of QoS 0 message payload bytes allowed to be buffered for a given MQTT subscriber client before dropping messages.

This limit is only valid for messages that are delivered at QoS level 0 and if the Queued QoS 0 flag is false (QoS 0 messages are not enqueued).

The number of payload bytes is incremented when a Publish message is delivered to the subscriber (waiting to be sent through the connection) and decremented when the message has been sent through the connection.

Above the limit, newly arrived messages are dropped.

```
  <property name="com.scalagent.jorammq.mqtt.adapter.qos0WindowSize"
    value="16777216" />
```

Short name:

```
qos0.win
```

5.3.7 Client Receive Maximum

This property replaces the deprecated property Inflight Message Count.

This property fixes the maximum number of Publish messages in transit between the broker and the client. It limits the number of QoS 1 and QoS 2 publications that it is willing to process concurrently. There is no mechanism to limit the QoS 0 publications that the Server might try to send.

Above the limit, messages are not sent any more to the MQTT connection (flow control is enabled). As soon as the number of messages goes below half the limit, messages are sent (flow control is disabled).

This property competes with the ReceiveMaximum property defined by the MQTT v5 specification (section 3.1.2.11.3). If the MQTT client specifies a value in the Connect packet it overrides the value defined in the broker configuration[14].

The default value is 65536, the values allowed by the specification are between 0 (excluded) and 65535 (included).

```
<property name="com.scalagent.jorammq.mqtt.adapter.clientReceiveMaximum"
          value="32768" />
```

Short name:

```
client.mx.inflight
```

[14] In particular, this value is used with MQTT v3 clients.

5.3.8 Broker Receive Maximum

This property defines the maximum number of Publish messages allowed to be in transit between the client and the broker. The Broker uses this value to limit the number of QoS 1 and QoS 2 publications that it is willing to process concurrently for the Client. It does not provide a mechanism to limit the QoS 0 publications that the Client might try to send.

The default value is 65536, the values allowed by the specification are between 0 (excluded) and 65535 (included).

```
<property name="com.scalagent.jorammq.mqtt.adapter.brokerReceiveMaximum"
          value="32768" />
```

Short name:

```
broker.mx.inflight
```

5.3.9 Max queued count

Maximum number of messages that are allowed to be enqueued for a given MQTT client (i.e. subscriber). Above this limit, if 'Discard old' is false, then newly arrived messages are rejected, otherwise old messages are discarded.

This limit is for messages delivered at QoS level 1, 2, or 0 if 'Queued QoS 0 flag' is true.

Set to 0 for no maximum.

```
<property name="com.scalagent.jorammq.mqtt.adapter.maxQueuedCount"
    value="0" />
```

Short name:

```
mx.q
```

5.3.10 Discard old

When the maximum number of messages allowed to be enqueued is reached, this flag states whether the old messages are discarded (flag is true), or the new messages are rejected (flag is false).

Default value is false (reject new messages).

```
<property name="com.scalagent.jorammq.mqtt.adapter.discardOld" value="false" />
```

Short name:

```
discard.old
```

5.3.11 Access control enabled

Enables authentication, i.e. requires user credentials to be passed when connecting.

If set to false, then any client is allowed to connect regardless of its credentials.

```
<property name="com.scalagent.jorammq.mqtt.adapter.accessControlEnabled"
          value="false" />
```

Short name:

```
actl
```

5.3.12 Client authentication

Sets how the client authentication is done by the server. Three modes are available:

1. NEED: client authentication is required, anonymous clients are disconnected.
2. WANT: request a client authentication (default value), but allow anonymous clients to connect.
3. NONE: no client authentication is desired.

```
<property name="com.scalagent.jorammq.mqtt.adapter.clientAuth" value="NONE" />
```

Short name:

```
client.auth
```

5.3.13 TCP backlog

Sets the TCP backlog parameter as specified by the Java class “java.net.ServerSocket”.

You should check the meaning of the parameter 'backlog' on your system. On Linux, it is the queue length for completely established sockets waiting to be accepted.

The property is valid for every protocol below MQTT: tcp, ssl, tls, ws, wss.

Default value is 128.

```
<property name="com.scalagent.jorammq.mqtt.adapter.tcpBacklog" value="128" />
```

Short name:

```
tcp.backlog
```

5.3.14 TCP keepalive

Sets the TCP keepalive parameter as specified by the Java class “java.net.Socket”.

The property is only valid for the following protocols supporting MQTT: tcp, ssl, tls.

Default value is true.

```
<property name="com.scalagent.jorammq.mqtt.adapter.tcpKeepAlive" value="true" />
```

Short name:

```
tcp.keepalive
```

5.3.15 TCP Connection timeout

The TCP Connection timeout, measured in milliseconds, defines the maximum time interval between the TCP connection and the MQTT CONNECT message received from a MQTT client. The TCP Connection timeout is an unsigned int value, a value of zero (0) means that the client is never disconnected (no timeout). By default, set to 15000 (15 seconds).

```
<property name="com.scalagent.jorammq.mqtt.serverConnectionTimeout"
          value="15000" />
```

5.3.16 Maximum number of pending connections

Defines the maximum number of pending connections (a pending connection is an incoming connection for which the MQTT handshake is not yet effective). When this number is reached incoming connections are refused. By default, set to 100.

This mechanism makes it possible to prevent denial of service attacks by limiting the impact of a large number of incoming connections on the resources of the broker.

```
<property name="com.scalagent.jorammq.mqtt.serverMaxPendingConnections"
          value="100" />
```

To avoid clogging log files and limit resource consumption, each connection reject is not signaled individually. A property allows you to specify the number of successive refusals resulting in a report in the logs (by default, 1000). The first reject is systematically reported.

```
<property name="com.scalagent.jorammq.mqtt.serverPendingConnectionsWarnThreshold"
          value="1000" />
```

5.3.17 Max read rate

Maximum read (inbound) rate in bytes per second.

The property is only valid for the following protocols below MQTT: tcp, ssl, tls.

Default value is 0 (rate control is disabled).

```
<property name="com.scalagent.jorammq.mqtt.adapter.maxReadRate" value="0" />
```

Short name:

```
mx.read.rate
```

5.3.18 Max write rate

Maximum write (outbound) rate in bytes per second.

The property is only valid for the following protocols below MQTT: tcp, ssl, tls.

Default value is 0 (rate control is disabled).

```
<property name="com.scalagent.jorammq.mqtt.adapter.maxWriteRate" value="0" />
```

Short name:

```
mx.write.rate
```

5.3.19 Message priority

Priority level assigned to the MQTT messages[15]. Allowed priority levels are integers starting from 0 to 9. Messages with the highest level are delivered first.

Default value is 4.

```
<property name="com.scalagent.jorammq.mqtt.adapter.defaultPriority" value="4" />
```

Short name:

```
priority
```

5.3.20 Replicate subscriptions

Enable subscriptions to be replicated in a cluster of MQTT servers. Depending on the MQTT connector it may be useful to enable or not the replication of subscriptions (see the example in section 18.5).

Default value is true.

```
<property name="com.scalagent.jorammq.mqtt.adapter.replicateSubscriptions"
  value="true" />
```

Short name:

```
rep.sub
```

5.3.21 MQTT server Keystore

Every JoramMQ server owns a unique Keystore which is a Java concept. A Keystore contains:

- cryptographic private keys, accompanied by a certificate chain for the corresponding public key; this content is used by the KeyManager to authenticate the local server to its peer (the MQTT client).
- trusted certificates whose owner (identified by its public key) is trusted by the keystore; this content is used by the TrustManager to authenticate the remote MQTT client to the local server.

When a TLS/SSL connection is opened the Keystore is used to select the best private key according to the secure socket negotiations (key algorithm, acceptable certificate authorities) and also to decide whether credentials presented by the MQTT client should be accepted.

For a more detailed definition of a Keystore, see the Java documentation of the class “java.security.KeyStore”. This Keystore can be created and managed with the Java command “keytool”.

[15]Message priority is a feature that is not provided by MQTT 3.1.1. However it should be provided in a next version of MQTT. Meanwhile JoramMQ allows to set the priority of the messages at the connector level.

The location of the keystore, its type, and the password to read/write are specified in conf/jorammq.xml by the following properties:

```
<property name="org.objectweb.joram.keystore" value="./conf/keystore" />
<property name="org.objectweb.joram.keystorepass" value="jorampass" />
<property name="org.objectweb.joram.keystoretype" value="JKS" />
```

All the server certificates (i.e. private keys and their associated certificate chain) that are registered in the Keystore must be protected with the same password. This password is generally the same as the password of the Keystore itself, which is used by default. However it is possible, if necessary, to declare a password for the server certificates that differs from the password of the Keystore. You may use the keypass property to that end.

```
<property name="org.objectweb.joram.keypass" value="jorampass" />
```

It is also possible to separate the truststore from the keystore. This is specified by defining three other properties:

```
<property name="org.objectweb.joram.truststore" value="./conf/truststore" />
<property name="org.objectweb.joram.truststorepass" value="jorampass" />
<property name="org.objectweb.joram.truststoretype" value="JKS" />
```

Keystore update

When the validity of a certificate is about to expire, it is necessary to add a new valid certificate to the keystore. However the broker must be stopped and restarted for this change to take effect.

Keystore checking

The broker performs some basic sanitary checks over the content of the keystore. It first occurs when the broker starts, and then when certificates are about to expire. The output of this checking is done using the logging system. It is controlled by the logger named com.scalagent.jorammq.-mqtt.adapter.KeyStoreStatus.

The broker main keystore must contain a valid private key and its associated certificate. If none can be found, an error log is issued. It should also contain a valid certificate of a trusted authority. This certificate is necessary when the broker requires client authentication. If none can be found, a warning log is issued.

The broker may also issue warning and error logs when certificates are about to expire. The notice periods are configured in the jorammq.xml file by two properties, whose syntax and default values are detailed below:

```
<property name="com.scalagent.jorammq.mqtt.adapter.securityKeyWarningDelay"
value="30" />
<property name="com.scalagent.jorammq.mqtt.adapter.securityKeyErrorDelay"
value="3" />
```

The properties set the number of days before the expiration date of a certificate when a warning and then an error log will be issued. The reserved value of -1 invalidates the checking.

6 MQTT server

This chapter covers the configuration of the server structure hosting the MQTT connector. These are advanced functions that must be handled only by experienced users.

6.1 Data storage

The storage module is one of the key components of the MQTT server, which is essential in particular for reliability and performance aspects.

Data storage operations are done in a transactional way ensuring that data are always consistent even after a server failure or system crash.

The way the data storage behaves can be customized through several properties explained below.

6.1.1 Storage class

The storage class has to implement the Java interface 'Transaction' defined by Joram. The Javadoc can be found at:

```
http://joram.ow2.org/current/javadoc/fr/dyade/aaa/util/Transaction.html
```

The name of the class is configured in conf/jorammq.xml by the following property:

```
<property name="Transaction" value="com.scalagent.txlog.TxLogTransaction" />
```

The storage class used by default is TxLogTransaction. TxLog is a transactional monitor optimized for messaging purpose, allowing to quickly save and delete persistent messages. Transaction logs are handled efficiently with mechanisms such as sequential writing, asynchronous fsync with callbacks, asynchronous compacting, and garbage of old messages still enqueued in a data store (also called repository).

6.1.2 Storage location

Persistent data are stored in a directory configured in bin/jorammq-server by a Java environment property:

```
-Dfr.dyade.aaa.agent.AgentServer.storage=$DATA_DIR/jorammq/s$SERVER_ID
```

Persistent data include MQTT messages delivered at QoS level 1 or 2, and with a clean session flag equal to false.

6.1.3 Log file size

Size of a log file in MB.

```
<property name="Transaction.FileSize" value="10" />
```

6.1.4 Minimum number of log files

Minimum number of log files used by the storage module.

```
<property name="Transaction.MinFileCount" value="10" />
```

6.1.5 Compacting criteria

Number of log files to be filled before compacting. Should be lesser than the minimum number of log files.

```
<property name="Transaction.MinCompactFileCount" value="5" />
```

6.1.6 Log garbage collection

Number of times an object has been compacted in the log before going in the repository (database).

```
<property name="Transaction.CompactCountThreshold" value="1" />
```

6.1.7 Synchronous compaction flag

Asserts whether the log compaction is done synchronously (stopping the storage activities) or not (concurrently with the storage activities).

```
<property name="Transaction.SynchronousCompact" value="false" />
```

6.1.8 Disk Synchronization

Asserts whether a disk synchronization (fsync) is done after a write operation in a log file or in the repository. Should be 'true' if no data, in particular no message, should be lost even in case of a system crash. Default value is 'true'.

If enabled then 'AsyncWrite' should also be enabled in order to reduce the overhead caused by the disk synchronization (see 6.1.9).

```
<property name="Transaction.SyncOnWrite" value="true" />
```

6.1.9 Asynchronous write

Asserts whether write operations are done synchronously (stopping the storage activities) or not (concurrently with the storage activities).

'AsyncWrite' does not lower the reliability level provided by 'SyncOnWrite'. 'AsyncWrite' allows to synchronize (fsync) only once for a group of write operations. The disk synchronization completeness is notified by a callback mechanism. For example, the disk synchronizations required by several concurrent message producers can be done by a single disk synchronization. And the producers do not return until the synchronization (fsync) has been completed successfully.

When this parameter is set (true), it allows to factorize the write operations on disk and therefore to reduce the overall cost of these. This has a very positive impact on scalability when the number of connections is high, and increases the overall message throughput.

Default value is 'true'.

```
<property name="Transaction.AsyncWrite" value="true" />
```

If the number of connections is not very large, it is preferable to unset this parameter (false). This will have a good effect on latency and message throughput for each connection.

6.1.10 Buffer size

Size in bytes of the buffer used when 'AsyncWrite' is enabled. The storage activities write data in this buffer, register callbacks (to guaranty the reliability) and continue. The real write operation to disk, and callback activations are done concurrently (see 6.1.9).

```
<property name="Transaction.BufferSize" value="4194304" />
```

6.1.11 Flush timeout

Duration in nanoseconds before a flush to disk is done when 'AsyncWrite' is enabled.

This duration is checked by an asynchronous activity in charge of writing the content of the buffer to disk, calling the disk synchronization primitive (fsync) and activating the callbacks.

```
<property name="Transaction.BufferTimeout" value="30000000" />
```

6.1.12 Use sleep

Asserts whether the flush timeout should be done using the primitive 'sleep' or not. This property should always be true as it is automatically disabled if the primitive 'sleep' does not work properly.

```
<property name="Transaction.UseSleep" value="true" />
```

6.1.13 Use NIO file

Asserts whether the NIO API should be used for handling files. Should be true on Linux (NIO is more efficient) and false on Windows (NIO is less efficient).

Default value is 'true'.

```
<property name="Transaction.UseNioFileChannel" value="true" />
```

6.1.14 Repository class

Class of the module responsible for storing data in a repository (database). Data are stored in the repository after several compacting stages (see 6.1.6).

The default repository is based on LevelDB.

```
  <property name="Transaction.RepositoryImpl"
    value="com.scalagent.txlog.LevelDBRepository" />
```

6.1.15 Use lock file

Asserts whether a lock file should be used to prevent several MQTT servers from writing in the same storage location.

```
<property name="Transaction.UseLockFile" value="true"/>
```

6.2 Engine

The Engine is a key component of the MQTT server, responsible for the execution of the core activities of the server, e.g. queues, subscription matching, message routing. Activities are executed by agents which are similar to actors sending events to each other and reacting to these events. A reaction is simply a call to an agent's method that processes an event. The Engine executes the reactions either in a single-threaded or multi-threaded way. The multi-threaded way ensures that each agent is executed by a single thread.

6.2.1 Engine class

The Engine has to implement the Java interface 'AgentEngine' defined by Joram. The Javadoc can be found at:

```
http://joram.ow2.org/current/javadoc/fr/dyade/aaa/agent/AgentEngine.html
```

The name of the class is configured in conf/jorammq.xml by the following property:

```
<property name="Engine" value="com.scalagent.batchengine.BatchEngine" />
```

The Engine used by default is called BatchEngine. BatchEngine boosts the performance by committing transactions in batches. The performance gains are maximum if BatchEngine works with the storage module TxLog.

6.2.2 Maximum number of serialized reactions

Maximum number of reactions that are allowed to be serialized in a single transaction.

If the number of storage operations waiting to be committed is greater than MaxSerialReactions then a commit is done even if the number of serialized reactions is less than MaxSerialReactions.

```
<property name="Engine.MaxSerialReactions" value="10000" />
```

6.3 MQTT message swap

Maximum number of bytes allowed for the enqueued MQTT messages that are stored in memory.

Above this limit, MQTT messages are swapped to disk.

```
<property name="com.scalagent.jorammq.mqtt.adapter.maxMessageTableSize"
          value="16777216" />
```

6.4 Max message size

Maximum payload size in bytes for incoming MQTT messages.

Above this limit, messages are dropped.

```
<property name="com.scalagent.jorammq.mqtt.adapter.maxMessageSize"
          value="1048576" />
```

6.5 Global flow control for publishers

Maximum number of events waiting to be processed by the Engine.

Useful when many MQTT clients are publishing concurrently as the flow control is done per publisher (and not globally for all the publishers). This limit enables the MQTT server to slow down the publishers when the global amount of work, called the “engine load”, is too high.

```
<property name="com.scalagent.jorammq.mqtt.adapter.maxEngineLoad"
          value="10000" />
```

7 MQTT shared subscriptions

MQTT shared subscriptions allow a set of MQTT clients to share the consumption of messages produced in response to a subscription. The set of subscribers is created by the use of a common share name. Shared subscriptions provide a facility similar to point to point messaging in that the processing of messages is not limited to the availability or capacity of a single consumer, the share name is analogous to the queue name in that it names the list of messages to be processed.

7.1 Create a shared subscription

A shared subscription is defined by the following topic filter syntax:

```
$share/<sharedSubscriptionName>/<topicFilter>
```

This specialised topic filter takes the place of the normal topic filter in the subscribe packet.

The "$share" prefix uses a reserved $ topic name and will therefore not cause a conflict with other topic filters.

'sharedSubscriptionName' is a string which labels the shared subscription and must not contain any "/" characters.

'topicFilter' follows the standard topic filter rules.

Subscribing clients will be a member of the share only if both the sharedSubscriptionName and topicFilter are identical. So, a different sharedSubscriptionName with the same topicFiler will create a different share. Similarly a sharedSubscriptionName followed by a different topic filter will create separate share.

The subscribing clients may make both non durable and durable subscriptions by setting clean session true or false. However they must set the clean session to the same value (true or false). A shared subscription cannot be durable and non durable at the same time.

The clients must also use the same QoS level when subscribing.

The maximum number of shared subscriptions per client is 256.

7.2 Delete a shared subscription

The shared subscription ceases to exist if the number of clients subscribed to it reaches zero. If the shared subscription ceases to exist the messages queued for processing are deleted by the server as with a non shared subscription.

7.3 Flow control and load balancing

Messages are delivered at a rate that depends on the speed of the consumers, i.e. a slow consumer receives less messages than a fast consumer.

The message flow control is done according to the following parameters:

- Consumer window size (see 5.3.3)
- Inflight message count (see 5.3.8)

An additionnal system property allows to distribute messages from shared subscription to connected client that does not have too many messages waiting (enqueued) even if it is not delivering (queuing):

- “com.scalagent.jorammq.mqtt.sharedSubscription.optimizeDelivery”, by default false.

7.4 Disconnected consumer

If a subscriber shares a subscription and is disconnected, these waiting messages are reallocated to another connected subscriber. If there is no available subscriber the messages remain in the shared subscription.

For messages delivered to the client but not yet acknowledged, the behavior depends on the QoS.

Messages delivered at QoS 0 cannot be reallocated because there is no acknowledgement returned by the subscriber. Therefore, some messages delivered at QoS 0 can be lost during the disconnection.

Messages delivered at QoS 2 are never reallocated because of the “exactly once” semantic that requires the messages to be redelivered to the same subscriber. The Server will complete the delivery of the message to that client when it reconnects.

By default, messages delivered at QoS 1 that are not acknowledged are not reallocated to another subscriber and remain in the session. The Server will complete the delivery of the message to that client when it reconnects. However a system property allows to optionally reallocate delivered messages from shared subscription of a disconnected client to other connected client:

- “com.scalagent.jorammq.mqtt.sharedSubscription.reallocateDelivered”, by default false.

7.5 How to make a message queue with MQTT

An interesting usage of shared subscriptions is to benefit from the message queue pattern with MQTT. This pattern is not directly provided by MQTT. But it can be built on top of the pattern publish/subscribe thanks to the shared subscription and without any overhead.

In order to make a queue, you just need to define the name of the topic that routes the messages to the queue and the name of the shared subscription used to consume messages from the queue.

Moreover if reliability is needed then the QoS level should be 1 (at least once) and the clean session flag of the consumers should be false.

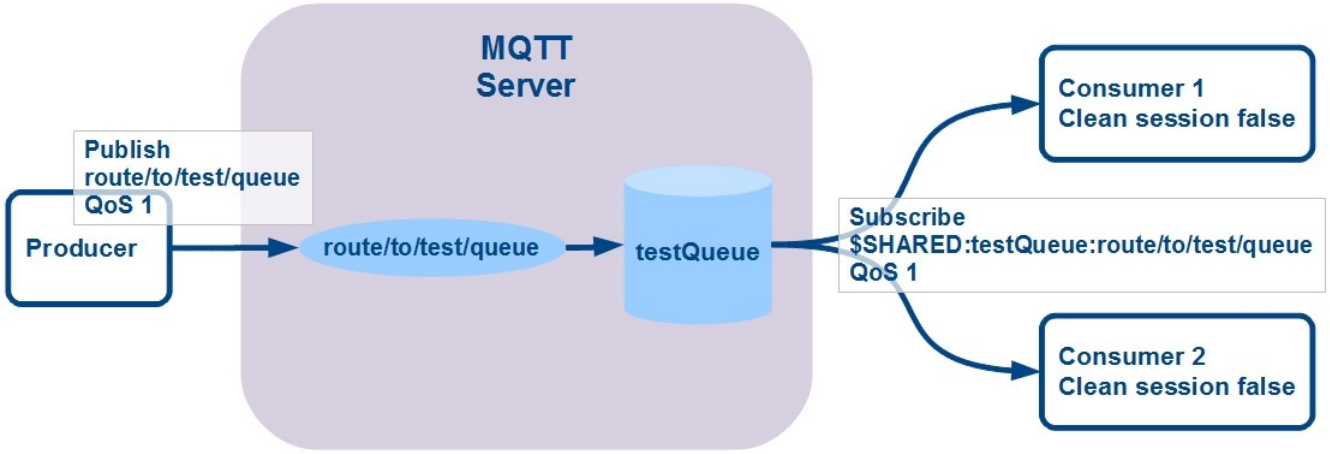

Figure 1: Interacting through a message queue with MQTT

Figure 1 illustrates an application where 3 MQTT clients interact through a message queue. Two clients consume messages from the queue. The third client produces messages to the queue. The name of the shared subscription is “testQueue” (the subscription is the queue). The topic to route messages to “testQueue” is “route/to/test/queue”.

The following Java pseudo-code creates the MQTT client called 'Consumer 1' and the queue 'testQueue'. This code has to be updated according to your MQTT client library.

```
MqttClient consumer1 = new MqttClient();
consumer1.setCleanSession(false);
consumer1.connect();
// Subscribe to topic “route/to/test/queue”
// with the shared subscription “testQueue”
// and the QoS level 1 (at least once)
consumer1.subscribe(“$share/testQueue/route/to/test/queue”, 1);
```

The second client called 'Consumer 2' also receives the messages from 'testQueue'. In this way, the message delivery is load-balanced across two consumers.

```
MqttClient consumer2 = new MqttClient();
consumer2.setCleanSession(false);
consumer2.connect();
consumer2.subscribe(“$share/testQueue/route/to/test/queue”, 1);
```

Then a third client called 'Producer' can send a message to 'testQueue' as follows:

```
MqttClient producer = new MqttClient();
producer.connect();
// Set the QoS level to 1 (at least once)
message.setQos(1);
```

```
// Publish the message to topic "route/to/test/queue"
producer.publish("route/to/test/queue", message);
```

8 JMX MBeans

8.1 Introduction

Java Management Extensions (JMX) is a Java technology that supplies tools for managing and monitoring applications. Each resource is represented by an object called MBean (for Managed Bean).

JMX is the heart of the administration and monitoring of JoramMQ:

- All administration functions and metrics of JoramMQ are implemented through MBeans. This chapter describes all generic JoramMQ MBeans.
- All JoramMQ administration and monitoring tools use the JMX API to interact with JoramMQ. There are generic JMX console[16] (see section 8.1.2 below), MQTT monitoring (see chapter 9), SSH commands (see chapter 10), Nagios plugins (see chapter 11), and HawtIO console (see chapter 23).
  The JoramMQ JMX Rest API described in section 8.11 allows a direct access to MBean through a Web browser.

8.1.1 JoramMQ MBeans

Fives categories of MBeans are provided by a JoramMQ server in the namespace “com.scalagent.jorammq”:

1. AccessControl: 1 MBean
2. ClientManager: 1 MBean
   - 1 MBean per MQTT client
   - 1 MBean per MQTT shared subscription
3. ConnectorManager: 1 MBean
   - 1 MBean per MQTT connector
4. ConnectionManager
   - 1 MBean per MQTT connection
5. SubscriptionManager
   - 1 MBean per subscription context
     - 1 MBean per topic level

[16] Jconsole, JvisualVM or JMC (Java Mission Control) for example.

Some components of JoramMQ can be deployed in very large numbers (clients, connections, etc.). To prevent scalability problems in JMX monitoring mechanisms, the corresponding MBeans are not automatically enabled. The methods for specifically enabling (resp. disabling) these MBeans are described in the corresponding sections below.

Other MBeans can be provided by additional components as health-check (chapter 12), MQTT-SN gateway (chapter 13) or bridges (chapters 14, 15, 16 or 17).

8.1.2 JMX console

The JMX entry point is configured in the command file 'bin/jorammq-server' by the following Java environment properties:

```
-Dcom.sun.management.jmxremote.port=3333
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote
```

By default, listening port 3333 is used and security is disabled.

8.2 AccessControl

Gives global indicators about access control of the MQTT server.

8.2.1 Attributes

Attribute	Type	Description
Initialized	boolean	True if the component is active, false otherwise.
PasswordFilePath	String	Path of the file containing user definitions.
AclFilePath	String	Path of the file containing access control rules.
UpdatePeriod	int	Defines the checking period (in ms) for files containing user definitions and access control rules. By default, 30.000ms.

8.2.2 Operations

Currently there is no operation defined by the default Access Control component.

8.3 ClientManager

Gives global indicators for all the MQTT clients of the server.

8.3.1 Attributes

Attribute	Type	Description
AckCount	int	Number of acknowledgements received from all the MQTT clients (subscribers) since the server is running
ConnectCount	int	Number of Connect done by all the MQTT clients (subscribers) since last server boot.
ConsumedMemory	int	Number of bytes consumed by the payloads of the enqueued messages in memory (QoS 1, 2 and 0 if queuing enabled for QoS 0)
DisconnectCount	Int	Number of Disconnect done by all the MQTT clients (subscribers) since last server boot.
GarbageCount	int	Number of reactions executed to garbage (i.e. delete) messages owned by a subscription queue that is purged
LoadedClientCount	int	Number of clients currently loaded in the MQTT server
LoadedShared SubscriptionCount	int	Number of shared subscriptions currently loaded in the MQTT server
MaxNotAckedSize	int	Compares all the values of the attribute: NotAckedSize for all the MQTT clients that are loaded in the server and return the maximum value.
MaxToDeliverMessage Count	int	Compares the values of the attribute ToDeliverMessageCount for all the MQTT clients that are loaded in the server and return the maximum value.
MessageCounter	int	Number of messages that have been received and processed by this MQTT server. If a message is delivered several times, then it is counted only once.
MessageDeleteCount	int	Number of messages that have been received and deleted after acknowledgement. If a message is delivered several times, then it is deleted only once. Only QoS 1 and 2 messages received with the clean session flag set to false are deleted after acknowledgement.
MessageSaveCount	int	Number of messages that have been received and saved before delivery. If a message is delivered several times, then it is saved only once. Only QoS 1 and 2 messages received with the clean session flag set to false are saved before delivery.
MessageTableSize	int	Number of enqueued messages in memory, i.e. messages delivered at QoS 1, 2 and 0 if queuing enabled for QoS 0. If the client is disconnected and clean session is false, then messages are not kept in memory.
PublishDeliverCount	int	Number of Publish messages that have been delivered. If a message is delivered several times, then it is counted only once.

SwappedLoadedCount	int	Number of swapped messages that have been re-loaded from disk
SwappedMessageCount	int	Number of messages that have been swapped to disk
SwappedToDeleteCount	int	Number of swapped messages that have been deleted
TriggerDeliveryCount	int	Number of reactions executed to deliver MQTT messages. A delivery is triggered only if a client is connected. Several messages may be delivered during a unique reaction.

8.3.2 Operations

Operation	Parameter(s)	Description
enableClientMBean	ClientId	Enable the MBean of the specified MQTT client
disableClientMBean	ClientId	Disable the MBean of the specified MQTT client
enableShared SubscriptionMBean	Shared subscription name	Enable the MBean of the specified MQTT shared subscription
disableShared SubscriptionMBean	Shared subscription name	Disable the MBean of the specified MQTT shared subscription
dumpClients	-	Return a brief description of every active MQTT client in the logs
logClients	-	Print a brief description of every active MQTT client in the logs
dumpDurableClientIds	-	Return a list of the durable clientIds
logDurableClientIds	-	Print a list of the durable clientIds in the logs
checkToDeliverMessageCount	threshold	Lists all clients with more than '*threshold*' messages waiting to be delivered.
checkNotAckedSize	threshold	Lists all clients for which the total payload size of the unacknowledged messages exceeds the '*threshold*' parameter.
checkDisconnectionTime	threshold	Returns the list all MQTT sessions disconnected for at least threshold seconds.
dumpClient	ClientIds	Returns the state of the specified MQTT clients. The parameter can be a unique client identifier or a comma separated list. If empty all MQTT clients are dumped.
cleanSession	clientIds, and silence	Delete the specified clientIds context. True for silence.
cleanOlderSession	Duration in second and ignore regExp and silence	Delete all clientIds context older than the duration. Use a regExp to ignore some clientIds. True for silence.

8.4 MQTT client

Gives indicators about a single MQTT client.

It has to be enabled explicitly by calling the operation 'enableClientMBean' provided by ClientManager.

8.4.1 Attributes

Attribute	Type	Description
AckCount	int	Number of acknowledgements received from this MQTT client (subscriber) since the server is running.
Clean	boolean	Clean session flag.
ClientId	String	MQTT Client Id.
ConnectCount	int	Number of times the client has connected since the server has been started.
Connected	boolean	Asserts whether the client is connected or not.
DeliveredMessageCount	int	Number of messages delivered since the client is connected.
DeliveredTransientMessageCount	int	Number of QoS0 messages delivered since the client is connected.
DeliveredReliableMessageCount	int	Number of reliable messages (QoS 1 or 2) delivered since the client is connected.
DroppedMessageCount	int	Number of messages that have been dropped by the server. Messages can be dropped because the maximum number of messages allowed to be enqueued is reached (see section 5.3.9). Messages delivered at QoS 0 to a disconnected client are dropped, except if the property 'Queued QoS 0' is enabled (see 5.3.5).
LastCnx	String	Duration since the last client connections, ex: xxHyyMzzS or connected
MqttMsgIdCounter	int	Counter used to allocate an MQTT identifier to the delivered messages
NotAckedMessageCount	int	Number of messages waiting to be acknowledged
NotAckedSize	int	Total payload size of the unacknowledged messages
Queuing	boolean	Asserts whether the subscriber is overflowed or not
QueuingCount	int	Count the number of times a subscriber has been overflowed since the MQTT server has been started
QueuingCountPerMin	int	Number of times a subscriber has been overflowed during a minute Calculated from the value obtained by the last invocation of the command if the delay is at least one minute

SessionId	int	Identifier of the current session (incremented each time the client reconnects)
ToDeliverMessageCount	int	Number of messages waiting to be delivered
RemoteAdress	String	The network address of client, i.e. hostname or IP address, and port.

8.4.2 Operations

Operation	Parameter(s)	Description
purge	-	Purge all the messages waiting to be delivered
dumpNotAcked	-	Print in the logs the description of every message waiting to be acknowledged

8.5 MQTT shared subscription

Gives indicators about a single shared subscription.

It has to be enabled explicitly by calling the operation 'enableSharedSubscriptionMBean' provided by ClientManager.

8.5.1 Attributes

Attribute	Type	Description
ClientCount	int	Number of clients sharing this subscription
Clean	boolean	Clean session flag set by the clients sharing this subscription. All the clients must have the same clean session flag.
Name	String	Name of the shared subscription
SubscriptionQos	int	QoS level set by the clients sharing this subscription. All the clients must have the same QoS level.
ToDeliverMessageCount	int	Number of messages waiting to be delivered

8.5.2 Operations

Operation	Parameter(s)	Description
listClients	-	Return the description of every client sharing this subscription

8.6 ConnectorManager

Gives global indicators about the connectors.

8.6.1 Attributes

Attribute	Type	Description
AccessControlEnabled	boolean	Asserts whether the access control is enabled or not
ClientsDisconnected	int	$SYS topic: the total number of persistent clients (with clean session disabled) that are registered at the broker but are currently disconnected. $SYS/broker/clients/disconnected
ConnectionCount	int	$SYS topic: the number of currently connected clients $SYS/broker/clients/connected
LoadBytesReceived	long	$SYS topic: the total number of bytes received since the broker started. $SYS/broker/load/bytes/received
LoadBytesSent	long	$SYS topic: the total number of bytes sent since the broker started. $SYS/broker/load/bytes/sent
MaxConsumerWindowSize	int	MQTT server property: 'maxConsumerWindowSize' com.scalagent.jorammq.mqtt.adapter.maxConsumerWindowSize
MaxEngineLoad	int	MQTT server property: 'maxEngineLoad' com.scalagent.jorammq.mqtt.adapter.maxEngineLoad
MaxMessageSize	int	MQTT server property: 'maxMessageSize' com.scalagent.jorammq.mqtt.adapter.maxMessageSize
MaxProducerWindowSize	int	MQTT server property: 'maxProducerWindowSize' com.scalagent.jorammq.mqtt.adapter.maxProducerWindowSize
MessagesDropped	long	$SYS topic: the total number of publish messages that have been dropped due to inflight/queuing limits. $SYS/broker/messages/publish/dropped
MessagesInflight	int	$SYS topic: the number of messages with QoS>0 that are awaiting acknowledgments. $SYS/broker/messages/inflight
MessagesPublishReceived	long	$SYS topic: the total number of PUBLISH messages received since the broker started. $SYS/broker/messages/publish/received:
MessagesPublishSent	long	$SYS topic: the total number of PUBLISH messages sent since the broker started. $SYS/broker/messages/publish/sent
MessagesRetainedCount	int	$SYS topic: the total number of retained messages active on the broker. $SYS/broker/messages/retained/count:
MessagesReceived	long	$SYS topic: the total number of MQTT frames of any type received since the broker started. $SYS/broker/messages/received

MessagesSent	long	$SYS topic: the total number of MQTT frames of any type sent since the broker started. $SYS/broker/messages/sent
PendingAttachTableSize	int	Number of MQTT clients with at least one pending session to be opened.
PendingSessionCount	int	Total number of sessions waiting to be opened.
PublishReceivedRate	int	Number of Publish messages received per second by the MQTT server. Enabled with $SYS topics
PublishSentRate	int	Number of Publish messages sent per second by the MQTT server. Enabled with $SYS topics
ReceiveBufferSize	int	MQTT server property: 'receiveBufferSize' com.scalagent.jorammq.mqtt.adapter.receiveBufferSize
SendBufferSize	int	MQTT server property: 'sendBufferSize' com.scalagent.jorammq.mqtt.adapter.sendBufferSize
SubscriptionCount	int	Total number of subscriptions stored by the MQTT server.
Swapping	boolean	Asserts whether the MQTT server is currently swapping messages to disk or not.

8.6.2 Operations

Operation	Parameter(s)	Description
enableConnectionMBean	ClientId	Enable the MBean of the specified MQTT connection
disableConnectionMBean	ClientId	Disable the MBean of the specified MQTT connection
logConnections	-	Print in the logs the description of all the connections
dumpConnections	-	Returns the description of all connections.
dumpConnection	ClientId	Returns the description of the specified connection.
closeConnection	ClientId	Closes the specified connection.
logTopicRepository	-	Print in the logs the description of all the topics
checkKeyStoreSanity	int, int	Performs sanitary checks over the content of the main adapter keystore. Returns a String. First param: max validity delay (days) used to issue WARN messages. 0: use configured value, -1: issue no WARN messages. Second param: max validity delay (days) used to issue ERROR messages. 0: use configured value, -1: issue no ERROR messages.
checkKeyStoreSanityStatus	int, int	Same as checkKeyStoreSanity but returns an int. OK(0), WARN(1), ERROR(2)

8.7 MQTT connector

Gives indicators related to each connector.

8.7.1 Attributes

Attribute	Type	Description
ClientAuth	String	Connector property: 'client-auth'
ConnectionCount	int	Number of connections opened by this connector
MaxConsumerWindowSize	int	Connector property: 'cons.win'
MaxProducerWindowSize	int	Connector property: 'prod.win'
ClientReceiveMaximum	int	Connector property: 'client.mx.inflight'
BrokerReceiveMaximum	int	Connector property: 'client.mx.inflight'
MaxQueuedCount	int	Connector property: 'mx.q'
Qos0WindowSize	int	Connector property: 'qos0.win'
QueuedQos0		Connector property: 'q0'
ReceiveBufferSize	int	Connector property: 'rcv.buf'
SendBufferSize	int	Connector property: 'snd.buf'
Priority	[0..9]	Default incoming message priority.
PendingConnections	int	Number of pending connections.
PendingClose	int	Number of pending connections closed before the connection was established.

8.7.2 Operations

Operation	Parameter(s)	Description
resume	-	Resume the specified MQTT connection
suspend	-	Suspend the specified MQTT connection

8.8 MQTT connection

Gives indicators related to each MQTT connection.

It has to be enabled explicitly by calling the operation 'enableConnectionMBean' provided by ConnectorManager.

8.8.1 Attributes

Attribute	Type	Description
ClientId	String	MQTT Client Id
Closed	boolean	Asserts whether the connection has been closed. A closed connection should remove its MBean. So the attribute 'Closed' should always be false except at the time the connection is being closed and before the MBean is removed.
CommandCounter	int	Number of MQTT commands received by this connection.
Dead	boolean	Asserts whether the connection has been killed internally, for example after a missed keep alive (not sent by the client). The connection is in the process of being closed.
Disconnected	boolean	Asserts whether the connection has been explicitly disconnected, i.e. after having received an MQTT command 'Disconnect'. The connection is in the process of being closed.
DroppedPublishCount	int	Number of QoS 0 Publish messages that have been dropped. QoS 0 Publish messages are dropped if the flag 'Queued Qos 0' is false.
FlowControlCounter	int	Number of times the connection has been overflowed by Publish messages received from the client (input messages).
Full	boolean	Asserts whether the transport output is full. If full, then no message can be sent to the client.
InFlightPublishCount	int	Number of in-flight messages which QoS level is either 1 or 2. QoS 0 messages are not counted.
InputOverflowed	boolean	Asserts whether the connection input is overflowed or not.
MaxConsumerWindowSize	int	Connector property: 'cons.win'
MaxInflightMessageCount	int	Connector property: 'mx.inflight'
MaxProducerWindowSize	int	Connector property: 'prod.win'
OutputCounter	int	Number of messages that have been sent to the client.
OutputOverflowSize	int	Total payload size of the Publish messages that could not be sent to the client because the transport output is full.
OutputOverflowed	boolean	Asserts whether the connection output is overflowed or not.

OverflowedSizeQos0	int	Total payload size of the enqueued QoS 0 Publish messages that could not be sent to the client because the transport output is full. QoS 0 Publish messages are not dropped if the flag 'Queued Qos 0' is true.
PingInputCounter	int	Number of ping received from the client
ProducerWindowSize	int	Total payload size of the Publish messages received from the client and being processed by the server. As soon as a message has been processed (e.g. enqueued and stored) its payload size is removed from the total.
PubackInputCounter	int	Number of Puback messages received from the client
PubcompInputCounter	int	Number of Pubcomp messages received from the client
PublishCount	int	Number of Publish messages received from the client
PubrecInputCounter	int	Number of Pubrec messages received from the client
Qos0WindowSize	int	Connector property: 'qos0.win'
QueuedQos0	boolean	Connector property: 'q0'
ReceiveBufferSize	int	Connector property: 'rcv.buf'
SendBufferSize	int	Connector property: 'snd.buf'
ServerUri	String	URI of the connector's entry point expressed as “protocol underneath MQTT” - “listening port”
SessionId	int	Identifier of the current session (incremented each time the client reconnects)
SubscriptionCount	int	Number of topic filters used as subscriptions by the client

8.8.2 Commands

Operation	Parameter(s)	Description
close	-	Close the connection
dumpInFlightPublish	-	Print in the logs the list of all the inflight messages
flush	-	Flushes the transport output
suspendRead	-	Stop reading input commands from the client
resumeRead	-	Start reading input commands from the client
unsubscribe	-	Unsubscribe from all the topics

8.9 MQTT subscription context

Gives indicators related to a subscription context.

8.9.1 Attributes

Attribute	Type	Description
ClusterSubscriptionCount	int	Number of subscriptions made by the MQTT servers of the cluster
ParentTopic	String	Identifier of the parent topic (i.e. parent subscription context) or null if no parent topic is defined
SubscriptionCount	int	Number of subscriptions in this subscription context
TopicCluster	String	List of the internal references of the clustered subscription contexts. Empty if this subscription context does not belong to a cluster.
TopicReference	String	Internal reference of this subscription context. Internal references are assigned by JoramMQ and allow several subscription contexts located in distinct MQTT servers to communicate with each other.

8.9.2 Operations

Operation	Parameter(s)	Description
displayRootLevel	-	Enables the MBeans for the root level of the subscription context, i.e. the root topic level. The MQTT topic root is named “<root>”. The MQTT topic level length must be greater than one character. However, it is possible to subscribe to “#” so the root level may be necessary to visualize.
dumpRetainedMessages	-	List all the retained messages
dumpSubscriptions	-	List all the subscriptions
getSubscriptions	ClientId	Returns the description of all subscriptions of the corresponding MQTT client in the selected subscription context.

8.10 MQTT topic level

Gives indicators related to a subscription context for a given topic level.

It has to be enabled explicitly by calling the operations:

- 'displayRootLevel' provided by an MQTT subscription context
- 'displaySubLevels' provided by the parent MQTT topic level

8.10.1 Attributes

Attribute	Type	Description
ClusterSize	int	Number of subscriptions made for this topic level by the MQTT servers of the cluster
MultiLevelClusterSize	int	Number of subscriptions made for '#' at this topic level by the MQTT servers of the cluster
MultiLevelSharedSubscriberList	String	List of the shared subscriptions made for '#'
MultiLevelSubscriberList	String	List of the subscriptions made for '#'
RemoteTopicRef	String	Reference of the remote subscription context that represents this topic level. Used in a distributed topic level tree when a remote subscription context represents a particular topic level.
SharedSubscriberList	String	List of the shared subscriptions made for this topic level
SubscriberList	String	List of the subscriptions made for this topic level

8.10.2 Operations

Operation	Parameter(s)	Description
DisplaySubLevels	-	Enables the MBeans for the sub levels of this topic level

8.11 JoramMQ JMX Rest API

The JoramMQ JMX Rest API allows you to expose the JMX Mbeans of the JoramMQ server over a simple Rest/HTTP interface. For example this allows any web capable device to get Mbean's attributes using a regular HTTP request.

8.11.1 Installation and configuration

The JoramMQ JMX Rest interface is normally declared in the default configuration.

Installation

The JoramMQ JMX Rest interface is installed as an OSGi bundle (joram-tools-rest-jmx.jar in the bundle directory), it depends on the HTTP service and JAX-RS service implemented by the Jetty and Jersey bundles. If needed you can add these bundles to the list of installed bundles as follows:

```
felix.auto.start.1= \
 file:./bundle/monolog.jar \
 …
 file:./bundle/jorammq-ssh.jar \
 file:./bundle/ow2-jms-2.0-spec.jar \
```

```
file:./bundle/ow2-jta-1.1-spec.jar \
file:./bundle/javax.annotation-api.jar \
file:./bundle/javax.inject.jar \
file:./bundle/javax.ws.rs-api.jar \
file:./bundle/validation-api.jar \
file:./bundle/jersey-container-servlet-core.jar \
file:./bundle/jersey-server.jar \
file:./bundle/jersey-guava.jar \
file:./bundle/hk2-api.jar \
file:./bundle/aopalliance-repackaged.jar \
file:./bundle/hk2-utils.jar \
file:./bundle/jersey-common.jar \
file:./bundle/hk2-locator.jar \
file:./bundle/osgi-resource-locator.jar \
file:./bundle/jersey-client.jar \
file:./bundle/javassist.jar \
file:./bundle/servlet.jar \
file:./bundle/jetty.jar \
file:./bundle/org.osgi.compendium.jar \
file:./bundle/jndi-client.jar \
file:./bundle/joram-client-jms.jar \
file:./bundle/gson.jar \
file:./bundle/joram-tools-rest-jmx.jar
```

Then you need to stop the server, delete Felix bundle cache 'data/felix/', and restart the server.

Configuration

The JoramMQ JMX Rest implementation does have some configuration options. These are configured via the Felix properties file in the conf directory.

Authentication

by default all users are allowed to connect to the JoramMQ JMX Rest component, by defining the properties below you can restrain its access:

- rest.jmx.root: defines the name of the allowed user (default “admin”).
- rest.jmx.password: defines the password of the allowed user (default “adminpass”).

Jetty configuration

The main Jetty properties are listed in the table below (see the Felix HTTP service documentation: https://github.com/apache/felix-dev/tree/master/http#configuration-properties).

Property	Description

org.apache.felix.http.enable	Flag to enable the use of HTTP. The default is true.
org.osgi.service.http.port	The port used for servlets and resources available via HTTP. The default is 8080. See port settings below for additional information. A negative port number has the same effect as setting org.apache.felix.http.enable to false.
org.apache.felix.https.enable	Flag to enable the use of HTTPS. The default is false.
org.osgi.service.http.port.secure	The port used for servlets and resources available via HTTPS. The default is 8443. See port settings below for additional information. A negative port number has the same effect as setting org.apache.felix.https.enable to false.
org.apache.felix.https.keystore	The name of the file containing the keystore.
org.apache.felix.https.keystore.password	The password for the keystore.

You may also have to specify other variables related to the use of Jetty and especially in a SSL context. Below is the format of the configuration file and the default values for each property.

```
################################################################
# Felix HTTP Service
################################################################
org.apache.felix.http.enable=true
org.osgi.service.http.port=8989
#org.apache.felix.https.enable=true
#org.osgi.service.http.port.secure=8443
#org.apache.felix.https.keystore=./conf/keystore
#org.apache.felix.https.keystore.password=jorampass
```

8.11.2 Usage

The JoramMQ JMX Rest interface publishes a variety of REST resources to perform various operations on JMX Mbeans. For each response there is a set of links allowing to access the additional information.

Authentication

If the JoramMQ JMX API is configured to require authentication (see section 8.11.1) you need to add an authorization header in the request. This property is an encoded string built from the user's name and password separated by a colon ':'.

For example to get the number of pending messages in the destination 'queue' with a curl request:

- curl --user "admin:admin" --get http://192.168.1.36:8989/joram/jmx/domains/Joram%230/Joram%230:name=queue,type=Destination/PendingMessageCount

List JMX domains

You get the list of all JMX domains by doing a simple GET on the following URI pattern:

- http://host:port/joram/jmx/domains

It returns the list of JMX domains either in HTML or JSON depending of the required document type.

List JMX object's names of a domain

You get the list of all JMX object's name of a domain by doing a simple GET on the following URI pattern:

- http://host:port/joram/jmx/domains/{domain}

It returns the list of all JMX object's name of the domain either in HTML or JSON depending of the required document type.

List attributes of a JMX Mbean

You get the list of all attributes of a particular MBean by doing a simple GET on the following URI pattern:

- http://host:port/joram/jmx/domains/{domain}/{object-name}

It returns the list of all attributes (name and value) of a particular MBean either in HTML or JSON depending of the required document type.

Get the value of an attribute.

You get the value of a particular attribute by doing a simple GET on the following URI pattern:

- http://host:port/joram/jmx/domains/{domain}/{object-name}/{attribute}

It returns the value of the specified attribute.

8.11.3 Use with an HTML browser

Since the protocol relies on a standard protocol (HTTP) and format (HTML/JSON), the JoramMQ JMX Rest interface can be even accessed from within the browser with a simple URL. The following URLs give a human readable view of the interface:

- http://host:port/joram/jmx
  - Give a browsable view of JMX Mbeans.
- http://host:port/joram/jmx/domains
  - Give the list of all JMX domains.
- http://host:port/joram/jmx/domains/{domain}
  - Give the list of all MBeans of the specified domain.

- http://host:port/joram/jmx/domains/{domain}/{object-name}
  - Shows all attributes of the MBean.

9 MQTT monitoring

An MQTT server can be monitored with the protocol MQTT in two ways. The first way is simply to subscribe to the predefined $SYS topics. The second way requires configuring the attributes of the MBeans that we want to publish.

Any MQTT client can be used as a monitoring tool, either standalone or web based (connected to the MQTT server with WebSocket).

9.1 $SYS topics

These topics are specified at https://github.com/mqtt/mqtt.github.io/wiki/SYS-Topics.

$SYS topics are regularly published after a time interval set in 'conf/jorammq.xml'. The time unit is second.

```
<property name="com.scalagent.jorammq.mqtt.adapter.sysTopicInterval"
    value="0" />
```

The value 0 disables the pre-defined $SYS topics.

9.2 JMX topics

JMX attributes can be regularly polled by a local monitoring module that publishes MQTT messages to some topics (e.g. $SYS topics).

The configuration is specified in 'conf/jorammq.xml' file.

```
<property name="com.scalagent.jorammq.mqtt.monitor.conf"
    value="./conf/monitor.xml" />
```

The configuration file contains a unique 'monitor' root element. The monitoring is activated if the attribute 'enabled' is set to 'true'. The default value is 'false'.

```
<monitor xmlns="http://www.scalagent.com/schema/MqttMonitorSchema"
         enabled="false">
...
```

9.2.1 Create a monitoring task

Several monitoring tasks can be executed by the monitor. Create a child element 'task' and specify its period (in seconds) and the root topic used to publish MQTT messages (by default '$SYS/monitor').

```
  <task period="10" topicPrefix="$SYS/monitor">
```

Potentially all existing MBeans of the JVM can be monitored using the ‘mbean’ element. However, there are shortcuts allowing easier access to the usual JoramMQ MBeans.:

- ConnectorManager, MQTT connectors and connections.
- ClientManager and MQTT clients.
- MQTT shared subscription.
- 'java.lang' MBeans

JMX attributes can be monitored by sending their raw value or a calculated rate. The rate is calculated with two successive raw values obtained in a period of time. Monitoring raw values and rates can be specified for every MBeans. The elements to declare are 'attribute' and 'rate':

```
      <attribute name="jmxAttribute1"/>
      <attribute name="jmxAttribute2"/>
      <rate name="jmxAttribute1"/>
      <rate name="jmxAttribute2"/>
```

These elements can be declared inside the elements defined hereafter.

All values and rates are published on the subtree of the root topic of the task. The final topic path depends of the MBean and attribute name.

9.2.2 ConnectorManager

An element 'connectorManager' has to be declared. Only one element 'connectorManager' can be defined in a task.

```
    <connectorManager>
      <attribute name="ConnectionCount"/>
      <attribute name="SubscriptionCount"/>
      <rate name="MessagesPublishReceived"/>
      <rate name="MessagesPublishSent"/>
    </connectorManager>
```

The full list of available JMX attributes can be found in section 8.6. Some of those attributes are enabled only if the $SYS topic is activated (see section 9.1). For example, if the $SYS topic is not enabled, then the values of the attributes MessagesPublishReceived and MessagesPublishSent remain at 0.

Values are published on the topic ‘<root>/connectorManager/<attribute name>’.

Rates are published on the topic ‘<root>/connectorManager/<attribute name>Rate’.

9.2.3 MQTT connector

An element 'connectors' allows to monitor a list of attributes owned by a list of MQTT connectors. Several elements 'connectors' can be defined in a task.

```
<connectors>
  <connector id="<protocol>-<port>"/>
  <connector id="tcp-1883"/>
  <attribute name="ConnectionCount"/>
</connectors>
```

The full list of available JMX attributes can be found in section 8.7.

Values are published on the topic '<root>/connector/<id>/<attribute name>'.

Rates are published on the topic '<root>/connector/<id>/<attribute name>Rate'.

9.2.4 MQTT connection

An element 'connections' allows to monitor a list of attributes owned by a list of MQTT connections. Several elements 'connections' can be defined in a task.

```
<connections>
  <connection id="<client id 1>"/>
  <connection id="<client id 2>"/>
  <attribute name="CommandCounter"/>
  <rate name="PublishCount"/>
</connections>
```

The full list of available JMX attributes can be found in section 8.8.

Values are published on the topic '<root>/connection/<id>/<attribute name>'.

Rates are published on the topic '<root>/connection/<id>/<attribute name>Rate'.

9.2.5 MQTT client

An element 'clients' allows to monitor a list of attributes owned by a list of MQTT clients. Several elements 'clients' can be defined in a task.

```
<clients>
  <client id="<client id 1>"/>
  <client id="<client id 2>"/>
  <attribute name="Connected"/>
  <attribute name="ToDeliverMessageCount"/>
  <rate name="DeliveredMessageCount"/>
</clients>
```

The full list of available JMX attributes can be found in section 8.4.

Values are published on the topic '<root>/client/<id>/<attribute name>'.

Rates are published on the topic '<root>/client/<id>/<attribute name>Rate'.

9.2.6 MQTT shared subscription

An element 'sharedSubscriptions' allows to monitor a list of attributes owned by a list of MQTT shared subscriptions. Several elements 'sharedSubscriptions' can be defined in a task.

```
    <sharedSubscriptions>
      <sharedSubscription id="<shared subscription name 1>:<topic filter 1>"/>
      <sharedSubscription id="<shared subscription name 2>:<topic filter 2>"/>
      <attribute name="ClientCount"/>
      <attribute name="ToDeliverMessageCount"/>
    </sharedSubscriptions>
```

The full list of available JMX attributes can be found in section 8.5.

Values are published on the topic '<root>/sharedSubscription/<id>/<attribute name>'.

Rates are published on the topic '<root>/sharedSubscription/<id>/<attribute name>Rate'.

As the name of a shared subscription can contain forbidden characters in the topic paths, these are automatically replaced: '#' is replaced by '*', and '+' is replaced by '.'.

9.2.7 'java.lang' MBeans

An element 'java' allows to monitor a list of attributes owned by a 'java.lang' MBean. Several elements 'java' can be defined in a task.

```
    <java type="OperatingSystem">
      <attribute name="ProcessCpuLoad"/>
    </java>
```

The full list of available JMX attributes depends on the JVM.

Values are published on the topic '<root>/java/<attribute name>'.

Rates are published on the topic '<root>/java/<attribute name>Rate'.

9.2.8 MBeans

Any MBean of the JVM can be monitored using element 'mbean'. Several elements 'mbean' can be defined in a task, each needs to specify the MBean object name.

```
    <mbean name="com.scalagent.jorammq:host=localhost,manager=Status,connector=tcp
-1883" id="status/tcp-1883">
```

```
      <attribute name="StatusInfo"/>
    </mbean>
```

The list of available JMX attributes depends of the MBean specification.

Values are published on the topic '<root>/<id>/<attribute name>'.

Rates are published on the topic '<root>/<id>/<attribute name>Rate'.

If the 'id' attribute of the 'mbean' element is not specified the MBean object name is used. As this name can contain forbidden characters for topic paths, these are automatically replaced: '#' is replaced by '*', and '+' is replaced by '.'.

10 Secure shell commands

This command line interface (CLI) is based on Apache Felix Gogo shell, it is installed by default. The new console provides:

- support for the ssh protocol,
- JAAS-based user authentication,
- improved tab completion.

The default authentication is password-based, but key-based authentication is also supported.

10.1 Installation and configuration

JoramMQ provides a secure shell (ssh) to monitor and control an MQTT server. The secure shell is configured in the file "conf/felix.properties". Two authentication mechanisms can be used: user/password and public key[17].

The SSH properties are listed in the table below.

Property	Value	Default
com.scalagent.jorammq.ssh.ip	Listening address. If several addresses are available then it can be useful to choose a particularly one for the SSH server entry point. If fixed to 127.0.0.1 (localhost) only local connection are authorized.	
com.scalagent.jorammq.ssh.port	Listening port	18090
com.scalagent.jorammq.ssh.user	User name of the administrator (optional if a key is provided)	admin
com.scalagent.jorammq.ssh.password	Password of the administrator (optional if a key is provided)	adminpass
com.scalagent.jorammq.ssh.admin.public.key	File path to the public key of the administrator. The key format is a X509 certificate.	./conf/admin-cert.pem
com.scalagent.jorammq.ssh.key	File path to the keys of the SSH server. This file is created when the SSH server starts.	./conf/sshkey.ser
com.scalagent.jorammq.ssh.thread.count	Maximum number of threads allocated for the SSH. Each opened ssh session allocates one thread.	2

An example of SSH configuration is:

[17] Public key is required when using Nagios (see section 11).

```
com.scalagent.jorammq.ssh.port=18090
com.scalagent.jorammq.ssh.key=./conf/sshkey.ser
com.scalagent.jorammq.ssh.admin.public.key=./conf/admin-cert.pem
```

Note: Be careful, since JoramMQ 1.11 the SSH administration connection requires Java 8. It's due to the integration of a new release of Apache Mina for security reasons. If you need to use JoramMQ with Java version prior to 8 you have to disable the SSH administration connection in the file conf/felix.properties.

10.2 Usage

Once the server has started, simply connect to the listening port configured with an SSH client, for example:

- ssh -p 18090 admin@localhost

You can also use the jorammq-admin script in bin directory to execute a unique command. For example to stop the server:

- jorammq-admin -exec “stop 0”

🛠 **Warning :** If the related command needs parameters, you must enclose the entire command in quotes as in the example above..

10.3 Global Commands

Below is a recap list of basic commands:

Command	Parameter	Description
stop 0		Shutdown: stop cleanly the server, stopping all OSGi bundles.
exit 0		Shutdown: suddenly stop the server.
lb filter	Optional filter parameter for bundle name.	The “`felix:lb`” command lists installed bundles with identifier and state.
install url	Target URLs.	The “`felix:install`” command installs the bundle specified by the given URL.
start id	Target bundle identifiers.	The “`felix:start`” command starts the specified bundle.
stop id	Target bundle identifiers.	The “`felix:stop`” command stops the specified bundle.
uninstall id	Target bundle identifiers.	The “`felix:uninstall`” command uninstalls the specified bundle.

10.4 JoramMQ/MQTT commands

There are several commands allowing access to the MBean attributes.

10.4.1 Generic SSH commands to access to the MBeans

This section describes generic commands allowing to access all MBeans. The available shell commands are listed in the table below. Each command has to be prefixed with 'jorammq:mqtt:'.

Command	Parameter	Description
getAttribute obj att	<obj>: The object name of the MBean containing the attribute. <att>: A String specifying the name of the attribute to be retrieved.	Gets the value of a specific attribute of a named MBean, the MBean is identified by its object name.
getMBeanInfo obj	<obj>: The object name of the MBean to analyze.	This command discovers all the attributes that an MBean exposes for management.
queryNames exp	<exp>: The object name pattern identifying the MBean names to be retrieved, or a query expression. A '*' character specifies that the name of all registered MBeans will be retrieved[18].	Gets the names of MBeans controlled by the MBean server. This method enables any of the following to be obtained: • the names of all MBeans, • the names of a set of MBeans specified by pattern matching on theObjectName. It returns the set of ObjectNames for the MBeans selected.

10.4.2 Shortcut to some JMX attributes

This section describes specific commands allowing simple access to some MBean attributes. The available shell commands and the returned JMX attributes are listed in the table below. Each command has to be prefixed with 'jorammq:mqtt:'.

Command	Parameter	MBean	JMX attribute
QueuingCountPerMin id	id: MQTT client identifier	MQTT client	QueuingCountPerMin
ToDeliverCount id	id: MQTT client identifier	MQTT client	ToDeliverMessageCount
publishSentRate	-	Mqtt Connector	PublishSentRate
publishReceivedRate	-	Mqtt Connector	PublishReceivedRate
getOperatingSystemAttribute	<att> [<att> ..]	java.lang OperatingSystem	Given in parameter.

[18] Be careful, JMX query expression must be encapsulated between " characters.

processCpuLoad	-	java.lang OperatingSystem	ProcessCpuLoad
processCpuTime	-	java.lang OperatingSystem	ProcessCpuTime
systemCpuLoad	-	java.lang OperatingSystem	SystemCpuLoad
getRuntimeAttribute	<att> [<att> ..]	java.lang Runtime	Given in parameter.
getThreadingAttribute	<att> [<att> ..]	java.lang Threading	Given in parameter.

Be careful, calling a command on a specific MQTT client requires activating its MBean. This MBean must then be explicitly disabled if you do not want to keep it.

10.4.3 Manage connections

This section describes specific commands allowing to manage active connections. Each command has to be prefixed with 'jorammq:mqtt:'.

Command	Parameter	Description
dumpConnection	id1 [id2 ..]: MQTT client identifiers	Prints information about the specified connections.
dumpConnections	-	Lists all connections.
closeConnection	id1 [id2 ..]: MQTT client identifiers	Closes the specified connections.
connectionCount conn	conn: MQTT connector name formatted as: <protocol>-<port>	Prints the number of active connections for the specified connector.
IsConnected id	id: MQTT client identifier	Returns if the specified client is connected.

10.4.4 Manage and clean sessions

If a client is not connected for a significant time, then you might want to consider deleting that client[19]. There are several commands allowing to manage the MQTT sessions, these commands are listed in the table below. Each command has to be prefixed with 'jorammq:mqtt:'.

Command	Parameter	Description
dumpDurableClientIds logDurableClientIds	-	These commands list all durable MQTT sessions[20].
dumpClientIds logClientIds	-	These commands list all MQTT sessions printing only the client identifier[20].

[19] Deleting a session also delete any durable subscriptions (and contained messages) associated to this session.
[20] The “dump” (resp. “log”) command print the resulting list in the standard output (resp. in the current log file).

dumpSessions logSessions	-	These commands list all MQTT sessions[20].
dumpClient id [id ..]	id: MQTT client identifier.	Print the status of the clients whose identifiers are specified. If empty the status of all MQTT clients are printed.
getSubscriptions [context] id	context: Subscription context. id: MQTT client identifier.	Print the list of subscriptions for the specified client identifier in the selected subscription context.
checkToDeliverMessage Count	threshold	Returns the list all MQTT sessions with more than *threshold* messages waiting to be delivered.
checkNotAckedSize	threshold	Returns the list all MQTT sessions for which the total payload size of the unacknowledged messages exceeds the *threshold* parameter.
checkDisconnectionTime	threshold	Returns the list all MQTT sessions disconnected for at least threshold seconds.
deleteSessions id [id ..] -quiet	id: MQTT client identifier.	This command removes the MQTT sessions corresponding to the list of client identifiers. -quiet option suppresses verbose output
deletesOldSessions filter delay except -quiet	filter: regular expression describing the client identifiers to remove. delay: expiration period in seconds. except: optional regular expression allowing to ignore corresponding client identifiers.	This command removes old inactive MQTT sessions for more than delay. Only sessions with a client identifier that matches filter and does not match except are processed. -quiet option suppresses verbose output
cleanExpiredMessages [quiet]	quiet: if true suppresses verbose output.	This command deletes all expired messages.

10.4.5 List of JoramMQ/MQTT commands

The list of command can be retrieved using the usage command without parameter.

```
g! usage
Usage: [jorammq:mqtt:]usage [<command> ..]
        Returns the usage of the specified commands. Without parameter prints
        help about all commands.

Usage: [jorammq:mqtt:]shutdown
        Shutdowns the broker and exits.

Usage: [jorammq:mqtt:]getOperatingSystemAttribute <att> [<att> ..]
        Returns the value of specific attributes of the OperatingSystem
```

```
        MBean.

Usage: [jorammq:mqtt:]processCpuLoad
        Returns the "recent cpu usage" for the Java Virtual Machine process.

Usage: [jorammq:mqtt:]processCpuTime
        Returns the CPU time used by the process on which the Java virtual
        machine is running in nanoseconds.

Usage: [jorammq:mqtt:]systemCpuLoad
        Returns the "recent cpu usage" for the whole system.

Usage: [jorammq:mqtt:]getRuntimemAttribute <att> [<att> ..]
        Returns the value of specific attributes of the Runtime MBean.

Usage: [jorammq:mqtt:]getThreadingAttribute <att> [<att> ..]
        Returns the value of specific attributes of the Threading MBean.

Usage: [jorammq:mqtt:]backup <directory path>]
        Backup the server and returns the name of backup file.
        The backup file is created in directory given in parameter.

Usage: [jorammq:mqtt:]dumpServerState <filename>
        Dump the server state in the specified file.

Usage: [jorammq:mqtt:]setLoggerLevel <logger name> <level name>
        Sets the level of the specified logger to the given value.

Usage: [jorammq:mqtt:]dumpConnections
        Returns the description of all connections.

Usage: [jorammq:mqtt:]dumpConnection <clientId>
        Returns the description of the specified connections.

Usage: [jorammq:mqtt:]closeConnection <clientId>
        Closes the specified connections.

Usage: [jorammq:mqtt:]connectionCount <connectorId>
        Returns the number of active connections for the specified connector.
```

```
Usage: [jorammq:mqtt:]publishReceivedRate
        Returns the number of Publish messages received per second by the MQTT
        server.

Usage: [jorammq:mqtt:]publishSentRate
        Returns the number of Publish messages sent per second by the MQTT
        server.

Usage: [jorammq:mqtt:]joramMQVersion
        Returns the version of the JoramMQ broker.

Usage: [jorammq:mqtt:]joramMQLicense
        Returns the license of the JoramMQ broker.

Usage: [jorammq:mqtt:]joramMQExpiration
        Returns the expiration date of the JoramMQ broker.

Usage: [jorammq:mqtt:]isConnected <clientId>
        Returns true if the specified client is connected.

Usage: [jorammq:mqtt:]toDeliverCount <clientId>
        Returns the number of messages waiting to be delivered to the specified
        client.

Usage: [jorammq:mqtt:]queuingCountPerMin <clientId>
        Returns the number of times a subscriber has been overflowed during a
        minute. If -quiet option is set, suppress the verbose output.

Usage: [jorammq:mqtt:]printClientAttribute <attribute name> <clientID>
[<clientID>]
        Returns the value of the specified attribute for the given clients.

Usage: [jorammq:mqtt:]deleteSessions <clientID> [<clientID> .. -quiet]
        Removes the MQTT sessions corresponding to the list of client
        identifiers. If -quiet option is set, suppress the verbose output.

Usage: [jorammq:mqtt:]deleteOldSessions <filter> <duration> [<except> -quiet]
        Removes old inactive MQTT sessions for more than delay. Only sessions
        with a client identifier that matches filter and does not match except
        are processed. If -quiet option is set, suppress the verbose output.
```

```
Usage: [jorammq:mqtt:]cleanExpiredMessages [<quiet>]
        Removes all expired messages. if quiet is 'true', suppress the verbose
        output.

Usage: [jorammq:mqtt:]dumpDurableClientIds
        Returns the list of all durable MQTT sessions (clean=false), printing
        only the client identifier.

Usage: [jorammq:mqtt:]logDurableClientIds
        Returns the list of all durable MQTT sessions (clean=false), printing
        only the client identifier. The list is also written in log.

Usage: [jorammq:mqtt:]dumpClientIds
        Returns the list of all MQTT sessions, printing only the client
        identifier.

Usage: [jorammq:mqtt:]logClientIds
        Returns the list of all MQTT sessions, printing only the client
        identifier. The list is also written in log.

Usage: [jorammq:mqtt:]dumpClient [<client_id1> [<client_id2> ..]]
        Print the status of the specified MQTT clients.
        If empty the status is printed for all MQTT clients.

Usage: [jorammq:mqtt:]dumpSessions
        Returns the list all MQTT sessions with status.

Usage: [jorammq:mqtt:]logSessions
        Returns the list all MQTT sessions with status.
        The list is also written in log.

Usage: [jorammq:mqtt:]checkToDeliverMessageCount threshold
        Returns the list all MQTT sessions with more than threshold messages
        waiting to be delivered.

Usage: [jorammq:mqtt:]checkNotAckedSize threshold
        Returns the list all MQTT sessions for which the total payload size of
        the unacknowledged messages exceeds the threshold parameter.
```

```
Usage: [jorammq:mqtt:]checkDisconnectionTime threshold
        Returns the list all MQTT sessions disconnected for at least threshold
        seconds.

Usage: [jorammq:mqtt:]getSubscriptions [<context>] <clientID>
        Print the list of subscription for the specified client identifier.
        If the context is not specified, the default one is used.

Usage: [jorammq:mqtt:]queryNames <exp>
        Returns the names of MBeans controlled by the MBean server. The exp
        parameter allows to filter the MBeans.

Usage: [jorammq:mqtt:]getAttribute <obj> <att>
        Returns the value of a specific attribute of a named MBean. The MBean
        is identified by the obj parameter and the attribute is given by the
        att parameter.

Usage: [jorammq:mqtt:]getMBeanInfo <obj>
        Returns all the attributes that an MBean exposes for management. The
        MBean is identified by the obj parameter.

Usage: [jorammq:mqtt:]checkMqttBridgeStatus [<stream1> [<stream2> ..]]
        Print the status of the specified MQTT Bridge streams. If no stream is
         specified as a parameter, the status of all streams are displayed.

Usage: [jorammq:mqtt:]checkMQBridgeStatus [<stream1> [<stream2> ..]]
        Print the status of the specified MQTT/JMS bridge streams. If no stream is
         specified as a parameter, the status of all streams are displayed.

Usage: [jorammq:mqtt:]checkMQtoFileStatus
        Print the status of the JMS to file bridge.

Usage: [jorammq:mqtt:]checkFileToMQStatus
        Print the status of the file to JMS bridge.
```

10.4.6 Examples

- Connection to the secure shell connector:

```
$ ssh admin@localhost -p 18090
```

- List all available commands:

```
g! help
jorammq:mqtt:cleanOlderSession
jorammq:mqtt:cleanSession
jorammq:mqtt:connectionCount
jorammq:mqtt:dumpDurableClientIds
jorammq:mqtt:getAttribute
jorammq:mqtt:getMBeanInfo
jorammq:mqtt:isConnected
jorammq:mqtt:logClients
jorammq:mqtt:publishReceivedRate
jorammq:mqtt:publishSentRate
jorammq:mqtt:queryNames
jorammq:mqtt:queuingCountPerMin
jorammq:mqtt:systemCpuLoad
jorammq:mqtt:toDeliverCount
...
```

- List all MBeans:

```
g! queryNames "*"
ObjectNames:
AgentServer:server=AgentServer#0,cons=Engine#0,agent=MqttAgent[#0.0.16]
…
g! queryNames "*:*"
ObjectNames:
AgentServer:server=AgentServer#0,cons=Engine#0,agent=MqttAgent[#0.0.16]
…
```

- List all Joram destination MBeans:

```
g! jorammq:mqtt:queryNames "Joram#0:type=Destination,*"
ObjectNames:
Joram#0:type=Destination,name=JoramAdminTopic
Joram#0:type=Destination,name=$MqttTopic
Joram#0:type=Destination,name=MqttTopic
```

- List various JoramMQ/MQTT MBeans using a JMX query expression:

```
g! queryNames "com.scalagent.jorammq:host=localhost,manager=ConnectorManager,*"
ObjectNames:
com.scalagent.jorammq:host=localhost,manager=ConnectorManager,connector=ws-9001
com.scalagent.jorammq:host=localhost,manager=ConnectorManager
```

```
com.scalagent.jorammq:host=localhost,manager=ConnectorManager,connector=tcp-1884
com.scalagent.jorammq:host=localhost,manager=ConnectorManager,connector=tcp-1883
g! queryNames "com.scalagent.jorammq:*"
ObjectNames:
com.scalagent.jorammq:host=localhost,manager=Status
com.scalagent.jorammq:host=localhost,manager=SubscriptionManager,subscriptionContext=MqttTopic
com.scalagent.jorammq:host=localhost,manager=Status,connector=tcp-1883
com.scalagent.jorammq:host=localhost,manager=ConnectorManager,connector=ws-9001
com.scalagent.jorammq:host=localhost,manager=SubscriptionManager,subscriptionContext=$MqttTopic
com.scalagent.jorammq:host=localhost,manager=ConnectorManager
com.scalagent.jorammq:host=localhost,manager=ClientManager
com.scalagent.jorammq:host=localhost,manager=Status,connector=tcp-1884
com.scalagent.jorammq:host=localhost,manager=ConnectorManager,connector=tcp-1884
com.scalagent.jorammq:host=localhost,manager=ConnectorManager,connector=tcp-1883
g! queryNames "*:host=localhost,*"
ObjectNames:
com.scalagent.jorammq:host=localhost,manager=Status
com.scalagent.jorammq:host=localhost,manager=SubscriptionManager,subscriptionContext=MqttTopic
com.scalagent.jorammq:host=localhost,manager=Status,connector=tcp-1883
com.scalagent.jorammq:host=localhost,manager=ConnectorManager,connector=ws-9001
com.scalagent.jorammq:host=localhost,manager=SubscriptionManager,subscriptionContext=$MqttTopic
com.scalagent.jorammq:host=localhost,manager=ConnectorManager
com.scalagent.jorammq:host=localhost,manager=ClientManager
com.scalagent.jorammq:host=localhost,manager=Status,connector=tcp-1884
com.scalagent.jorammq:host=localhost,manager=ConnectorManager,connector=tcp-1884
com.scalagent.jorammq:host=localhost,manager=ConnectorManager,connector=tcp-1883
```

- Check the status of a MQTT connector (needs the health-check component, see chapter 12):

```
g! getAttribute com.scalagent.jorammq:host=localhost,manager=Status,connector=tcp-1883 StatusInfo
RUNNING
```

- List all attributes for the MqttTopic destination:

```
g! jorammq:mqtt:getMBeanInfo Joram#0:type=Destination,name=MqttTopic
Joram#0:type=Destination,name=MqttTopic NumberOfSubscribers
Joram#0:type=Destination,name=MqttTopic SubscriberIds
Joram#0:type=Destination,name=MqttTopic Name
Joram#0:type=Destination,name=MqttTopic FreeWriting
```

```
Joram#0:type=Destination,name=MqttTopic FreeReading
Joram#0:type=Destination,name=MqttTopic DMQId
Joram#0:type=Destination,name=MqttTopic DestinationId
Joram#0:type=Destination,name=MqttTopic CreationTimeInMillis
Joram#0:type=Destination,name=MqttTopic CreationDate
Joram#0:type=Destination,name=MqttTopic NbMsgsReceiveSinceCreation
Joram#0:type=Destination,name=MqttTopic NbMsgsDeliverSinceCreation
Joram#0:type=Destination,name=MqttTopic NbMsgsSentToDMQSinceCreation
Joram#0:type=Destination,name=MqttTopic Rights
Joram#0:type=Destination,name=MqttTopic Period
Joram#0:type=Destination,name=MqttTopic ReactTime
Joram#0:type=Destination,name=MqttTopic CommitTime
Joram#0:type=Destination,name=MqttTopic AgentProfiling
Joram#0:type=Destination,name=MqttTopic Fixed
Joram#0:type=Destination,name=MqttTopic ReactNb
Joram#0:type=Destination,name=MqttTopic AgentId
Joram#0:type=Destination,name=MqttTopic ClusterElements
```

- Get the AgentId attribute value for the MqttTopic MBean:

```
g! jorammq:mqtt:getAttribute Joram#0:type=Destination,name=MqttTopic AgentId
#0.0.17
```

- Various specific JoramMQ/MQTT commands:

```
g! jorammq:mqtt:connectionCount tcp-1883
4
g! jorammq:mqtt:dumpSessions
[MqttClientContext [id=MqttQueueId [id=toto, sharedSub=false], hasGarbage=false,
clean=false, lastGroupToGarbage=0, modified=false], MqttClientContext
[id=MqttQueueId [id=Jorammq-MQTT-check-tcp-1883, sharedSub=false],
hasGarbage=false, clean=false, lastGroupToGarbage=0, modified=false],
MqttClientContext [id=MqttQueueId [id=Jorammq-MQTT-check-tcp-1884,
sharedSub=false], hasGarbage=false, clean=false, lastGroupToGarbage=0,
modified=false], MqttClientContext [id=MqttQueueId [id=tutu, sharedSub=false],
hasGarbage=false, clean=false, lastGroupToGarbage=0, modified=false],
MqttClientContext [id=MqttQueueId [id=titi, sharedSub=false], hasGarbage=false,
clean=false, lastGroupToGarbage=0, modified=false], MqttClientContext
[id=MqttQueueId [id=tata, sharedSub=false], hasGarbage=false, clean=false,
lastGroupToGarbage=0, modified=false], MqttClientContext [id=MqttQueueId
[id=bridge-jorammq, sharedSub=false], hasGarbage=false, clean=false,
lastGroupToGarbage=0, modified=false], MqttClientContext [id=MqttQueueId
[id=freyssin_162525971, sharedSub=false], hasGarbage=false, clean=true,
lastGroupToGarbage=0, modified=false], ]
g! jorammq:mqtt:isConnected bridge-jorammq
true
g!  deleteOldSessions "Jorammq-.*-tcp-.*" 30  "Jorammq-.*-tcp-1884"
```

```
Clean older sessions: [Jorammq-MQTT-check-tcp-1883:36S, ]
```

10.4.7 Unix shell shortcuts

The *postinstall-sshcommands* command in the bin directory allows you to install several additional commands accessible from your favorite shell. The syntax and semantics of these commands are identical to the commands described above. A non-exhaustive list of these shortcuts is:

- isConnected,
- connectionCount,
- deleteSessions,
- deleteOldSessions,
- dumpClient and getSubscriptions,
- checkMqttBridgeStatus,
- checkMQBridgeStatus,
- checkMQtoFileStatus,
- checkFileToMQStatus.

For example:

```
$ ./dumpClient Jorammq-MQTT-check-tcp-1883 Jorammq-MQTT-check-tcp-1884
Password:

_______________________________
Welcome to Apache Felix Gogo

g! dumpClient Jorammq-MQTT-check-tcp-1883 Jorammq-MQTT-check-tcp-1884
MQTTClientContext ClientId=Jorammq-MQTT-check-tcp-1883 CONNECTED
 # Clean: false, Queued QoS0: false, TTL: 0
 # Last connection: Thu Dec 05 13:24:41 CET 2019, Connections since last boot: 2
 # Delivered: 2, Dropped: 0
MQTTClientContext ClientId=Jorammq-MQTT-check-tcp-1884 DISCONNECTED
 # Clean: false, Queued QoS0: false, TTL: 0
 # Last connection: Thu Dec 05 13:24:41 CET 2019, Connections since last boot: 3
 # Delivered: 3, Dropped: 0
$ ./checkMqttBridgeStatus
Password:

_______________________________
Welcome to Apache Felix Gogo
g! checkMqttBridgeStatus
```

```
MQTTBridge stream=bridge-jorammq CONNECTED
 # Last connection: 4 dec. 2019 07:39:47 / 1 successful
 # Last disconnection: - (0 ongoing attempts)
 # 153219 forwarded messages since last boot
```

10.5 A3 and Joram/JMS commands

Additional administration commands are available from Joram to manipulate the server. These commands are packaged in three additional bundles:

- shell-a3.jar for commands related to the underlying A3 middleware,
- shell-mom.jar for commands related to the JMS MOM,
- and shell-jndi.jar for commands related to the embedded implementation of JNDI.

10.5.1 A3 Commands description

The *shell-a3.jar* bundle give access to commands allowing to administrate the A3 server on which Joram is running.

- `[joram:a3:]close` - This commands stops the Joram and exits the server.
- `[joram:a3:]engineLoad` - This commands returns the average number of pending notifications over the last minute, it is an evaluation of the server's load.
- `[joram:a3:]garbageRatio` - This commands gives the garbage ratio when the NG transaction mode is set.
- `[joram:a3:]info [options...]` - This commands shows numerous information about the server, the network (in the case of clustering) and/or the transactional persistence manager. By default this command show information about all embedded components, the following options are available:
- `-eng` - Shows info about the engine,
- `-ngt` - Shows info about the transactional persistence manager,
- `-net` - Shows info about the network (in the case of clustering).
- `[joram:a3:]restartServer` - Stops and restarts Joram
- `[joram:a3:]startServer` - Starts Joram
- `[joram:a3:]stopServer` - Stops Joram

10.5.2 MOM/JMS Commands description

The *shell-mom* bundle offers commands allowing to monitor and administrate the destination, subscriptions and users.

- `[joram:mom:]clear queue <name>`
  `[joram:mom:]clear subscription <user name> <subscription name>`
  Clears all pending messages of the queue or suscription.

- `[joram:mom:]create queue <name> [options]`
  `[joram:mom:]create topic <name> [options]`
  Creates a new queue or topic, the following options are avaiblable:

  - `-sid` - Unique identifier of the server on which the destination must be deployed.

  - `-ext` - Implementation class of the destination.

- `[joram:mom:]delete queue <name>`
  `[joram:mom:]delete topic <name>`
  Deletes the specified destination.

- `[joram:mom:]create user <name>`
  Creates a new user, the command prompts for a password.

- `[joram:mom:]delete user <name>`
  Deletes the specified user.

- `[joram:mom:]deleteMsg queue <name> <identifier>`
  `[joram:mom:]deleteMsg subscription <user> <subscription>) <identifier>`
  Delete a pending message from a queue or a subscription.

- `[joram:mom:]help [<command name>]`
  Displays usage information about the specified MOM command. If no command is given, lists all MOM commands.

- `[joram:mom:]info queue <name>`
  `[joram:mom:]info topic <name>`
  `[joram:mom:]info subscription <user name> <subscription name>)`
  Gives info about the destination or subscription.

- `[joram:mom:]list destination`
  `[joram:mom:]list queue`
  `[joram:mom:]list topic`
  Lists all destinations defined on the server, or only queues or topics.

- `[joram:mom:]list user`
  Lists all users defined on the server.

- `[joram:mom:]list subscription <user name>`
  Lists subscriptions of the specified user.

- `[joram:mom:]lsMsg queue <queue name> [[first]:[last]]`
  `[joram:mom:]lsMsg subscription <user name> <sub. name> [[first]:[last]]`
  Lists the pending messages of a queue or subscription. Returns messages only messages in the specified interval if it is set, all messages otherwise.

- `[joram:mom:]ping` – Tells whether this Joram server is running.
- `[joram:mom:]queueLoad` <queue name> – Gives the average load for the specified queue.
- `[joram:mom:]subscriptionLoad <user name> <subscription name>` – Gives the average load for the specified subscription.
- `[joram:mom:]receiveMsg` – Not yet implemented.
- `[joram:mom:]sendMsg` – Not yet implemented.

10.5.3 JNDI Commands description

The *shell-jndi* bundle currently give access only to the `joram:jndi:list` command. This commands lists all records in the naming context, giving useful info about the registered destinations and connection factories.

11 Nagios

JoramMQ can be monitored with Nagios by installing some plugins and configuration files.

The Nagios plugins and a configuration example are separately released in:

```
jorammq-mqtt-nagios-<version>.zip
```

11.1 Authentication

The secure shell provided by JoramMQ is used by the plugins.

RSA public/private keys need to be generated for the user 'nagios'.

```
ssh-keygen -t rsa -C "nagios@host"
```

And a certificate (format X509) has to be passed to the MQTT server (see section 10.1).

```
openssl req -x509 -days 365 -new -key .ssh/id_rsa -out admin-cert.pem
```

Copy 'admin-cert.pem' in ./conf and uncomment the certificate property in 'felix.properties':

```
# Path to a certificate file (X509 format)
com.scalagent.jorammq.ssh.admin.public.key=./conf/admin-cert.pem
```

Login a first time in order to add the MQTT server host in .ssh/known_hosts:

```
ssh <mqtt server host> -p 18090
The authenticity of host '[192.168.1.212]:18090 ([192.168.1.212]:18090)' can't be
established.
DSA key fingerprint is 3c:8e:19:47:c6:ad:67:9d:22:df:79:41:cc:d6:5d:22.
Are you sure you want to continue connecting (yes/no)?
```

11.2 Nagios plugins

The Nagios plugins are scripts that query attributes from the MQTT server through the secure shell. These scripts are written in Python.

11.2.1 Delta check

This script checks that an indicator reaches a target value and remains between lower and upper limits. The limits are specified as gaps between the target value and the actual value of the indicator.

Argument	Short option	Long option	Type	Description

Host name	-H	--host	string	Host name or IP address of the MQTT server
Port	-P	--port	integer	Listening port of the secure shell provided by the MQTT server
Upper warning gap	-W	--upperWarning	string	Added to the target value to obtain the upper warning limit
Upper critical gap	-C	--upperCritical	string	Added to the target value to obtain the upper critical limit
Lower warning gap	-w	--lowerWarning	string	Removed from the target value to obtain the lower warning limit
Lower critical gap	-c	--lowerCritical	string	Removed from the target value to obtain the lower critical limit
Operation	-o	--operation	string	Shell command to be executed by the MQTT server
Indicator name	-n	--name	string	Name of the indicator displayed by Nagios
Indicator type	-t	--type	string	Available values are: int, float
Target value	-v	--value	string	Value to be reached by the indicator

11.2.2 Flag check

This script checks that a flag (boolean) has the expected value. If not the script returns CRITICAL. There is no WARNING level.

Argument	**Short option**	**Long option**	**Type**	**Description**
Host name	-H	--host	string	Host name or IP address of the MQTT server
Port	-P	--port	integer	Listening port of the secure shell provided by the MQTT server
Operation	-o	--operation	string	Shell command to be executed by the MQTT server
Flag name	-n	--name	string	Name of the flag displayed by Nagios
Expected value	-v	--value	string	If not equal to this value, then CRITICAL is returned

11.2.3 Limit check

This script checks that an indicator reaches a target value and remains between lower and upper limits. The limits are specified as options.

Argument	**Short option**	**Long option**	**Type**	**Description**

Host name	-H	--host	string	Host name or IP address of the MQTT server
Port	-P	--port	integer	Listening port of the secure shell provided by the MQTT server
Upper warning limit	-W	--upperWarning	string	Upper warning limit
Upper critical limit	-C	--upperCritical	string	Upper critical limit
Lower warning limit	-w	--lowerWarning	string	Lower warning limit
Lower critical limit	-c	--lowerCritical	string	Lower critical limit
Operation	-o	--operation	string	Shell command to be executed by the MQTT server
Indicator name	-n	--name	string	Name of the indicator displayed by Nagios
Indicator type	-t	--type	string	Available values are: int, float

11.3 Check a JMX attributes

This section shows how to check a given JMX MBean attribute, it uses the Nagios plugins defined above, and the getAttibute command described in the section 10.4.1. It defines two Nagios plugins called 'checkAttribute' and 'deltaCheckAttribute' and a Nagios service monitoring the number of active MQTT connections.

```
define command {
  command_name checkAttribute
  command_line /home/nagios/jorammq-nagios/limit_check.py -H $HOSTADDRESS$ -P
18090 $ARG2$ -o 'getAttribute $ARG1$'
}
define command {
  command_name deltaCheckAttribute
  command_line /home/nagios/jorammq-nagios/delta_check.py -H $HOSTADDRESS$ -P
18090 $ARG2$ -o 'getAttribute $ARG1$'
}
```

This command can be used to define several service, the arguments of the check_command are:

1. the object name of the MBean and the attribute name, for example:
   - com.scalagent.jorammq:host=localhost,manager=ConnectorManager ConnectionCount
2. the plugin options, for example:
   - -W 3 -C 6 -n 'ConnectionCount' -t int

```
define service{
        use                             local-service
```

```
        host_name                       myHost
        service_description             MQTT Attribute:ConnectorManager
ConnectionCount
        check_command checkAttribute!
com.scalagent.jorammq:host=localhost,manager=ConnectorManager ConnectionCount!-W 3
-C 6 -n 'ConnectionCount' -t int
        check_interval 1
        retry_interval 1
}
```

12 MQTT connector health-check

This component periodically monitors the health of the TCP connectors and publish indicators related to the Jorammq MQTT server status. It checks each TCP transport every 'period' and publish either a status info ("INSTALLED", "RUNNING" or "STOPPED"), connection latency and publish/subscribe latency.

Additionally in case of multiple failures it can generate an error report with the complete state of the server (all MBean's attributes and thread's stacktraces).

12.1 Installation

The JoramMQ/MQTT health check plugin is shipped as an OSGi bundle (jorammq-mqtt-check.jar in the bundle directory). Normally jorammq-mqtt-check bundle is installed and activated in JoramMQ/MQTT server default configuration. In other case, you need to add the jorammq-mqtt-check bundle to the list of installed bundles as follows:

```
felix.auto.start.1= \
  file:./bundle/monolog.jar \
  ...
  file:./bundle/jorammq-ssh.jar \
  file:./bundle/jorammq-mqtt-check.jar
```

12.2 Configuration

The Check component does have some configuration options. These are configured via the Felix properties file in the conf directory.

Property	Description
com.scalagent.jorammq. check.period	Time wait in seconds between two successive check (Default 60 seconds). A value equal or less to zero deactivates the check.
com.scalagent.jorammq. check.timeout	Maximum wait time in seconds during connection attempts (Default 10 seconds).
com.scalagent.jorammq. check.dump.file	Base name of the file containing the generated dump of the server state (Default “jorammq.dump”). A timestamp is added to this name.
com.scalagent.jorammq. check.dump.threshold	Number of unsuccessful attempt to connect before generating a dump of the server (Default 5).
com.scalagent.jorammq. check.dump.delay	Minimum delay between the generation of two dump files (By default 600 seconds, 10 minutes).
com.scalagent.jorammq. check.user	User name to use for connection (by default empty for anonymous connection).

com.scalagent.jorammq.check.password	Password to use for connection (by default empty for anonymous connection).
com.scalagent.jorammq.check.clientid.prefix	Prefix used to define the client identifier of the connection (By default, 'Jorammq-MQTT-check-'). The client identifier is then built by adding the name of the checked connector.
com.scalagent.jorammq.check.topic.prefix	Prefix used to define the topic used by the Check component (By default, 'topic/Jorammq-MQTT-check-'). The topic name is then built by adding the name of the checked connector.
com.scalagent.jorammq.check.qos	QoS used to check the connectors (default 1).
com.scalagent.jorammq.check.keystore	The pathname of the file containing the dedicated keystore, if not defined the JoramMQ default keystore is used (cf. Section 5.3.21). These settings are considered valid only if both “keystore” and “keystorepass” are defined.
com.scalagent.jorammq.check.keystorepass	The password for the keystore.
com.scalagent.jorammq.check.keystoretype	The type of keystore to use. The default is JKS.

By default when installed the health checking is done for all defined MQTT connectors. However you can separately configure the monitoring of each TCP connector via the com.scalagent.jorammq.check.{name}.period, com.scalagent.jorammq.check.{name}.timeout and com.scalagent.jorammq.check.{name}.qos properties, where {name} is the name of the related TCP connector (tcp-1883 for example).

12.2.1 Access control

By default the Check component uses an anonymous connection and therefore no longer works when access control is enabled.

In order to allow it to work properly you must define a user with the appropriate rights, and specify the use of that user by the module using the properties defined above.

Note: In the default configuration, a ‘guest’ user is defined (name 'guest', password 'guest') with appropriate ACL configuration in conf/acl.xml file. If you want to use it you can uncomment the related properties in conf/felix.properties file.

12.3 MBeans

12.3.1 Status MBean

ObjectName: com.scalagent.jorammq:host=localhost,manager=Status

Attributes

Attribute	Type	Description
Period	int	Time wait between two check, in second (Default 60).
Threshold	int	Number of unsuccessful attempt to connect before generating a dump of the server (Default 5).
Period	int	Minimum delay between the generation of two dump files (By default 600 seconds, 10 minutes).
QoS	int	The default QoS for all checked connectors.
Status	int	The global status value (0 if all connector are running, 1 otherwise).
StatusInfo	String	Global status info "RUNNING" or “UNREACHABLE”.

Operations

Operation	Parameter(s)	Description
updateMqttServers	-	Update the TCP transport list for this server.
enableConnectorStatus	Client Id	Enable the MBean of the status of the specified MQTT connection
disableConnectorStatus	Client Id	Disable the MBean of the status of the specified MQTT connection
enableAllConnector	-	Enable the MBean of the status of all MQTT connection
disableAllConnector	-	Disable the MBean of the status of all MQTT connection
dumpServerState	Pathname	Dump all MBean's attributes and thread's stacktraces in a file with the given pathname.

12.3.2 MQTT connector status MBean

ObjectName:com.scalagent.jorammq:host=localhost,manager=Status,connector=<name>

Attributes

Attribute	Type	Description
Period	int	Time wait between two successive checks, in second (Default value 60).
TimeOut	int	The maximum time to wait in second (Default value 10).
QoS	Int	The QoS used to check this connector.
LatencyConnect	long	The last measured connection latency.
LatencyPubSub	long	The last measured pub/sub latency.

Status	int	The status value (0 if the last connection is ok, number of unsuccessful attempts otherwise).
StatusInfo	String	Status info "INSTALLED", "RUNNING", “UNREACHABLE” or "STOPPED"

Operations

Operation	Parameter(s)	Description
start	-	Start the check task (send/receive message).
stop	-	Stop the check task.

12.4 Secure Shell commands

This section describes specific commands allowing simple access to some MBean attributes and operation of the MQTT health check plugin.

The Check bundle adds some commands to the secure Shell:

Command	Parameter	Description
getLatencyConnect name	The connector name.	Print the last connexion latency of the specified connector.
getLatencyPubSub name	The connector name.	Print the last publish/Subscribe latency of the specified connector.
getStatusInfo name	The connector name.	Print the last status of the specified connector.
getStatusAttribute name att	The connector and attribute names.	Print the last value of the attribute for the specified connector.
dumpServerState path	The pathname of the file.	Dumps the state of the server in the specified file.

Each command may be prefixed with 'jorammq:mqtt:'.

12.5 MQTT health check with nagios

The examples below show how to use the health-check commands to monitor the status and latency of the TCP connectors in Nagios.

Status

The following example of Nagios command called 'checkStatus' checks that the default TCP connectors are reachable using the getStatusAttribute command of the shell (see section above). It defines a Nagios service for each MQTT connector to check the connection status:

- if the return value is 0 the last check is OK,
- if the return value is less than 4 then a warning is displayed,
- the status becomes critical if greater than 5.

```
define command {
  command_name checkStatus
  command_line /home/nagios/jorammq-nagios/limit_check.py -H $HOSTADDRESS$ -P
18090 -W 1 -C 5 -o 'getStatusAttribute $ARG1$' -n 'checkStatus' -t int
}
```

This command can be instantiated several times for several TCP connectors. For example, the following services monitor the connectors listening on port 1883 and 1884.

```
define service{
        use                             local-service
        host_name                       myHost
        service_description             MQTT tcp-1883: Status
        check_command checkStatus!tcp-1883 Status
        check_interval 1
        retry_interval 1
}

define service{
        use                             local-service
        host_name                       myHost
        service_description             MQTT tcp-1884: Status
        check_command checkStatus!tcp-1884 Status
        check_interval 1
        retry_interval 1
}
```

Latency

The following example of Nagios commands called 'checkLatencyConnect' and 'checkLatencyPubSub' check the responsiveness of the TCP connectors. It uses getLatencyConnect and getLatencyPubSub commands of the shell to get the latency of the corresponding connector during the connection and hear-beat exchange.

```
define command {
  command_name checkLatencyConnect
  command_line /home/nagios/jorammq-nagios/limit_check.py -H $HOSTADDRESS$ -P
18090 -W 20 -C 100 -c -1 -o 'getLatencyConnect $ARG1$' -n 'LatencyConnect' -t int
}

define command {
  command_name checkLatencyPubSub
  command_line /home/nagios/jorammq-nagios/limit_check.py -H $HOSTADDRESS$ -P
18090 -W 250 -C 500 -c -1 -o 'getLatencyPubSub $ARG1$' -n 'LatencyPubSub' -t int
```

```
}
```

These commands can be instantiated several times for several TCP connectors. For example, the following services monitor the connectors listening on ports 1883 and 1884.

```
define service{
        use                             local-service
        host_name                       myHost
        service_description             MQTT tcp-1883: LatencyConnect
        check_command checkLatencyConnect!tcp-1883
        check_interval 1
        retry_interval 1
}

define service{
        use                             local-service
        host_name                       myHost
        service_description             MQTT tcp-1883: LatencyPubSub
        check_command checkLatencyPubSub!tcp-1883
        check_interval 1
        retry_interval 1
}

define service{
        use                             local-service
        host_name                       myHost
        service_description             MQTT tcp-1884: LatencyConnect
        check_command checkLatencyConnect!tcp-1884
        check_interval 1
        retry_interval 1
}

define service{
        use                             local-service
        host_name                       myHost
        service_description             MQTT tcp-1884: LatencyPubSub
        check_command checkLatencyPubSub!tcp-1884
        check_interval 1
        retry_interval 1
}
```

13 MQTT-SN Gateway

JoramMQ provides a gateway component for MQTT-SN. This gateway takes the form of an OSGi component that can be added to the configuration of JoramMQ through the felix.properties file in the conf directory.

13.1 Installation

The JoramMQ/MQTT-SN plugin is shipped as an OSGi bundle (jorammq-mqtt-sn-gw.jar in the bundle directory). By default the jorammq-mqtt-sn-gw bundle is not activated in JoramMQ/MQTT server configuration, you need to explicitly add this bundle to the list of installed bundles as follows:

```
felix.auto.start.1= \
 file:./bundle/monolog.jar \
  …
 file:./bundle/jorammq-ssh.jar \
 file:./bundle/jorammq-mqtt-check.jar \
 file:./bundle/jorammq-mqtt-sn-gw.jar \
  …
```

13.2 Configuration

The MQTT-SN gateway does have some configuration options. It can be configured either via the Felix properties file in the conf directory or through an external property file.

Property	Description
com.scalagent.jorammq.mqtt-sn-gw.config	Path to external configuration file. If not set the configuration is done through properties in the Felix properties file.

The properties below allow to configure the gateway:

Property	Description
gwId	The ID of the gateway, this field is required.
inVM	If true the gateway uses the JoramMQ InVM connector (should be configured), otherwise it uses the TCP connector (see below for host and port configuration). By default inVM is set to false.

brokerURL	The hostname or IP adress of the MQTT broker, this field is required if TCP connector is used.
brokerTcpPort	The TCP port where MQTT broker listens, this field is required if TCP connector is used.
clientInterfaces	Comma separated list of available client interfaces. Each interface is given by the class implementing the corresponding client interface. Currently, for compatibility reasons with the previous version of the MQTT-SN gateway, the class name is in parentheses between '<' and '>'. By default the UDP client interface is used.
udpPort	The UDP port that will be used for the UDP socket of the UDPClientInterface. By default udpPort is set to 20.000.
predfTopicIdSize	Predefined topic ids can take values in [1,N] where N is indicated in predfTopicIdSize. By default the maximum of predefined topic is 30.
predefinedTopicsFile	Properties file for predefined topic ids. By default the gateway try to use the file "predefinedTopics.properties" in the conf directory.
advPeriod	The period (in seconds) of broadcasting the MQTT-SN ADVERTISE message to the network. By default set to 1.200 seconds.
keepAlivePeriod	The period (in seconds) of sending the MQTT PINGREQ message to the broker. By default set to 10 seconds.
maxRetries	Maximum retries of sending a message to the client, by default 3 attempts.
waitingTime	Maximum time (in seconds) waiting for a message from the client. By default 10 seconds.
maxMqttsLength	The maximum length of the MQTT-SN message, by default 180 bytes.
minMqttsLength	The minimum length of the MQTT-SN message, by default 2 bytes.
handlerTimeout	The time (in seconds) that a ClientMsgHandler can remain inactive. By default 86.400 seconds (1 day).
forwarderTimeout	The time (in seconds) that a Forwarder can remain inactive, by default 300 seconds (5 minutes).
checkingPeriod	The period (in seconds) that a control message is sent to all ClientMsgHandlers for removing themselves from Dispatcher's mapping table if they are inactive for at least handlerTimeout seconds. By default 86.400 seconds (1 day).

These properties below allow to configure the options of the MQTT connections handled by the gateway:

Property	Description
cleanSession	Sets whether the client and server should remember state across restarts and reconnects. By default false.
willQoS	QoS for the "Last Will and Testament" message. By default 0.
willRetain	Retain option for the "Last Will and Testament" message. By default false.
willTopic	Sets the "Last Will and Testament" topic for the connection. If willTopic and willMessage are not defined the mechanism is disabled.
willMessage	Sets the "Last Will and Testament" message for the connection. If willTopic and willMessage are not defined the mechanism is disabled.

These properties below allow to configure the gateway logging:

Property	Description
logLevel	Set level of logging for gateway, it should be INFO, WARN or ERROR. By default, logging is set to WARN.
logDir	Directory containing the gateway log file. By default, the log directory of installation.
logFile	Name of the gateway logging file. By default, “mqtts_gateway.log”.

These properties below allow to configure the MQTT protocol used by the gateway, currently MQTT v3.1. It should not be modified by the end-user of the gateway.

Property	Description
protocolName	Protocol name as defined in the MQTT specification. By default "MQIsdp", corresponding to the version 3 of protocol.
protocolVersion	Protocol version as defined in the MQTT specification. By default 3, corresponding to MQTT v3.1.

Note: Most properties have default values. However, it is at least necessary to define the gateway id (property gwId) and the MQTT broker URL (brokerURL and brokerPort properties) if the internal connector is not used (inVM property).

13.3 MQTT-SN Gateway MBean

Attributes

Attribute	Type	Description
GatewayId	int	Unique identifier of this MQTT-SN gateway.
InVM	boolean	True if the gateway uses the internal connector.
runnning	boolean	

Operations

Operation	Parameter(s)	Description
start	-	Starts the gateway if it is stopped.
stop	-	Stops the gateway if it is running. Currently inactive.

14 MQTT bridge

An MQTT bridge allows a JoramMQ server to connect to another MQTT server and publish some MQTT messages according to some criteria. As seen in the figure below, the bridge consists of 2 parts:

- On the one hand, a local MQTT client which manages the wanted subscriptions, and stores the messages waiting to be forwarded to the remote broker.
- On the other hand, a client to the remote MQTT broker, which transfers the messages received to this broker.

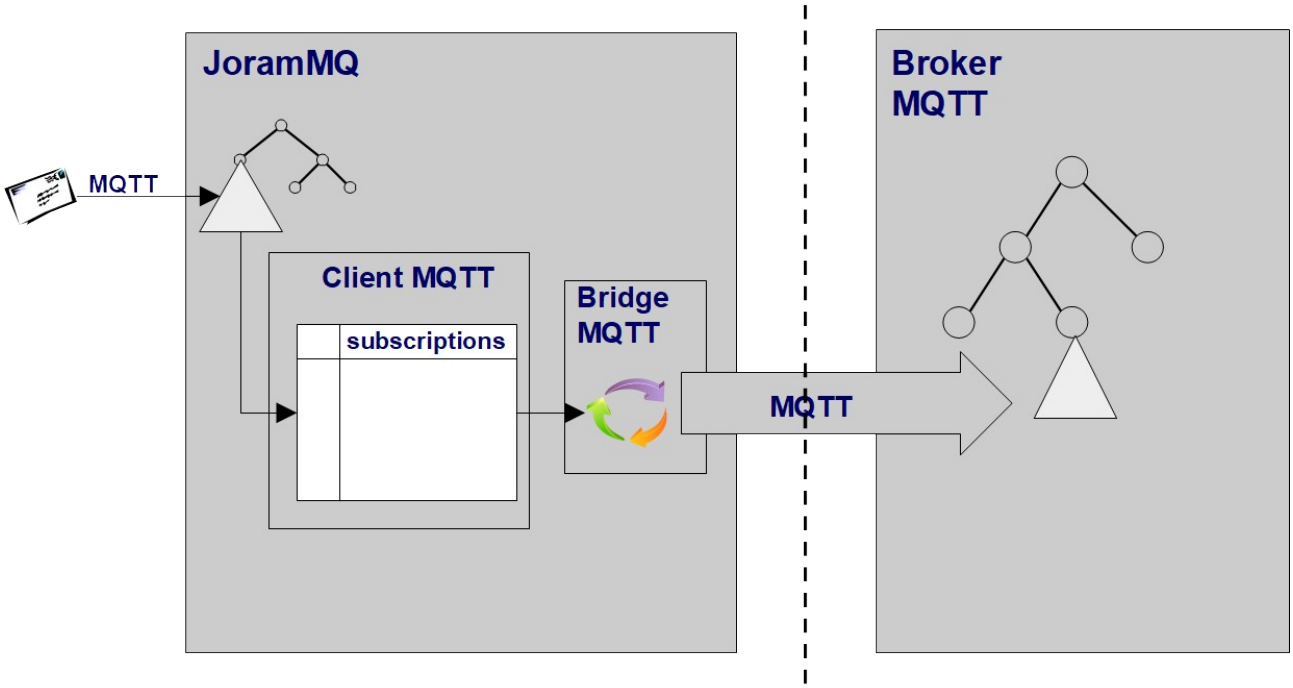

Figure 2: Bridge MQTT

The configuration file is specified in 'conf/jorammq.xml'.

```
<property name="com.scalagent.jorammq.mqtt.bridge.file"
    value="./conf/bridge.xml" />
```

The configuration file contains a unique 'config' element.

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="http://www.scalagent.com/schema/MqttBridgeSchema">
  …
```

An element 'bridge' needs to be declared for every connection with a remote MQTT server (named a stream). It takes as parameters:

- The URI of the remote MQTT server (available protocols are tcp, ssl, tls):
  - The QoS properties defined in section 5.3 can be appended to the URI using their short names and separated by the XML entity "*amp&;*".

- The MQTT clientId to be used when connecting. Be careful, this client identifier is used both in the local and remote server, you must take care that it is unique in these two contexts.
- The *clean* session flag, by default true[21].
- The user name and the password used for authentication on the remote server. Please note that there is no user associated locally with the bridge, and the bridge is not affected by ACLs, whether at connection or subscription level.
- The topic prefix to be added to the topic of the published messages. Be careful '/' character is always added between this prefix and the topic of the published messages. Depending of the given prefix, and of the topic of published messages, this may lead to 2 or 3 '/' together but this is conform with the spirit of the MQTT specification.
  - Since JoramMQ 1.16, it is possible to specify an empty prefix or not to specify a prefix. In this case, no character is added to the prefix in order to allow the transmission to the remote broker of messages on a topic identical to that of the broker bridge. This makes it possible to define a "transparent" bridge.
- The configuration parameters of the connection will: willMessage, willTopic, willQos (by default 0, “at most once”) and willRetain (by default false). The will is only used if parameters willMessage and willTopic are defined.

For example:

```
<bridge name="bridge-jorammq" uri="tcp://<remote server address>:<remote port>"
    clientId="<bridge client id>" clean="true"
    userName="<bridge user name>" password="<bridge password>"
    prefix="bridge/jorammq">
```

An element 'outbound' specifies the (local) topic that is published to the remote MQTT server and the QoS used to publish the messages.

```
  <bridge … >
    <outbound topic="b/c" qos="1" />
    <outbound topic="a/#" qos="0" />
    <outbound topic="a/+/d" qos="1" />
  </bridge>
```

Each bridge element can define specific properties, properties allowed are:

- delay.base, delay.max, connect.max, reconnect.max
- keepalive, maxConsumerWindowSize, maxInflightMessageCount
- maxQueuedCount, queuedQos0, discardOld

🛠 **Warning :** Be careful, shared subscriptions are not handled by the MQTT bridge.

🛠 **Warning :** The MQTT bridge experimentally allows the use of WebSocket connections (WS and WSS), these streams may still be subject to malfunctions.

[21] This value is always false before JoramMQ 1.13.

14.1 Using durable sessions

Since JoramMQ 1.13 the old unwanted subscriptions are deleted, however some messages handled before the server restart can be forwarded to the destination broker.

Before JoramMQ 1.13

When the “clean” attribute of the bridge configuration is set to false, the bridge uses a durable session. As specified by the MQTT semantics, the related subscriptions are then durable. If you reconfigure an outbound stream, modifying the topic filter, the old subscriptions will remain active. As a consequence the messages published on these old subscriptions will still be sent onto the resulting outbound stream.

This behaviour may not be desirable. If you need to prevent it, it is required that you change the client identifier of the modified bridge in order to use a new session. The old session may then be removed using JMX or SSH commands.

14.2 Using secure connections

If the connection to the remote MQTT server is secure, the default security settings used are those of the server hosting the bridge. However you can specify different settings for each connection:

- “keystore” specifies the pathname of the keystore file to use.
- “keystorepass” specifies the password of the keystore.
- “keystoretype” specifies the type of the keystore, by default “JKS”.

For example:

```
<bridge name="bridge-jorammq" uri="ssl://<remote server address>:<remote port>"
    clientId="<bridge client id>" clean="true"
    userName="<bridge user name>" password="<bridge password>"
    keystore=”/home/user/bridgeks.jks” keystorepass=”mypassword”
    prefix="bridge/jorammq/" >
```

These settings are considered valid only if both “keystore” and “keystorepass” are defined.

Keystore update

When the validity of a certificate is about to expire, it is necessary to add a new valid certificate to the keystore. However the broker must be stopped and restarted for this change to take effect.

Keystore checking

The broker performs some basic sanitary checks over the content of the keystore. It first occurs when the broker starts, and then when certificates are about to expire. The output of this checking is done using the logging system. It is controlled by the logger named com.scalagent.jorammq.-mqtt.adapter.KeyStoreStatus.

The MQTT bridge keystore must contain a valid certificate of a trusted authority, used to authenticate the remote broker. If none can be found, an error log is issued. It should also contain a valid private key and its associated certificate. This certificate is necessary when the broker requires client authentication. If none can be found, a warning log is issued.

The broker may also issue warning and error logs when certificates are about to expire. The notice periods are configured in the jorammq.xml file by the same properties used to check the main broker keystore (cf 5.3.21 MQTT server Keystore).

14.3 Using HTTP proxy

Since version 1.16 of JoramMQ it is possible to specify the use of an HTTP proxy when connecting the MQTT bridge to the remote server:

- For the TCP and SSL/TLS protocols, the declaration of the proxy will be done by setting up a tunnel using the HTTP CONNECT primitive.
- For the WS protocols, the declaration of the proxy requires that this protocol be taken into account by the proxy (to be checked according to the proxy used).
- For the WSS protocol, the declaration of the proxy will lead to the setting up of a tunnel using the HTTP CONNECT primitive in order to secure the flow from end to end.

The declaration of a proxy is made in the URI of the remote MQTT server of the bridge. You must fill in the "proxy.host" parameter with the DNS name or the IP address of the machine hosting the host, and the "proxy.port" parameter with the listening port of the proxy on this machine.

The example below shows us the declaration of an MQTT bridge to the broker with URI “tcp://remote:1883” using an HTTP proxy listening on the machine “proxy” and port 3128.

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ns="http://www.scalagent.com/schema/MqttBridgeSchema">
  <bridge name="bridge-jorammq" uri="tcp://remote:1883?
proxy.host=proxy&amp;proxy.port=3128"
    clientId="bridge-jorammq" userName="" password="" clean="false"
prefix="bridge">
    <outbound topic="home/#" qos="2" />
  </bridge>
</config>
```

With TCP and SSL/TLS protocol the MQTT bridge allows basic authentication with the HTTP proxy. It needs to complete the URI with the “proxy.auth” parameter. This parameter must contain the username and password separated by the ":" character. For example "uri="tcp://remote:1883? … &proxy.auth=jorammq:pass" if the declared user is "jorammq" and the corresponding password is "pass".

Currently the MQTT bridge for WebSocket does not allow authentication with the HTTP proxy.

14.4 Configuration

The MQTT bridge component does have some configuration options. These are globally configured via the jorammq.xml configuration file in the conf directory (see below). Each MQTT bridge flow can be specifically using property elements in the bridge XML configuration file.

Property	Description
com.scalagent.jorammq.mqtt.bridge.publishStatus	If this property is defined to true an MQTT message is published on $SYS/broker/mqttbridge/{name}/status topic each time the bridge is connected or disconnected. Default value: false.
com.scalagent.jorammq.mqtt.bridge.delay.base	Minimum delay (in ms) between two connection attempts, the delay increases after each attempt. Default value: 100ms.
com.scalagent.jorammq.mqtt.bridge.delay.max	Maximum delay (in ms) between two connection attempts. Default value: 10.000ms.
com.scalagent.jorammq.mqtt.bridge.connect.max	Maximum number of attempts before failure for the initial connection. Default value: -1, infinite tries.
com.scalagent.jorammq.mqtt.bridge.reconnect.max	Maximum number of connection attempts before failure for the reconnection mechanism. Default value: -1, infinite tries.
com.scalagent.jorammq.mqtt.bridge.keepalive	Maximum time interval in seconds between messages sent from the MQTT bridge. If needed PINGREQ are sent. Default value: 30s.
com.scalagent.jorammq.mqtt.bridge.maxConsumerWindowSize	Maximum number of message payload bytes allowed to be in-flight for the bridge MQTT subscriber client before doing flow control (see section 5.3.3). Default value: 32768
com.scalagent.jorammq.mqtt.bridge.maxQueuedCount	Maximum number of messages that are allowed to be enqueued for the bridge MQTT client (see section 5.3.9). Default value: 0 (no limit).
com.scalagent.jorammq.mqtt.bridge.maxInflightMessageCount	Maximum number of messages that are allowed to be sent before doing flow contro (This limit is only valid for messages whose QoS level is either 1 or 2, see section 5.3.8). Default value: 65535
com.scalagent.jorammq.mqtt.bridge.queuedQos0	Flag enabling to queue messages whose QoS level is 0 (see section 5.3.5). Default value: false
com.scalagent.jorammq.mqtt.bridge.discardOld	Flag determining the bridge behaviour when the maximum of queued message is reached (see section 5.3.10). Default value: false

The outgoing MQTT connection can be configured through URI parameters:

- snd.buf: size of the sending buffer in bytes.
- rcv.buf: size of the receiving buffer in bytes.

14.5 MQTT bridge MBean

MBean ObjectName: com.scalagent.jorammq:host=localhost,manager=MqttBridge,stream=*name* where *name* is the name of the bridge stream declared in the configuration file.

🛠 **Warning :** Be careful, the "MQTT bridge" MBean represents the connection to the remote server. As seen Figure 2 each bridge is associated with a local client context that manages the subscriptions. Additional information, such as the number of pending messages is available in the MBean of this client.

14.5.1 Attributes

Attribute	Type	Description
Name	String	Value of the name configuration attribute.
Uri	String	Value of the uri configuration attribute.
ClientId	String	Value of the clientId configuration attribute.
CleanSessionFlag	boolean	Value of the clean configuration attribute.
User	String	Value of the userName configuration attribute.
TopicPrefix	String	Value of the prefix configuration attribute.
OutboundConfigs	String[]	Value of the outbound configuration elements.
Subscribed	Boolean	True if the bridge has properly subscribed to the local topics.
MQTTConnected	Boolean	True if the bridge connection to the remote MQTT is up.
Status	String	State of the component.
LastConnect	String	Date of the last successful connection to the remote MQTT.
LastDisconnect	String	Date when the last successful connection to the remote MQTT was closed.
SuccessfulConnections	long	Number of successful connections to the remote MQTT since last reboot.
ReconnectAttempts	long	Number of connection attempts since last disconnection.
Forwarded	long	Number of forwarded message since last reboot.
Queued	long	Number of sent messages waiting for acknowledgment[22].

[22] This indicator only represents the messages being processed, the number of messages awaiting transfer is accessible in the MBean of the associated client context (ToDeliverMessageCount attribute).

14.5.2 Operations

Operation	Parameter(s)	Description
checkKeyStoreSanity	int, int	Performs sanitary checks over the content of the bridge keystore. Returns a String. First param: max validity delay (days) used to issue WARN messages. 0: use configured value, -1: issue no WARN messages. Second param: max validity delay (days) used to issue ERROR messages. 0: use configured value, -1: issue no ERROR messages.
checkKeyStoreSanityStatus	int, int	Same as checkKeyStoreSanity but returns an int. OK(0), WARN(1), ERROR(2)

14.6 Secure Shell commands

This section describes specific commands allowing simple access to some MBean attributes and operation of the MQTT bridge.

The MQTT bridge bundle adds some commands to the secure Shell:

Command	Parameter	Description
checkMqttBridgeStatus [stream1 [stream ..]]	Stream names	Print the status of the specified MQTT Bridge streams.

Each command may be prefixed with 'jorammq:mqtt:'.

For example:

```
g! checkMqttBridgeStatus
MQTTBridge stream=bridge-jorammq CONNECTED
 # Last connection: 4 dec. 2019 07:39:47 / 1 successful
 # Last disconnection: - (0 ongoing attempts)
 # 153219 forwarded messages since last boot
 # Queued: 0, Waiting: 0
```

15 MQTT/JMS bridge

An MQTT/JMS bridge allows a JoramMQ server to connect to a JMS server and publish some MQTT messages according to some criteria:

- If a list of *base* topics to forward is given by the user (see topic.prefix and topic.filter properties below) then a stream is created for each of them, this topic is named the base topic of the stream. This stream includes a MQTT connection with a subscription to the related topic and its subtree. It also includes a JMS connection allowing that sends each received message to the remote JMS queue with a name corresponding to the base topic of the stream.

- If the list of topics is empty, streams are dynamically created for each sub-topic of the root topic (see topic.prefix property below). There is a unique MQTT connection, and a unique JMS connection for all streams. Each received message is sent to the remote JMS queue with a name corresponding of the base topic of the stream.

Each forwarded JMS message includes a String property containing the name of the incoming topic of the corresponding MQTT message. By default, the name of this property is "mqtt_topic".

Since JoramMQ 1.17 you can specify multiples ConnectionFactory allowing connection to different JMS servers. The component operates in 2 modes depending on the value of the property "com.scalagent.jorammq.mqtt.mqbridge.jms.loadbalancing":

- If the property is false, only one JMS connection is active at any given time. In case of disconnection each flow will try to reconnect by successively using each defined ConnectionFactory.

- If the property is true, one JMS connection is established for each ConnectionFactory defined. Incoming messages are transmitted according to a round-robin algorithm to the different JMS brokers.

Of course, the necessary queues must have been previously created on each JMS broker and saved in the JNDI service. The JNDI name of each JMS Destination should be the concatenation between the ConnectionFactory name, and the name of the base topic. For example if we define a flow “queue”, with 2 ConnectionFactory (“cf1” and “cf2”), we needs to register in JNDI:

- “cf1_queue”: name of the Queue in the JMS broker accessible through the ConnectionFactory “cf1”.

- “cf2-queue”: name of the Queue in the JMS broker accessible through the ConnectionFactory “cf2”.

Authentication credentials can normally be stored in the ConnectionFactory and therefore can be found in JNDI. If this is not the case, username and password must be filled using the corresponding properties. if each server requires specific credentials then usernames and passwords must be separated using the ‘#’ character.

Since JoramMQ 1.18 the transacted[23] mode of JMS can be used to increase the performance of the JMS stream. This can be particularly useful when the JMS connection is inefficient. This mode is configured through 2 properties:

- “com.scalagent.jorammq.mqtt.mqbridge.jms.transacted": this property determines the maximum number of messages that can be bundled into a single transaction. If less than 2, the transacted mode is not used. Currently values greater than 1000 are not allowed, the default value is 10.

- "com.scalagent.jorammq.mqtt.mqbridge.jms.transacted.timeout”: this property determines the maximum delay (in milliseconds) between 2 messages grouped together in the same transaction. If this delay is exceeded between 2 successive messages, the messages of the current transaction are sent, and a new transaction is started. Currently values greater than 10000 (10 seconds) are not allowed, the default is 100.

The MQTT/JMS bridge is packaged as a bundle named “jorammq-mqtt-mq-bridge.jar” and its configuration is done in 'conf/felix.properties' file.

15.1 Installation

The JoramMQ MQTT/JMS bridge plugin is shipped as an OSGi bundle (jorammq-mqtt-mq-bridge.jar in the bundle directory). Normally the MQTT/JMS bridge is not activated in JoramMQ/MQTT server default configuration, so you need to add the jorammq-mqtt-mq-bridge bundle to the list of installed bundles as follows:

```
felix.auto.start.1= \
 file:./bundle/monolog.jar \
 ...
 file:./bundle/jorammq-mqtt-check.jar \
 file:./bundle/ow2-jms-2.0-spec.jar \
 file:./bundle/ow2-jta-1.1-spec.jar \
 file:./bundle/jndi-client.jar \
 file:./bundle/joram-client-jms.jar \
 file:./bundle/jorammq-mqtt-mq-bridge.jar \
 file:./bundle/javax.annotation-api.jar \
 ...
 file:./bundle/joram-tools-rest-jmx.jar
```

As it appears in the example above, the bridge requires the jars of the APIs (JMS and JTA) and that of the JMS client.

Note: For compatibility reason the old component is kept in the bundle “./bundle/jorammq-mqtt-mq-bridge-old.jar”, it corresponds to the version 1.17 of the component.

[23] Transacted mode enable to group multiple message send operations together in a single atomic unit. This mode makes it possible to group the sending and reduce the impact of network latency. It also optimizes the transactional cost on the broker.

15.2 Configuration

The MQTT/JMS bridge component does have some configuration options. These are configured via the Felix properties file in the conf directory.

Note: All these properties are prefixed by “com.scalagent.jorammq.mqtt.mqbridge”.

Property	Description
<prefix>.mqtt.uri	URI of source MQTT broker. Default value: “tcp://localhost:1883”
<prefix>.mqtt.user	User name needed to authenticate to source MQTT broker. Default value: empty, no authentification.
<prefix>.mqtt.password	Password needed to authenticate to source MQTT broker. Default value: empty, no authentification.
<prefix>.clientid.prefix	Prefix for the MQTT client identifier (clientid), the final clientid used by each stream is built concatenating this prefix with the name of the base topic of the stream. Default value: “Jorammq-MQTT-bridge-”
<prefix>.qos	QoS used for MQTT subscriptions, it shall be at least 1 to avoid loss of messages. Default value: 2.
<prefix>.topic.prefix	Path of base topic of the tree of topics to forward. Default value: empty.
<prefix>.topic.filter	List of all topic names[24] defining a MQTT/JMS stream, each item of the list is separated by the # character. The MQTT/JMS bridge defines a separate stream for each topic, forwarding all MQTT messages sent on this topic and subtree to the corresponding JMS destination. If empty the MQTT/JMS bridge defines a unique stream working dynamically. Be careful, in this mode a message needing to be forwarded to an undefined JMS destination will be lost. Default value: empty.
<prefix>.persistence.mode	Either “memory” (default) or “file”. Memory persistence can be used In cases where reliability is not required across client or device restarts. In other cases a non-volatile form of persistence should be used.
<prefix>.persistence.directory	Il persistence mode is set to file this property allows to fix the directory to use.
<prefix>.jndi.factory.class	This property specifies which initial context factory to use when creating a new initial context object. Default value: “fr.dyade.aaa.jndi2.client.NamingContextFactory” allowing to use the Joram's JNDI service.

[24] This name is relative to the root topic defining by the com.scalagent.jorammq.mqtt.mqbridge.topic.prefix property.

<prefix>.jndi.url	The URL of JNDI service allowing to retrieve JMS administered objects: ConnectionFactory and destinations. Default value: currently null (use the default Joram's TCPConnectionFactory), next “scn://localhost:16400”.
<prefix>.jndi.cf	The JNDI's name of the ConnectionFactory to use to connect to the remote JMS broker. Default value: “cf”.
<prefix>.jms.user	User name needed to authenticate to destination JMS broker. If not fixed the default authentication values from ConnectionFactory is used. If each server requires specific credentials then usernames must be separated by the ‘#’ character.
<prefix>.jms.password	Password needed to authenticate to destination JMS broker. If not fixed the default authentication values from ConnectionFactory is used. If each server requires specific credentials then passwords must be separated by the ‘#’ character.
<prefix>.reconnect.period	Period in milliseconds between 2 connection attempts, by default 15 seconds (15000).
<prefix>.jms.property	The name of the JMS property used to transport the incoming topic. Be careful, this name must conform to the JMS specification. Default value: “mqtt_topic”
<prefix>.jms.text	Corresponding to the MQTT semantic the MQTT/JMS bridge sends BytesMessage messages to the JMS destination. Setting this property to true allows to use TextMessage messages. Default value: false.
<prefix>.jms.loadbalancing	If false, a unique JMS connection is established (multiple ConnectionFactory are used for availability). If true, one JMS connection is established for each ConnectionFactory defined. The incoming messages are distributed according to a round-robin algorithm to the different brokers.

🛠 **Warning :** Be careful, if you do not use a JNDI service, the MQTT / JMS bridge uses the Session.createQueue JMS method to find the remote queues. These queues must first have been created administratively.

15.3 MBeans

15.3.1 MQBridge activator MBean

This MBean is the root Mbean of the MQTT/JMS Bridge component. It gives the global state of the component (running or not) and displays the main configuration parameters.

MBean ObjectName: com.scalagent.jorammq:host=localhost,manager=MQBridge

Attributes

Attribute	Type	Description
Status	String	State of the MQTT/JMS bridge component, should be RUNNING.
Delay	Integer	Delay in milliseconds between 2 connection attempts, by default 15.000 (15 seconds).
MQTTURI	String	URI of source MQTT broker.
TopicPrefix	String	Path of base topic of the tree of topics to forward.
TopicFilter	String	List of all topic names defining a MQTT/JMS stream (see com.scalagent.jorammq.mqtt.mqbridge.topic.filter property).
JNDIParameters	String	JNDI URL and parameters used to connect to the target JMS broker.
JMSProperty	String	Name of the JMS property used to transport the incoming topic.

Operations

Operation	Parameter(s)	Description
start	-	Starts the MQTT/JMS bridge.
stop	-	Stops the MQTT/JMS bridge.

15.3.2 MQBridge stream MBean

This MBean is the root MBean of each MQTT/JMS stream. It gives the global state of the stream (connected or not), the number of forwarded messages, and information about the stream.

MBean ObjectName: com.scalagent.jorammq:host=localhost,manager=MQBridge,stream=*name*

Where name is the name of the stream.

Attributes

Attribute	Type	Description
Name		Name of the stream.
Status	String	The state of the MQTT/JMS Stream, either CONNECTED or DISCONNECTED.
MQTTConnected	Boolean	True if the MQTT connection is up, false otherwise.
JMSConnected	Boolean	True if a JMS connection is up, false otherwise.
Forwarded	long	Number of forwarded message since last reboot.

Operations

Operation	Parameter(s)	Description
printFullStatus	-	Returns a synthetic status of the MQTT/JMS stream.
start	-	Start the MQTT/JMS stream.
stop	-	Stop the MQTT/JMS stream.

15.3.3 MQBridge stream MQTT connector MBean

This MBean describes the status of the MQTT part of the stream. It describes the state of the canal (connected or not), information about the connection, and configuration parameters.

MBean ObjectName: com.scalagent.jorammq:host=localhost,manager=MQBridge,stream=*name*, canal=MQTT

Where name is the name of the stream.

Attributes

Attribute	Type	Description
Name		Name of the stream.
Status	String	The state of the MQTT connector, either OFF, STOPPED, CONNECTED, DISCONNECTED or FAILED.
MQTTConnected	boolean	True if the MQTT connection is up, false otherwise.
LastConnect	String	Date of the last successful connection of the MQTT/JMS stream.
LastDisconnect	String	Date when the last successful connection of the MQTT/JMS stream was closed.
SuccessfulConnections	long	Number of successful connections of the MQTT/JMS stream since last reboot.
ReconnectAttempts	long	Number of current reconnection attempts.
MQTTClientId	String	The ClientId used for the current MQTT session.
MQTTTopicFilter	String	The topic filter used for subscribing.

Operations

Operation	Parameter(s)	Description
start	-	Starts the MQTT/JMS bridge.
stop	-	Stops the MQTT/JMS bridge.

15.3.4 MQBridge stream JMS connector MBean

This MBean describes the status of a JMS connector, there are as many MBeans as there are JMS connections defined by the stream. Each MBean describes the state of the canal (connected or not), information about the connection, and configuration parameters.

MBean ObjectName: com.scalagent.jorammq:host=localhost,manager=MQBridge,stream=*name, canal=JMS[cf]*

Where name is the name of the stream, and cf the name of the ConnectionFactory used by the connector.

Attributes

Attribute	Type	Description
Name		
Status	String	The state of the MQTT/JMS Stream.
JMSConnected	boolean	True if a JMS connection is up, false otherwise.
LastConnect	String	Date of the last successful connection of the MQTT/JMS stream.
LastDisconnect	String	Date when the last successful connection of the MQTT/JMS stream was closed.
SuccessfulConnections	long	Number of successful connections of the MQTT/JMS stream since last reboot.
ReconnectAttempts		Number of current reconnection attempts.
Forwarded	long	Number of forwarded message since last reboot.
currentCF	String	The name of the ConnectionFactory used to connect.
JMSDestinationName	String	The JMS destination name.

Operations

Operation	Parameter(s)	Description
start	-	Start the MQTT/JMS stream.
stop	-	Stop the MQTT/JMS stream.

15.4 Administration / Monitoring

Three SSH commands are available:

- GetMQBridgeStatus <stream>
  - This command returns the status of the stream whose name is given as a parameter.

- getMQBridgeForwarded <stream>
  - This command returns the number of forwarded messages of the stream whose name is given as a parameter.
- checkMQBridgeStatus [<stream1> [<stream2>] ..]
  - This command returns a synthetic status of the streams. It takes as parameter the list of streams that we want to list (if not specified all streams are listed).

The figure below gives an example of the checkMQBridgeStatus command. The MQTT/JMS bridge defines 2 streams (“queue1”, “queue2”), each using 2 ConnectionFactory (“cf1”, “cf2”) to forward messages in parallel to 2 brokers (load-balancing mode).

```
_____________________________
Welcome to Apache Felix Gogo

g! checkMQBridgeStatus
MQBridge stream=queue1 CONNECTED, 36 forwarded messages since last boot.
 + MQTT CONNECTED
 |      ClientId: BRIDGE-MQTT_JMS-queue1, TopicFilter: home/queue1/#
 |      Last connection: 14 oct. 2022 à 09:53:35 / 1 successful
 |      Last disconnection: - / 0 ongoing attempts
 + JMS JMS[cf1] CONNECTED
 |     CurrentCF: cf1, Destination: cf1_queue1
 |     Last connection: 14 oct. 2022 à 09:53:35 / 1 successful
 |     Last disconnection: - / 0 ongoing attempts
 |     18 forwarded messages since last boot
 + JMS JMS[cf2] CONNECTED
 |     CurrentCF: cf2, Destination: cf2_queue1
 |     Last connection: 14 oct. 2022 à 09:53:35 / 1 successful
 |     Last disconnection: - / 0 ongoing attempts
 |     18 forwarded messages since last boot
MQBridge stream=queue2 CONNECTED, 37 forwarded messages since last boot.
 + MQTT CONNECTED
 |      ClientId: BRIDGE-MQTT_JMS-queue2, TopicFilter: home/queue2/#
 |      Last connection: 14 oct. 2022 à 09:53:35 / 1 successful
 |      Last disconnection: - / 0 ongoing attempts
 + JMS JMS[cf1] CONNECTED
 |     CurrentCF: cf1, Destination: cf1_queue2
 |     Last connection: 14 oct. 2022 à 09:53:35 / 1 successful
 |     Last disconnection: - / 0 ongoing attempts
 |     18 forwarded messages since last boot
 + JMS JMS[cf2] CONNECTED
```

```
|     CurrentCF: cf2, Destination: cf2_queue2
|     Last connection: 14 oct. 2022 à 09:53:35 / 1 successful
|     Last disconnection: - / 0 ongoing attempts
|     19 forwarded messages since last boot
```

A Unix shell shortcut is available in bin directory: checkMQBridgeStatus (see section 10.4.7).

🛠 **Warning :** During its configuration, the bridge creates an MQTT session per user-defined stream. During subsequent starts, if streams are deleted, it is necessary to manually delete the corresponding sessions (see section 3.8 or 10.4.4).

15.5 Examples

This section describes briefly a typical usage of the MQTT/JMS bridge

The bridge is used to forward messages to 2 different JMS Brokers MQ#1 and MQ#2. The MQ#1 broker can be accessed via the ConnectionFactory “cf1” registered in the JNDI service, the MQ#2 through the ConnectionFactory “cf2”.

The configuration defines 2 streams:

- The first one corresponds to message published to MQTT topics “<root>/X1/#”, these messages are forwarded to JMS Queue X1 on brokers MQ#1 and MQ#2. These queues needs to be registered in JNDI with names “cf1_X1” and “”cf2_X1”.
- The second one corresponds to message published to MQTT topics “<root>/X2/#”, these messages are forwarded to JMS Queue X2 on brokers MQ#1 and MQ#2. These queues needs to be registered in JNDI with names “cf1_X2” and “”cf2_X2”.

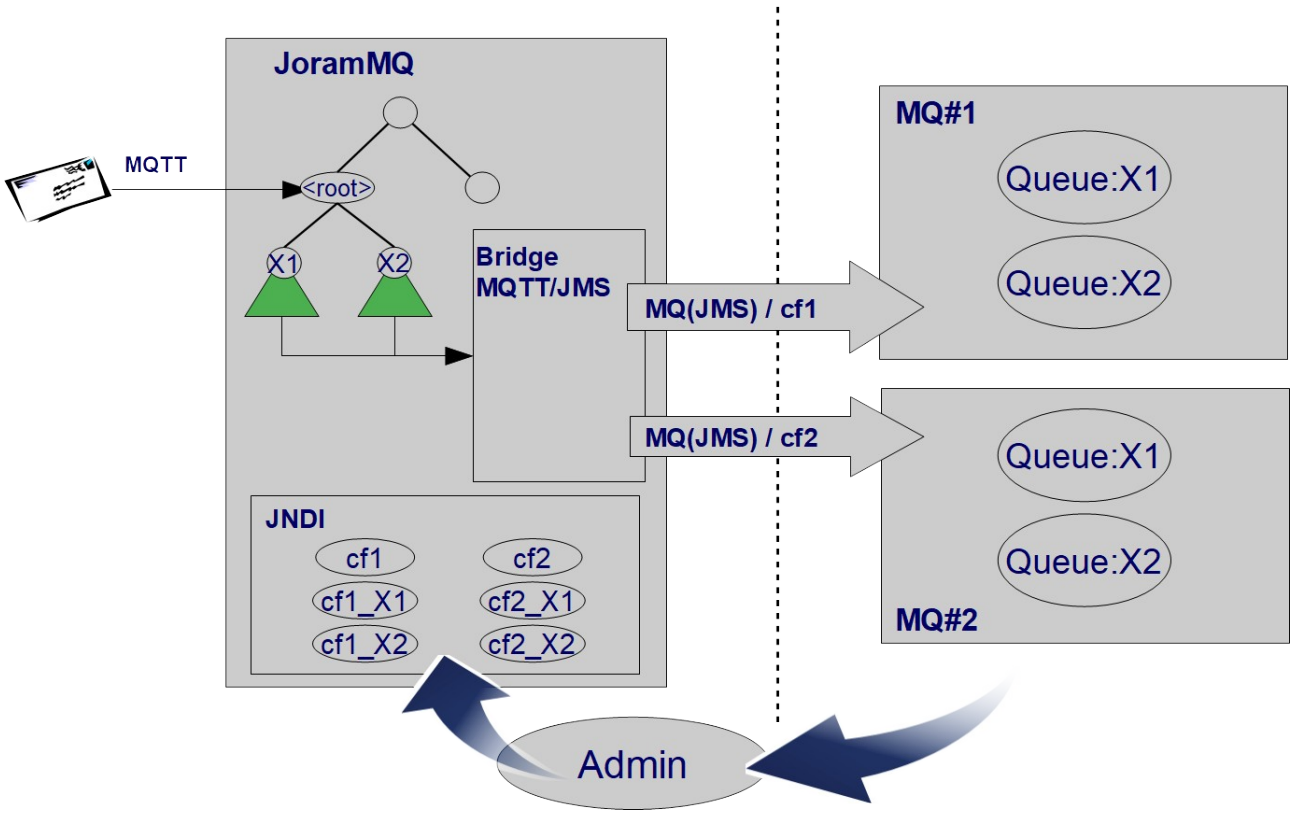

Figure 3: Architecture bridge MQTT/JMS

The configuration in felix.properties should look like:

```
com.scalagent.jorammq.mqtt.mqbridge.qos=2
com.scalagent.jorammq.mqtt.mqbridge.topic.prefix=<root>
com.scalagent.jorammq.mqtt.mqbridge.topic.filter=X1#X2
com.scalagent.jorammq.mqtt.mqbridge.jndi.cf=cf1#cf2
com.scalagent.jorammq.mqtt.mqbridge.jms.user=user1#user2
com.scalagent.jorammq.mqtt.mqbridge.jms.password=password1#password2
```

You can choose between availability or load-balancing mode by setting property "com.scalagent.jorammq.mqtt.mqbridge.jms.loadbalancing".

16 JMS to file

This component allows a JoramMQ server to connect to a JMS broker and consume messages from selected queues (see queue.list property below). The body of each received message (TextMessage or BytesMessage) is then saved locally in a disk file; the resulting filename is built from a global prefix (see prefix property below) and a JMS property associated to the message (see jms.property below).

If there is an error[25] during the handling of message this one is denied and put back in the queue. If the resulting file already exists, it is overwritten.

The JMS to file component is packaged as a bundle and its configuration is done in 'conf/felix.properties' file.

16.1 Installation

The JMS to file plugin is shipped as an OSGi bundle (jorammq-mqtofile.jar in the bundle directory). Normally the JMS to file bundle is not activated in JoramMQ/MQTT server default configuration, so you need to add the jorammq-mqtofile bundle to the list of installed bundles as follows:

```
felix.auto.start.1= \
 file:./bundle/monolog.jar \
 ...
 file:./bundle/jorammq-mqtt-check.jar \
 file:./bundle/ow2-jms-2.0-spec.jar \
 file:./bundle/ow2-jta-1.1-spec.jar \
 file:./bundle/jndi-client.jar \
 file:./bundle/joram-client-jms.jar \
 file:./bundle/jorammq-mqtofile.jar \
 file:./bundle/javax.annotation-api.jar \
 ...
 file:./bundle/joram-tools-rest-jmx.jar
```

As it appears in the example above, the bridge requires the jars of the APIs (JMS and JTA) and of the implementation JMS client[26].

[25] Bad message type (only TextMessage and BytesMessage are allowed) for example or error during file creation and writing.

[26] This configuration allows to get message from a Joram/JMS broker, if you use another broker you need to provide the corresponding libraries.

16.2 Configuration

The JMS to file component does have some configuration options. These are configured via the Felix properties file in the conf directory.

Property	Description
com.scalagent.jorammq. mqtofile.jndi.factory.class	This property specifies which initial context factory to use when creating a new initial context object. Default value: “fr.dyade.aaa.jndi2.client.NamingContextFactory” allowing to use the Joram's JNDI service.
com.scalagent.jorammq. mqtofile.jndi.url	The URL of JNDI service allowing to retrieve JMS administered objects: ConnectionFactory and destinations. Default value: “scn://localhost:16400” allowing to use the Joram's JNDI service.
com.scalagent.jorammq. mqtofile.jndi.cf	The JNDI's name of the ConnectionFactory to use to connect to the remote JMS broker. Default value: “cf”.
com.scalagent.jorammq. mqtofile.jms.user	User name needed to authenticate to destination JMS broker. If not fixed the default authentication values from ConnectionFactory is used.
com.scalagent.jorammq. mqtofile.jms.password	Password needed to authenticate to destination JMS broker. If not fixed the default authentication values from ConnectionFactory is used.
com.scalagent.jorammq. mqtofile.reconnect.period	Period in milliseconds between 2 connection attempts, by default 15 seconds (15000).
com.scalagent.jorammq. mqtofile.queue.list	List of names of all incoming queues, this property is mandatory. Each item in the list is separated by the ':' character. There is no default value.
com.scalagent.jorammq. mqtofile.prefix	Prefix of the path, this value is concatenated to the value of the message property to build the final path. The corresponding directory is created at initialisation, if there is an error the component is not started (see warning below). Default value: “.”, the current directory.
com.scalagent.jorammq. mqtofile.jms.property	Name of the message property containing the path of the file to store the body of the message. The final path is the concatenation of the prefix (see property above) and the value of the message property. If needed any necessary but nonexistent parent directories are created. Default value: “pathname”.
com.scalagent.jorammq. mqtofile.tmpdir	Path of the directory to create the temporary file, if it is not defined it uses the default temporary-file directory (see warning below). There is no default value.

🛠 **Warning :** In order to ensure atomicity this component uses a 'move' operation between the temporary file and its real destination. This operation might not be able to move a file from one

filesystem to another, or it might not be atomic. So we strongly recommend that you set the temporary-file directory in the same filesystem as the directory that is designated by the prefix property.

16.3 MBeans

16.3.1 MQtoFile MBean

MBean ObjectName: com.scalagent.jorammq:host=localhost,manager=MQtoFile

Attributes

Attribute	Type	Description
Status	String	State of the component.
Prefix	String	Prefix of the path, this value is concatenated to the value of the message property to build the final path.
JMSProperty	String	Name of the message property containing the path of the file to store the body of the message.
JNDIParameters	String	JNDI URL and ConnectionFactory name used to connect to the target JMS broker.

Operations

Operation	Parameter(s)	Description
start	-	Starts the component.
stop	-	Stops the component.

16.3.2 MQTT connector status MBean

MBean ObjectName: com.scalagent.jorammq:host=localhost,*manager=MQtoFile, stream=in*

Attributes

Attribute	Type	Description
Status	String	The state of the Stream.
LastConnect	String	Date of the last successful connection to the broker.
LastDisconnect	String	Date of the last loss of connection with the broker.
Stored	long	Number of forwarded message since last reboot.

JMSConnected	boolean	Status of the JMS connection.

Operations

Operation	Parameter(s)	Description
start	-	Start the component.
stop	-	Stop the component.

16.4 Administration / Monitoring

A synthetic status of the component can be obtained via the checkMQtoFileStatus[27] SSH command (see chapter 10).

A Unix shell shortcut is available in bin directory: checkMQtoFileStatus (see section 10.4.7).

[27] This command may be prefixed with 'jorammq:mqtt:'.

17 File to JMS

This component allows a JoramMQ server to scan files in directories and send them to a JMS broker. It scans files in specified directories and then sends each file content to the corresponding JMS queue in a JMS message (either TextMessage or BytesMessage depending of the related property). The pathname of the source file is added to a JMS property associated to the message (see jms.property below). The name of the JMS Queue is the last part of the input directory name.

When the file is successfully forwarded to the JMS broker it is removed from its source directory or moved in an archive subdirectory (default ‘saved’) as specified in configuration. If an error occurs during sending the file is moved in an error subdirectory (default ‘error’).

The File to JMS component requires the input file to be added to the input directory as an atomic operation. To this end a good practise would be to create a “draft” subdirectory in the input directory, to create the input file in that draft directory, then to move the input file from the draft directory into the input directory.

The File to JMS component is packaged as a bundle and its configuration is done in 'conf/felix.properties' file.

17.1 Installation

The File to JMS plugin is shipped as an OSGi bundle (jorammq-filetomq.jar in the bundle directory). Normally the File to JMS bundle is not activated in JoramMQ/MQTT server default configuration, so you need to add the jorammq-filetomq bundle to the list of installed bundles as follows:

```
felix.auto.start.1= \
 file:./bundle/monolog.jar \
 ...
 file:./bundle/jorammq-mqtt-check.jar \
 file:./bundle/ow2-jms-2.0-spec.jar \
 file:./bundle/ow2-jta-1.1-spec.jar \
 file:./bundle/jndi-client.jar \
 file:./bundle/joram-client-jms.jar \
 file:./bundle/jorammq-filetomq.jar \
 file:./bundle/javax.annotation-api.jar \
 ...
 file:./bundle/joram-tools-rest-jmx.jar
```

As it appears in the example above, the bridge requires the jars of the APIs (JMS and JTA) and of the implementation JMS client[28].

[28] This configuration allows to get message from a Joram/JMS broker, if you use another broker you need to provide the corresponding libraries.

17.2 Configuration

The File to JMS component does have some configuration options. These are configured via the Felix properties file in the conf directory.

Property	Description
com.scalagent.jorammq. filetomq.jndi.factory.class	This property specifies which initial context factory to use when creating a new initial context object. Default value: “fr.dyade.aaa.jndi2.client.NamingContextFactory” allowing to use the Joram's JNDI service.
com.scalagent.jorammq. filetomq.jndi.url	The URL of JNDI service allowing to retrieve JMS administered objects: ConnectionFactory and destinations. Default value: “scn://localhost:16400” allowing to use the Joram's JNDI service.
com.scalagent.jorammq. filetomq.jndi.cf	The JNDI's name of the ConnectionFactory to use to connect to the remote JMS broker. Default value: “cf”.
com.scalagent.jorammq. filetomq.jms.user	User name needed to authenticate to destination JMS broker. If not fixed the default authentication values from ConnectionFactory is used.
com.scalagent.jorammq. filetomq.jms.password	Password needed to authenticate to destination JMS broker. If not fixed the default authentication values from ConnectionFactory is used.
com.scalagent.jorammq. filetomq.directory.list	List of names of all scanned directory, this property is mandatory. Each item in the list is separated by the system path separator character (‘:’ on UNIX systems, ‘;’ on Microsoft Windows systems). There is no default value.
com.scalagent.jorammq. filetomq.delay	Delay in milliseconds between 2 directory scan. Default value: 10000.
com.scalagent.jorammq. filetomq.jms.property	Name of the message property containing the absolute path of the forwarded file. Default value: “pathname”.
com.scalagent.jorammq. filetomq.jms.text	By default the File to JMS component sends BytesMessage messages to the JMS destination. Setting this property to true allows to use TextMessage messages. Default value: false.
com.scalagent.jorammq. filetomq.archive	If true files are moved in an archive subdirectory after sending. If false files are deleted. Default value: “false”
com.scalagent.jorammq. filetomq.archive.subdirectory	Name of the archive subdirectory. Default value: “saved”
com.scalagent.jorammq. filetomq.error.subdirectory	Name of the error subdirectory. Default value: “error”

The configuration may be changed after a first execution of the component. It will be effective after the next restart of the broker.

17.3 MBeans

17.3.1 FileToMQ MBean

MBean ObjectName: com.scalagent.jorammq:host=localhost,manager=FileToMQ

Attributes

Attribute	Type	Description
Status	String	State of the component.
Directories	String	List of scanned directory.
JMSProperty	String	Name of the message property containing the absolute path of the forwarded file.
JNDIParameters	String	JNDI URL and ConnectionFactory name used to connect to the target JMS broker.

Operations

Operation	Parameter(s)	Description
start	-	Starts the component.
stop	-	Stops the component.

17.3.2 MQTT connector status MBean

MBean ObjectName: com.scalagent.jorammq:host=localhost,*manager=FiletoMQ, stream=out*

Attributes

Attribute	Type	Description
Status	String	The state of the Stream.
LastConnect	String	Date of the last successful connection to the broker.
LastDisconnect	String	Date of the last loss of connection with the broker.
Forwarded	long	Number of forwarded message since last reboot.
JMSConnected	boolean	Status of the JMS connection.

Operations

Operation	Parameter(s)	Description
start	-	Start the component.
stop	-	Stop the component.

17.4 Administration / Monitoring

A synthetic status of the component can be obtained via the checkFileToMQStatus[29] SSH command (see chapter 10).

A Unix shell shortcut is available in bin directory: checkFileToMQStatus (see section 10.4.7).

[29] This command may be prefixed with 'jorammq:mqtt:'.

18 Cluster of servers

A cluster of JoramMQ servers allows to share the message processing load among a group of servers. For example in a cluster of two servers A and B, a client can send messages to server A and another client can receive these messages from server B, assuming the latter client has subscribed to the right topics.

JoramMQ supports scaling up or down a cluster, i.e. dynamically adding or removing a server without stopping the other servers.

Client sessions are not replicated in the cluster. Therefore, if disconnected, clients with a durable session (clean session flag set to false) need to reconnect to the same server otherwise they would create a new session.

18.1 Installing a server

A new MQTT server (archive joramMQ-mqtt-**<version>**.zip) needs to be installed on every node of the cluster. A new installation should be done on every node instead of a copy of an existing installation. In particular, the directory 'data' of an existing server should never be copied when creating another server installation.

18.2 Creating the cluster

A cluster starts from an initial server, usually the server '0'. The file 'conf/joramMQ.xml' needs to be updated with the hostname (or IP address) allowing the other servers of the cluster to reach server '0'.

```
<server id="0" name="S0" hostname="<host name>">
```

If server '0' has already been started then you need to stop, update the configuration, and restart:

```
joramMQ-admin -stop
joramMQ-admin -update
joramMQ-server
```

Then, the cluster is created by calling the following command.

```
joramMQ-admin -cluster -port <port value>
```

The server needs to allocate a port number in order to accept connections from the other servers of the cluster. If the default port value is not appropriate (17700) then a port value has to be specified with the option “-port”.

The result message should be (assuming that the default identifier '0' has not been changed):

```
Cluster created, starting from server '0'.
```

Now the cluster has been created and contains a unique server.

18.3 Adding a server to the cluster

A server is added to the cluster as follows:

```
jorammq-admin -newserver -hostname <host name> -port <port value>
```

The hostname of the new server has to be given with the option “-hostname”. A server needs to allocate a port number in order to accept connections from the other servers of the cluster. If the default port value is not appropriate (17700) then a port value has to be specified with the option “-port”.

The result of the command should be:

```
New server '<server id>' added to the cluster.
```

The file 'conf/jorammq.xml' has been updated with the configuration of the new server.

In order to install the new server you need to make a new installation of JoramMQ (as explained above in 18.1) and copy the updated file 'conf/jorammq.xml' to the directory 'conf' of the new server.

The identifier of the new server has to be updated in the commands 'bin/jorammq-server' and 'bin/jorammq-admin':

```
SERVER_ID = <server id>
```

It may be useful to update the ports as specified in section 3.4.

The new server owns the same services as server 0, and these services are configured in the same way. If any changes are necessary then they should be done now, before starting the new server.

Once the commands and the configuration have been updated, you can start the new server:

```
jorammq-server
```

The following lines should appear among the output traces of the server:

```
Launching JoramMQ MQTT server <server id>
...
AgentServer.init : AgentServer#<server id>, init()
...
AgentServer.start : AgentServer#<server id>, started at...
```

18.4 Removing a server from the cluster

Removing a server may affect the MQTT clients using the cluster as there may be clients connected to the removed server. These clients will be disconnected, some published messages may be lost, and durable clients (having set their clean session flag to false) will not be able to recover their messages from another server of the cluster as client contexts are not replicated across the cluster.

Therefore, before removing a server, you should ensure that there is no more MQTT clients connected to this server.

Then you can stop the server with the following command locally executed.

```
jorammq-admin -stop
```

Finally the server can be removed from the cluster. The following command has to be launched on one of the remaining nodes of the cluster:

```
jorammq-admin -delserver <server id>
```

The result of the command should be:

```
Server '<server id>' removed from the cluster.
```

18.5 Cluster example

This section presents an example of cluster that mixes subscriptions that are replicated across the cluster, with subscriptions that are only active on a given server.

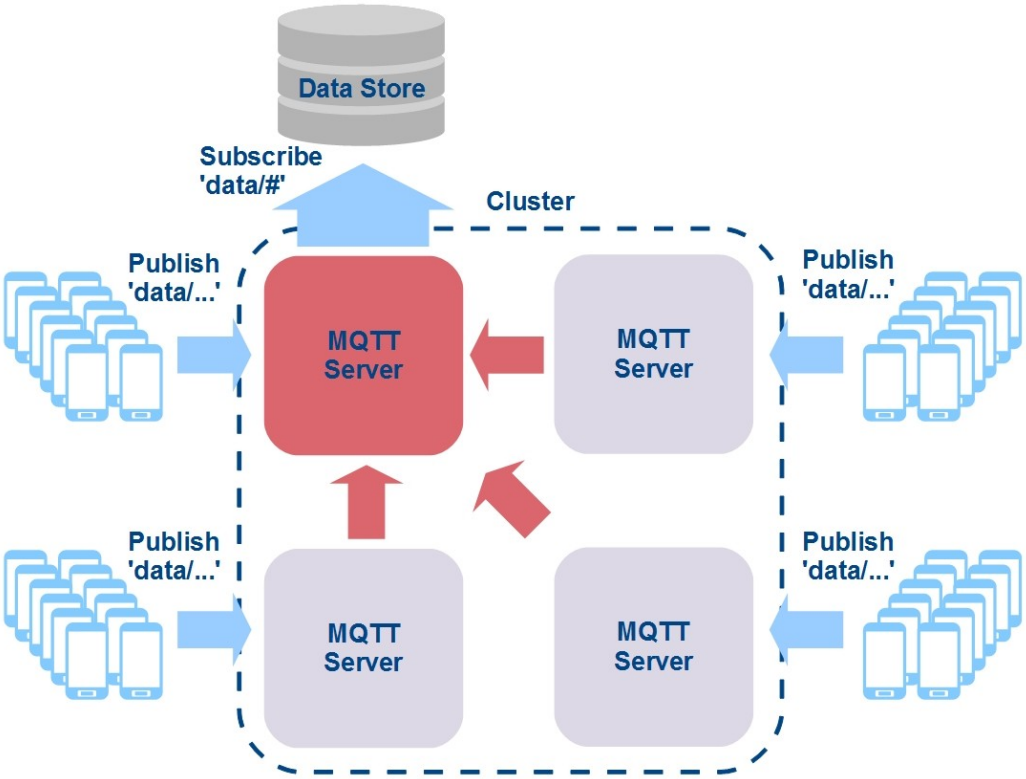

Figure 4: Leakage of horizontal scaling

In this example, there are two types of MQTT clients:

1. multiple devices (e.g. smartphones) producing data and interacting with each other
2. a single backend which stores the data produced by the devices

There are also two types of topics:

1. the topics used to transmit the data from the devices to the backend. The root topic is 'data'.

2. the topics used by the devices to communicate with each other, i.e. sending requests and receiving responses. The root topics are 'com/request' and 'com/response'.

The datastore needs to receive all the data and all the requests and responses exchanged by the devices: data/# and com/#. If those subscriptions are replicated across the cluster, then the datastore can only connect to a single server of the cluster otherwise each message is delivered once per connection. Such a configuration does not scale very well because of the leakage caused by the cross traffic between the servers (see Figure 4).The cluster works more efficiently if the datastore connects to every server. In this way, there is no more cross traffic caused by the datastore subscriptions (data/# and com/#). The remaining cross traffic only results from the communication between the devices (see Figure 5).

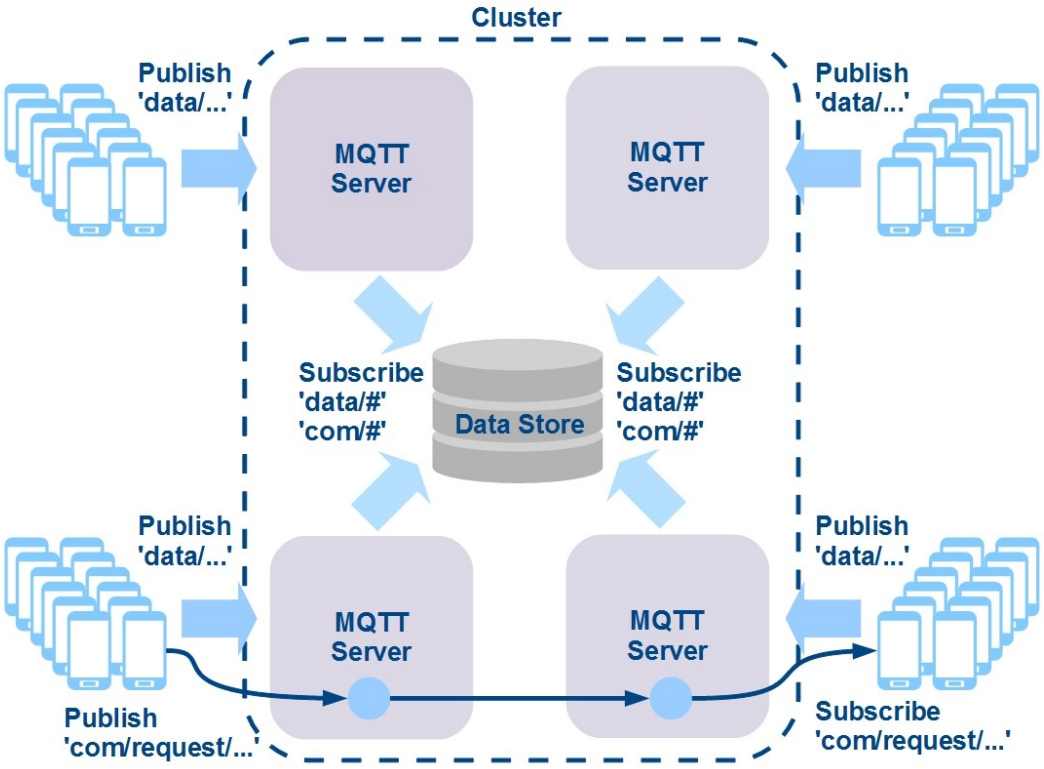

Figure 5: Cluster without leakage

This architecture is obtained thanks to the QoS property 'Replicate subscriptions' (see 5.3.20). Only the subscriptions made by the devices are replicated across the cluster. The subscriptions made by the datastore are not replicated. Two different MQTT connectors are used, one with the property 'Replicate subscriptions' set to 'true' (for devices) and the other one set to 'false' (for the datastore):

```
tcp://0.0.0.0:1883?rep.sub=true
tcp://0.0.0.0:1884?rep.sub=false
```

19 Interoperability with JMS

Interoperability requires to create subscription contexts that are common to MQTT and JMS/Joram. These subscription contexts are also called Topics in JMS/Joram.

19.1 Create a subscription context

A subscription context is created from a 1-level topic path. Allowed characters in the topic level are alphabetic characters, decimal digits and underscore (no wildcard character, + or #).

The following command creates the subscription context called “jmsTest”:

```
jorammq-admin -newtopic jmsTest
```

Once the subscription context “jmsTest” has been created, the JMS reference of the Topic “jmsTest” can be created by using the Joram administration API. For example:

```
import org.objectweb.joram.client.jms.Topic;
…
    Topic topic = Topic.create(“jmsTest”);
```

A JMS client can then use the topic either as a consumer or a producer.

```
    // Create a ConnectionFactory according to your environment
    ConnectionFactory cf = ...
    Connection cnx = cf.createConnection();
    cnx.setClientID("MqttInteropTest");
    Session session = cnx.createSession(false, Session.AUTO_ACKNOWLEDGE);
    MessageConsumer consumer = session.createDurableConsumer(topic, "MyConsumer");
    MessageProducer producer = session.createProducer(topic);
```

🛠 **Warning :** Be careful, it is very important to create the subscription context before the JMS topic.

19.1.1 Using Joram administration API

The subscription context can also be created using the Joram administration API:

```
    Properties properties = new Properties();
    properties.put(“MqttTopicName”, topicName);
    Topic topic = Topic.create(sid, topicName,
                      “com.scalagent.jorammq.mqtt.adapter.agent.MqttTopic”,
                      properties);
    topic.setFreeReading();
    topic.setFreeWriting();
```

19.1.2 Access to root topics

During initialization, JoramMQ has 2 root subscription contexts, one for the tree of "normal" subjects, the other for "special" subjects starting with the character "$". You can use these subscription contexts by retrieving the corresponding JMS topic.

19.2 MQTT publisher to JMS consumer

Every MQTT message is delivered to JMS clients as a BytesMessage.

```
BytesMessage msg = (BytesMessage) consumer.receive();
```

MQTT messages published to subtopics are also delivered to the JMS consumer. The JMS subscription to “topic” is equivalent to the MQTT subscription to “topic/#”.

The following JMS properties allow to get information about the MQTT publish:

- JMS_JORAM_MQTT_TOPIC – The topic path of the MQTT publish.
- JMS_JORAM_MQTT_QOS – The QoS of the MQTT publish.
- JMS_JORAM_MQTT_RETAIN – The retain option of the MQTT publish.

19.3 JMS producer to MQTT subscriber

Only ByteMessages can be delivered to MQTT clients with a message payload.

```
BytesMessage msg = session.createBytesMessage();
msg.writeBytes(bytes);
producer.send(msg);
```

Other JMS message types are delivered to MQTT clients with an empty payload.

MQTT subscribers can receive the messages published to the JMS topic “topic” by subscribing to “topic” or “topic/#”.

The JMS publisher can refine the topic path by assigning the JMS property JMS_JORAM_MQTT_TOPIC with the path of the MQTT topic typed as a String[30].

```
BytesMessage msg = session.createBytesMessage();
// Refine the topic path
msg.setStringProperty(“mqtt.topic”, “jmsTest/a/b”);
msg.writeBytes(bytes);
producer.send(msg);
```

The JMS publisher can fix the QoS of resulting MQTT message by assigning the JMS property JMS_JORAM_MQTT_QOS with the desired qos value typed either as an Integer or as a String. By

[30] Old property mqtt.topic is deprecated, it will be removed in future version.

default QoS is fixed to 1 (at least one) for persistent JMS messages, and 0 (at most one) for others.

The JMS publisher can fix the retain option of resulting MQTT message by assigning the JMS property JMS_JORAM_MQTT_RETAIN with the desired retain value typed either as a Boolean or as a String. By default retain option is set to false.

19.4 Delete a subscription context

If necessary, delete the subscription context “jmsTest” as follows:

```
joramMQ-admin -deltopic jmsTest
```

20 High availability

High availability is achieved by replicating JoramMQ on a cluster. Replication requires to copy a server installation on several nodes, and replicate the data store. There is no built-in mechanism to replicate the data store. The way the replication is done depends on the data store of your installation.

The default data store is based on the file system[31]. This data store is located in the directory 'data/jorammq' and can be replicated for example with a storage area network (SAN) or Linux HA (see section 20.1).

20.1 Linux HA

The following Linux HA tools allow to make a JoramMQ server highly available:

1. DRBD (Distributed Replicated Block Device)
2. Pacemaker

DRBD creates a virtual block device that can be replicated from a primary node to a secondary node. You create a file system on the virtual block device, and this information is then replicated, at the block level, from the primary node to the secondary node. DRBD can also ensure data integrity by only returning from a write operation on the primary node when the data has been written to the underlying physical block device on both the primary and secondary nodes. Section 20.2 explains how to configure JoramMQ for DRBD.

The Pacemaker program monitors the primary node and switches automatically to the secondary node in the event of failure. The secondary DRBD device becomes primary, the file system on this device is mounted, and JoramMQ is started. The original master (if still available) has its resources disabled, which means shutting down JoramMQ, unmounting the file system and switching the DRBD device to secondary. Pacemaker also manages a virtual IP address which has to be used for all communications to JoramMQ. Section 20.3 explains how to configure Pacemaker for JoramMQ.

[31]If the default data store based on the file system is not appropriate, some other data stores can be provided. Please contact us (contact@scalagent.com) to receive the list of the available data stores and the way to install them.

20.2 Configuring JoramMQ for DRBD

JoramMQ has to be installed on the primary and secondary nodes. The same procedure as installing a non-replicated server should be followed (see section 2).

Only the directory of the data store ('data/jorammq') should be moved to the DRBD device. You should not install JoramMQ entirely on the DRBD device, because such an installation would lead to replicate more data (e.g. logging traces and swapped data should not be replicated).

20.2.1 Configuring JoramMQ on the primary node

If the installed JoramMQ server is already running, you need to stop the server and copy the directory of the data store to the DRBD device and mounted file system (which is '/drbd' in the example):

```
jorammq-admin -stop
cp -R $JORAMMQ_MQTT_HOME/data/jorammq /drbd/jorammq
```

If you start from a new installation of JoramMQ then you can ignore the steps above.

The path to the data store has to be moved to the DRBD device. Set the JORAMMQ_DATA_DIR environment variable or edit the commands 'bin/jorammq-server and bin/jorammq-admin' to change the directory of the data store:

```
export JORAMMQ_DATA_DIR=/drbd/jorammq
```

Restart the server:

```
jorammq-server
```

The data store should now be located on the file system running on the DRBD device. The data is physically stored on the underlying device for the DRBD device and is replicated to the secondary DRBD node.

20.2.2 Configuring JoramMQ on the secondary node

The JoramMQ server on the secondary node reflects the server on the primary node. Therefore, both servers should have exactly the same configuration.

The file 'conf/jorammq.xml' is saved in the replicated data store so it is not necessary to copy this file on the secondary node. However, you should note that once the secondary node becomes primary, if the XML configuration needs to be updated, then it must be synchronized with the configuration saved in the data store before being updated (see section 3.13.2).

The other files in conf/ are not saved in the data store so it is necessary to copy these files from the primary node to the secondary node and update the copies if some modifications have been done. Those files include in particular:

- the password and access control files (passwd.properties, acl.xml)
- the bridge configuration file (bridge.xml)

Note that you cannot start the JoramMQ server on the secondary node, as a DRBD device working in secondary mode is not available for use.

20.3 Using Pacemaker with JoramMQ and DRBD

Pacemaker requires JoramMQ to be installed as a service (LSB or systemd) or an OCF Resource Agent.

20.3.1 Installing JoramMQ as a LSB service

JoramMQ will soon provide a service installer for Linux and Windows. Meanwhile, a simplistic template of service script for Linux is given hereafter. It needs to be copied in the file '/etc/init.d/jorammq-mqtt'.

```
#!/bin/bash

export JORAMMQ_MQTT_HOME=<JoramMQ MQTT home>
export JAVA_HOME=<Java home>
export JORAMMQ_DATA_DIR=<JoramMQ DATA dir>

MQTT_PID_FILE="$JORAMMQ_MQTT_HOME/jorammq-mqtt.pid"

cd $JORAMMQ_MQTT_HOME

do_start() {
  $JORAMMQ_MQTT_HOME/bin/jorammq-server > /dev/null 2>&1 &
}

do_stop() {
  $JORAMMQ_MQTT_HOME/bin/jorammq-admin -stop > /dev/null 2>&1
}

is_running() {
  if [ ! -f $MQTT_PID_FILE ]
  then
    return 0;
  fi
  ret=`pgrep -F $MQTT_PID_FILE`
  if [ "x$ret" == "x" ]
  then
    return 0;
```

```
  fi
  return $ret;
}

case "$1" in
start)
  echo "Starting JoramMQ MQTT in background."
  is_running
  if [ $? -eq 0 ]
  then
    do_start
  else
    echo "already running $ret"
  fi
;;
stop)
  echo "Stopping JoramMQ MQTT"
  is_running
  if [ $? -eq 0 ]
  then
    echo "already stopped"
  else
    do_stop
  fi
;;
restart)
  echo "Restart JoramMQ MQTT"
  do_stop
  do_start
;;
status)
  is_running
  if [ $? -eq 0 ]
  then
    echo "mqtt not running"
    exit 3;
  fi
  echo "mqtt running pid = $ret"
;;
*)
```

```
echo "usage: {start|stop|restart|status}" >&2
exit 3
;;
esac
```

20.3.2 Installing JoramMQ as a systemd service

A simplistic template of systemd service script for Linux is given hereafter. It needs to be copied in the file '/etc/systemd/system/jorammq-mqtt.service'.

```
[Unit]
Description=Service JoramMQ MQTT
#After=pacemaker.service

[Service]
Type=forking

Environment=JORAMMQ_MQTT_HOME=<JoramMQ MQTT home>
Environment=JAVA_HOME=<Java home>
Environment=JORAMMQ_DATA_DIR=<JoramMQ DATA dir>

ExecStart=<JoramMQ MQTT home>/bin/jorammq-server
ExecStop=<JoramMQ MQTT home>/bin/jorammq-admin -stop

#User=<JoramMQ user>
#Group=<JoramMQ group>

[Install]
WantedBy=multi-user.target
```

20.3.3 Installing JoramMQ as an OCF Resource Agent

A simplistic template of OCF script for Linux Pacemaker is given hereafter. It needs to be copied in the file '/usr/lib/ocf/resource.d/heartbeat/jorammq-mqtt'.

```
#!/bin/sh

#######################################################################
# Initialization:

: ${OCF_FUNCTIONS_DIR=${OCF_ROOT}/lib/heartbeat}
. ${OCF_FUNCTIONS_DIR}/ocf-shellfuncs
```

```
#######################################################################

NODENAME=$(ocf_local_nodename)

############# JoramMQ environement variable
if [ "x$OCF_RESKEY_JAVA_HOME" != "x" ]; then
  if [ -d "$OCF_RESKEY_JAVA_HOME" ]; then
    export JAVA_HOME=$OCF_RESKEY_JAVA_HOME
  fi
fi

if [ "x$OCF_RESKEY_JORAMMQ_MQTT_HOME" != "x" ]; then
  if [ -d "$OCF_RESKEY_JORAMMQ_MQTT_HOME" ]; then
    export JORAMMQ_MQTT_HOME=$OCF_RESKEY_JORAMMQ_MQTT_HOME
  fi
fi

if [ "x$OCF_RESKEY_JORAMMQ_DATA_DIR" != "x" ]; then
  export JORAMMQ_DATA_DIR=$OCF_RESKEY_JORAMMQ_DATA_DIR
fi

MQTT_PID_FILE="$JORAMMQ_MQTT_HOME/jorammq-mqtt.pid"

meta_data() {
cat <<END
<?xml version="1.0"?>
<!DOCTYPE resource-agent SYSTEM "ra-api-1.dtd">
<resource-agent name="ocf-jorammq-mqtt">
<version>1.0</version>

<longdesc lang="en">
This is the resource agent for the JoramMQ-MQTT server
</longdesc>
<shortdesc lang="en">Manage jorammq-mqtt server</shortdesc>

<parameters>
<parameter name="JORAMMQ_DATA_DIR" unique="1" required="1">
<longdesc lang="en">
The path to the data store has to be moved to the DRBD device.
```

```
</longdesc>
<shortdesc lang="en">set the JORAMMQ_DATA_DIR</shortdesc>
<content type="string" default="" />
</parameter>

<parameter name="JORAMMQ_MQTT_HOME" unique="1" required="1">
<longdesc lang="en">
The path to the JoramMQ MQTT home.
</longdesc>
<shortdesc lang="en">set the JORAMMQ_MQTT_HOME</shortdesc>
<content type="string" default="" />
</parameter>

<parameter name="JAVA_HOME" unique="1">
<longdesc lang="en">
The path to the java home.
</longdesc>
<shortdesc lang="en">set the JAVA_HOME</shortdesc>
<content type="string" default="" />
</parameter>
</parameters>

<actions>
<action name="start"        timeout="100s" />
<action name="stop"         timeout="60s" />
<action name="status"       timeout="30s" />
<action name="monitor"      timeout="30s" interval="10" depth="0" />
<action name="meta-data"    timeout="5s" />
<action name="validate-all" timeout="5s" />
</actions>
</resource-agent>
END
exit $OCF_SUCCESS
}
######################################################################

mqtt_usage() {
cat <<END
usage: $0 {start|stop|status|monitor|validate-all|meta-data}
```

```
Expects to have a fully populated OCF JoramMQ environment set.
END
}

mqtt_env_check() {
ocf_log debug "${NODENAME}: JORAMMQ_MQTT_HOME = $JORAMMQ_MQTT_HOME"
ocf_log debug "${NODENAME}: JAVA_HOME = $JAVA_HOME"
ocf_log debug "${NODENAME}: JORAMMQ_DATA_DIR = $JORAMMQ_DATA_DIR"

if [ "x$JAVA_HOME" != "x" ]; then
  if [ ! -d "$JAVA_HOME" ]; then
    ocf_log info "JAVA_HOME is not valid: $JAVA_HOME"
    return 1
  fi
else
  ocf_log info "JAVA_HOME is not defined"
  return 1
fi

if [ "x$JORAMMQ_MQTT_HOME" != "x" ]; then
  if [ ! -d "$JORAMMQ_MQTT_HOME" ]; then
    ocf_log info "JORAMMQ_MQTT_HOME is not valid: $JORAMMQ_MQTT_HOME"
    return 1
  fi
else
  ocf_log info "JORAMMQ_MQTT_HOME is not defined"
  return 1
fi

if [ "x$JORAMMQ_DATA_DIR" == "x" ]; then
  ocf_log info "JORAMMQ_DATA_DIR is not defined"
  return 1
fi
ocf_log info "JoramMQ environement variable is valid."
return 0
}

is_running() {
  if [ ! -f ${MQTT_PID_FILE} ]
  then
```

```
    return 1;
  fi
  ret=`pgrep -F ${MQTT_PID_FILE}`
  if [ "x$ret" == "x" ]
  then
    return 1;
  fi
  return 0;
}

do_start() {
  $JORAMMQ_MQTT_HOME/bin/jorammq-server
}

do_stop() {
  $JORAMMQ_MQTT_HOME/bin/jorammq-admin -stop
}

mqtt_monitor() {
        mqtt_env_check
        local rc
        is_running > /dev/null 2>&1
        rc=$?
        case "$rc" in
        0)
                ocf_log debug "${NODENAME}: JoramMQ server is running normally"
                return $OCF_SUCCESS
        ;;
        1)
                ocf_log debug "${NODENAME}: JoramMQ server is not running"
                return $OCF_NOT_RUNNING
        ;;
        *)
                ocf_log err "${NODENAME}: Unexpected return code: $rc"
                return $OCF_ERR_GENERIC
        ;;
        esac
}

mqtt_start() {
```

```
  ocf_log info "${NODENAME}: JoramMQ mqtt start... ${OCF_RESOURCE_INSTANCE}"
        local rc
        mqtt_monitor
        if [ $? -eq $OCF_SUCCESS ]; then
                return $OCF_SUCCESS
        fi

        # Start JoramMQ server
        do_start

        mqtt_monitor
        if [ $? -eq $OCF_SUCCESS ]; then
                return $OCF_SUCCESS
        else
            ocf_log info "JoramMQ not started !"
            return $OCF_ERR_GENERIC
        fi
        return $OCF_SUCCESS
}

mqtt_stop() {
  ocf_log info "${NODENAME}: JoramMQ mqtt stop... ${OCF_RESOURCE_INSTANCE}"
        mqtt_monitor
        if [ $? -eq $OCF_NOT_RUNNING ]; then
                return $OCF_SUCCESS
        fi
        do_stop
        #TODO add kill logic
        return $OCF_SUCCESS
}

mqtt_validate() {
        check_binary ${JORAMMQ_MQTT_HOME}/bin/jorammq-server
        check_binary ${JORAMMQ_MQTT_HOME}/bin/jorammq-admin
        return $OCF_SUCCESS
}

case "$__OCF_ACTION" in
meta-data)      meta_data;;
start)          mqtt_start;;
```

```
stop)           mqtt_stop;;
status)         mqtt_monitor;;
monitor)        mqtt_monitor;;
validate-all)   mqtt_validate;;
usage|help)     mqtt_usage
                exit $OCF_SUCCESS
                ;;
*)              mqtt_usage
                exit $OCF_ERR_UNIMPLEMENTED
                ;;
esac
rc=$?
ocf_log debug "${OCF_RESOURCE_INSTANCE} $__OCF_ACTION : $rc"
exit $rc
```

21 Plugins

21.1 Overview

JoramMQ provides plugin interfaces for the following usages:

1. Access control (see section 4).
2. Enhanced authentication
3. client listener: listens to events related to the MQTT clients connected to the server.

Only the Java language (or related) can be used to implement a plugin.

The table below lists the Java interfaces defined for every plugin.

Plugin	Interface
Access control	com.scalagent.jorammq.mqtt.MqttAccessControl
Authentication	com.scalagent.jorammq.mqtt.authentication.AuthPlugin com.scalagent.jorammq.mqtt.authentication.AuthSession
Client listener	com.scalagent.jorammq.mqtt.MqttClientListener

The directory 'javadoc' contains the Java documentation of these interfaces.

For every plugin, two main classes need to be implemented:

1. The class that implements the plugin interface
2. An OSGi activator that instantiates the plugin class and registers the instance to the MQTT server

The plugin has to be packaged and installed as an OSGi bundle.

The project 'jorammq-mqtt-examples' gives examples of plugins built with Maven. Each example can be used as a template to build your own plugin. The project is contained in the following archive:

```
jorammq-mqtt-examples-assembly-<version>.zip
```

The OSGi configuration file, “conf/felix.properties”, has to be updated.

Only one plugin should be installed for access control and client listener. Multiple authentication bundles can be added to JoramMQ. Each bundle can register several plugins, each plugin manages a specific enhanced authentication protocol (referenced by its corresponding “auth method”).

The jar file of plugin (OSGi bundle) has to be declared after 'jorammq-mqtt-osgi.jar' and before 'a3-osgi.jar'.

```
  ...
  file:./bundle/jorammq-mqtt-osgi.jar \
  file:./bundle/<your access control plugin>.jar \
  file:./bundle/<your client listener plugin>.jar \
  file:./bundle/a3-osgi.jar \
  …
```

And the bundle cache has to deleted[32]:

```
rm -rf data/felix
```

21.2 Connection context

Since JoramMQ 1.10 a connection context is given in parameter of the checkConnect and/or onConnect methods (see below). This connection context implements the MqttConnection interface as defined below:

```
/**
 * MQTT Connection context.
 */
public interface MqttConnection {
     public SocketAddress getRemoteAddress();
     public String getRemoteHostAddress();
}
```

21.3 Access control

21.3.1 Access control implementation

The methods doing access control must not block and should respond as fast as possible. If any remote call or synchronization is needed, then it must be done by a concurrent activity updating a cache.

```
import com.scalagent.jorammq.mqtt.MqttAccessControl;

public class AccessControlExample implements MqttAccessControl {

  public void init() throws Exception {
    …
  }
```

[32] Now the cache is deleted by default every time JoramMQ is started.

```
  public boolean checkConnect(CONNECT connect,
      X509Certificate[] clientCertificateChain,
      MqttConnection context) throws Exception {
    …
  }

  public boolean checkSubscribe(MqttTopicFilter[] subscriptionTopicFilters,
      SUBSCRIBE subscribe, CONNECT connect,
      X509Certificate[] clientCertificateChain) throws Exception {
    …
  }

  public boolean checkPublish(MqttTopicPath publicationTopicPath, PUBLISH publish,
      CONNECT connect, X509Certificate[] clientCertificateChain)
      throws Exception {
    …
  }

  public void close() throws Exception {
    …
  }
}
```

21.3.2 Activator

The OSGi activator resolves the reference of the MQTT server, instantiates the access control class and registers the instance.

```
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.util.tracker.ServiceTracker;

import com.scalagent.jorammq.mqtt.MqttAccessControl;
import com.scalagent.jorammq.mqtt.MqttServer;

public class AccessControlActivator implements BundleActivator {

  private MqttServer mqttServer;

  public void start(BundleContext context) throws Exception {
    ServiceTracker mqttServerTracker = new ServiceTracker(context,
```

```
        MqttServer.class.getName(), null);
    mqttServerTracker.open();
    mqttServer = (MqttServer) mqttServerTracker.waitForService(0);
    MqttAccessControl accessControl = new AccessControlExample();
    mqttServer.setAccessControl(accessControl);
  }

  public void stop(BundleContext context) throws Exception {}
}
```

21.4 Client listener

A client listener is notified when the following events occur:

Event	Description
AuthenticationFailure	A client fails to authenticate.
Connect	A client connects.
ConnectionClosed	A client's connection is closed. A connection is closed when the client disconnects, or after a failure (connection, authentication, protocol), or if the client is rejected by another client, or if the client has not sent a PINGREQ in time.
ConnectionFailure	An exception is raised at the connection level and the connection is closed.
Disconnect	A client disconnects.
JmxClose	A client's connection is closed by a JMX command.
MissingPingReq	A client has not sent the PINGREQ in time.
ProtocolFailure	A client does not comply with MQTT protocol rules.
Publish	A client publishes a message.
Reject	A client is rejected by another MQTT client with the same client id.
Subscribe	A client subscribes.
Unsubscribe	A client unsubscribes.
WillPublished	The Will message specified by a client is published.

21.4.1 Client listener implementation

The listener's methods must not block and should respond as fast as possible. If any remote call or synchronization is needed, then it must be done by a concurrent activity.

```
import com.scalagent.jorammq.mqtt.MqttClientListener;
```

```
public class ClientListenerExample implements MqttClientListener {

  public void init() throws Exception {
    …
  }

  public void onConnect(CONNECT connect,
      X509Certificate[] clientCertificateChain,
      MqttConnection context) {
    …
  }

  public void onSubscribe(MqttTopicFilter[] subscriptionTopicFilters,
      SUBSCRIBE subscribe, CONNECT connect,
      X509Certificate[] clientCertificateChain) {
    …
  }

  public void onPublish(MqttTopicPath publicationTopicPath, PUBLISH publish,
      CONNECT connect, X509Certificate[] clientCertificateChain) {
    …
  }

  public void onUnsubscribe(MqttTopicFilter[] unsubscriptionTopicFilters,
      UNSUBSCRIBE unsubscribe, CONNECT connect,
      X509Certificate[] clientCertificateChain) {
    …
  }

  public void onDisconnect(DISCONNECT disconnect, CONNECT connect,
      X509Certificate[] clientCertificateChain) {
    …
  }

  public void onConnectionFailure(IOException exception, CONNECT connect,
      X509Certificate[] clientCertificateChain) {
    …
  }
```

```
  public void onAuthenticationFailure(String failureReason, CONNECT connect,
      X509Certificate[] clientCertificateChain) {
    …
  }

  public void onProtocolFailure(String failureReason, Throwable error,
      CONNECT connect, X509Certificate[] clientCertificateChain) {
    …
  }

  public void onReject(CONNECT connect,
     X509Certificate[] clientCertificateChain) {
    …
  }

  public void onMissingPingReq(CONNECT connect,
      X509Certificate[] clientCertificateChain) {
    …
  }

  public void onJmxClose(CONNECT connect,
      X509Certificate[] clientCertificateChain) {
    …
  }

  public void onConnectionClosed(CONNECT connect,
      X509Certificate[] clientCertificateChain) {
    …
  }

  public void onWillPublished(CONNECT connect,
      X509Certificate[] clientCertificateChain) {
    …
  }

  public void close() throws Exception {
    …
  }
}
```

21.4.2 Activator

The OSGi activator resolves the reference of the MQTT server, instantiates the client listener and registers the instance.

```
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.util.tracker.ServiceTracker;

import com.scalagent.jorammq.mqtt.MqttServer;

public class ClientListenerActivator implements BundleActivator {

  private ClientListenerExample clientListener;

  public void start(BundleContext context) throws Exception {
    ServiceTracker mqttServerTracker = new ServiceTracker(context,
        MqttServer.class.getName(), null);
    mqttServerTracker.open();
    MqttServer mqttServer = (MqttServer) mqttServerTracker.waitForService(0);
    clientListener = new ClientListenerExample();
    mqttServer.setClientListener(clientListener);
  }

  public void stop(BundleContext context) throws Exception {}
}
```

21.5 Authentication

This plugin allows to implement new authentication methods corresponding to the enhanced authentication defined in the MQTTv5 specification. Implementing such a method needs a detailed reading of the relevant sections of the specification, specifically:

- Section 4.12 “Enhanced authentication”.
- Sections about the frames involved in the authentication process: 3.1-CONNECT, 3.2-CONNACK, 3.15-AUTH, and 3.14-DISCONNECT.

21.5.1 Simple authentication implementation

The authentication plugin is invoked (method initiate) each time a new client connection is established. It returns an authentication session to handle all authentication events during the connection lifetime. The authentication session is notified when the following events occur:

- CONNECT frame when the client session starts.
- AUTH frame during the initial authentication, then when the client triggers re-authentication phases.

The interfaces and classes necessary for the implementation of an authentication plugin are defined in the package "*com.scalagent.jorammq.mqtt.authentication*".

21.5.2 AuthPlugin interface

The AuthPlugin interface is used to initiate an enhanced authentication session. The new authentication session is returned by the initiate method. The authentication session must implement the *AuthSession* interface.

```
/** Plugin interface for the MQTT enhanced authentication. */
public interface AuthPlugin {
  /**
   * Initiates a new Session.
   *
   * @return  the Authentication session.
   * @throws Exception An error occurs during plugin session creation.
   */
  AuthSession initiate();
}
```

21.5.3 AuthSession interface

This session is associated with the connection and remains active throughout its lifetime. It is used to manage the initial authentication and then any re-authentication phases. It must maintain the authentication state for the current session. The session is notified when the following events occur:

- The *onConnect* method is called to process the CONNECT frame.
- The *onAuth* method is called for each AUTH frame send by the client.

These methods must perform all the checks necessary for the enhanced authentication process. They return a data structure (class AuthData) indicating to the broker the rest of the authentication process. This structure includes 3 fields:

- a return code determining the continuation of the process: continue, succeed, not authorized, bad authentication (see section 4.12 of the MQTTv5 specification).
- The reason associated with this response. This Reason String is a human readable string designed for diagnostics. It will be transmitted to the client with the response frame.
- The authentication data to be returned to the client. The contents of this data are defined by the authentication method and the state of already exchanged authentication data.

These methods must not block and should respond as fast as possible.

```
import com.scalagent.jorammq.mqtt.codec.AUTH;
import com.scalagent.jorammq.mqtt.codec.CONNECT;

/**
 * Authentication session interface.
 */
public interface AuthSession {
  /**
   * Handles the CONNECT packet during an enhanced authentication process.
   * The session must verify the validity of "Authentication method", process the
   * client data in "Authentication data" if any, and returns a Data object
   * allowing the broker to build the corresponding CONNACK or AUTH packet.
   *
   * Be careful, if the authentication mechanism delivers an identity different
   * from that recorded in the userName field of the CONNECT frame, then the
   * authUserName field must reflect this new identity.
   *
   * @param connect The incoming CONNECT packet from client.
   * @return  the data needed to build the AUTH or CONNACK packet.
   */
  AuthData onConnect(CONNECT connect);

  /**
   * Handling of AUTH packet during an enhanced authentication process.
   * The session must verify the validity of "Authentication method", process
   * the client data in "Authentication data" if any, and returns a Data object
   * allowing the broker to build the corresponding CONNACK or AUTH packet.
   *
   * Be careful, if the authentication mechanism delivers an identity different
   * from that recorded in the userName field of the CONNECT frame, then the
   * authUserName field must reflect this new identity.
   *
   * @param connect The incoming CONNECT packet from client.
   * @param auth The incoming AUTH packet from client.
   * @return  the data needed to build the AUTH or CONNACK packet.
   */
  AuthData onAuth(CONNECT connect, AUTH auth);
}
```

21.5.4 AuthData class

This class offers the static methods allowing to build the objects corresponding to the different types of possible responses to the invocations of the authentication session.

```
/** Authentication Data class. */
public final class AuthData {
  /**
   * Returns a Data object to reflect authentication success.
   *
   * @param data the "Authentication data" for the AUTH or CONNACK packet.
   * @return a Data object to reflect authentication success.
   */
  public static AuthData authSucceed(byte[] data) {…}

  /**
   * Returns a Data object asking to continue authentication process.
   *
   * @param data the "Authentication data" for the AUTH or CONNACK packet.
   * @return a Data object to reflect authentication request to continue.
   */
  public static AuthData authContinue(byte[] data) {…}

  /**
   * Returns a Data object to reflect authentication failure.
   *
   * @param data the "Authentication data" for the AUTH or CONNACK packet.
   * @param reasonString the "Reason string" for the CONNACK packet.
   * @return a Data object to reflect authentication failure.
   */
  public static AuthData badAuthentication(byte[] data, String reasonString) {…}

  /**
   * Returns a Data object to reflect authentication failure.
   *
   * @param data the "Authentication data" for the AUTH or CONNACK packet.
   * @param reasonString the "Reason string" for the CONNACK packet.
   * @return a Data object to reflect authentication failure.
   */
  public static AuthData notAuthorized(byte[] data, String reasonString) {…}
}
```

21.5.5 Activator

The activator allows to register one or more enhanced authentication plugins. Each plugin is registered with its corresponding “authentication method”.

```
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.util.tracker.ServiceTracker;

import com.scalagent.jorammq.mqtt.MqttServer;
import com.scalagent.jorammq.mqtt.authentication.AuthPlugin;

/**
 * Activator registering one or more Authentication plugins.
 */
public class AuthenticationActivator implements BundleActivator {
  @Override
  public void start(BundleContext context) throws Exception {
    ServiceTracker mqttServerTracker = new ServiceTracker(context,
MqttServer.class.getName(), null);
    mqttServerTracker.open();
    MqttServer mqttServer = (MqttServer) mqttServerTracker.waitForService(0);

    // Registers the plugin
    AuthPlugin plugin = new AuthenticationExamplePlugin();
    mqttServer.registerAuthPlugin("SimpleAuth", plugin);
  }

  @Override
  public void stop(BundleContext context) throws Exception {}
}
```

22 Analytics

22.1 Generic monitoring tasks

The JoramMQ Monitoring service allows to periodically watch all public MBean's attributes registered on the server. There are different monitoring task type.

22.1.1 Installation

The JoramMQ Monitoring service is shipped as an OSGi bundle (joram-monitoring.jar in the bundle directory). This bundle is not in the default configuration and you need to add it to the list of installed bundles in felix.properties configuration file.

Then you need to stop the server, delete Felix bundle cache 'data/felix/', and restart the server.

22.1.2 Configuration

The JoramMQ Monitoring service needs a XML configuration file to describe the monitoring tasks. By default this file is *conf/monitoring.xml* in the JoramMQ installation directory. The pathname of this file can be configured by the *com.scalagent.monitoring.LOG_CONFIG_PATH* property via the Felix properties file in the conf directory.

There are different monitoring task type:

- FileMonitoringTimerTask: save monitored data on a file in CSV format.
- LogMonitoringTimerTask: write monitored data on the log file depending on the log level.
- WindowMonitoringTimerTask: display monitored data on a window (use only for debug).

To configure a monitoring task, you must use an XML script which respect the DTD bellow:

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT MonitoringTimerTasks (Task)>

<!ELEMENT Task (properties* | monitor*)>
<!ATTLIST Task name CDATA #REQUIRED >
<!ATTLIST Task type CDATA #REQUIRED >
<!ATTLIST Task period CDATA >

<!ELEMENT property (EMPTY)>
```

```
<!ATTLIST property name CDATA #REQUIRED >
<!ATTLIST property value CDATA #REQUIRED >

<!ELEMENT monitor (EMPTY)>
<!ATTLIST monitor mbean CDATA #REQUIRED >
<!ATTLIST monitor attribute CDATA #REQUIRED >
```

- A “Task” element describes a monitoring task:
  - “name”: the name used to register the associated MBean.
  - “type”: indicate the monitoring task type. You must specify the entire class name.
  - “period”: indicate the time in ms between two value checking.
- A “property” element inside a “Task” element describes a specific property (name, value) of the MonitoringTimerTask.
- A “monitor” element inside a “Task” element describes an MBean attribute(s) to monitor.
  - “mbean”: the Mbean name.
  - “attribute”: specify the attribute or list of attributes to monitor[33].

FileMonitoringTimerTask

This type of task records monitored values on a CSV file with semicolon separator. To configure it, you need to indicate “fr.dyade.aaa.common.monitoring.FileMonitoringTimerTask” as type of the monitoring task. You also have to specify the output file path through the property named “result.path”.

The output file can be read with a text editor or a spreadsheet. The first column contains the date of the record and following columns come in pairs: the first is the name of the attribute and the second its value.

LogMonitoringTimerTask

This type of task records monitored values on the server's log file depending on the log level. To configure it, you need to indicate “fr.dyade.aaa.common.monitoring.LogMonitoringTimerTask” as type of the monitoring task.

You also need to specify:

- the name of the logger (property “result.logger”), by default “fr.dyade.aaa.Monitoring”.
- the logging level (property “log.level”), by default WARNING.
- the message to display at the beginning of each record (property “log.message”).

[33] You can specify many attributes of the same MBean, either in the same element or in different elements with the same MBean name.

WindowMonitoringTimerTask

This type of monitoring task display requested data on a GUI window. This window allow you to choose dynamically attributes you want to monitor by adding, or deleting attribute. To configure it you need to indicate “fr.dyade.aaa.common.monitoring.WindowMonitoringTimerTask” as type of the monitoring task.

22.1.3 Exemple

The configuration given in example below creates two monitoring tasks:

- first task, named “task1” stores monitored values in a CSV File.
- Second task, named “task2” stores monitored values in the JoramMQ log files.

```
<MonitoringTimerTasks>
  <Task name="task1"
        type="fr.dyade.aaa.common.monitoring.FileMonitoringTimerTask"
        period="30000">
    <property name="result.path" value="MonitoringStats-1.csv" />
    <monitor mbean="AgentServer:server=AgentServer#0,cons=Transaction"
             attribute="*" />
    <monitor mbean = "AgentServer:server=AgentServer#0,cons=Engine#0"
             attribute="NbWaitingMessages" />
  </Task>

  <Task name="task2"
        type="fr.dyade.aaa.common.monitoring.LogMonitoringTimerTask"
        period="30000">
    <property name="result.logger" value="fr.dyade.aaa.Monitoring" />
    <property name="log.message" value="**Monitoring**" />
    <property name="log.level" value="MonitoringStats-1.csv" />
    <monitor mbean="AgentServer:server=AgentServer#0,cons=Transaction"
             attribute="*" />
    <monitor mbean = "AgentServer:server=AgentServer#0,cons=Engine#0"
             attribute="NbWaitingMessages" />
  </Task>
</MonitoringTimerTasks>
```

22.2 Specific monitoring plugin

This section provides an example of an analytics module that queries JMX attributes, processes their values and publishes MQTT messages to specific topics, e.g. $SYS topics.

In the same ways as plugins (see section 21), you need to implement an OSGi bundle with an activator, install it in the directory 'bundle' and declare it in 'conf/felix.properties' at the end of the list of bundles.

```
  ...
  file:./bundle/jorammq-ssh.jar \
  file:./bundle/<your analytics module>.jar
  …
```

And the bundle cache has to deleted:

```
rm -rf data/felix
```

22.2.1 Analytics task example

An analytics task is simply a TimerTask (java.util) that periodically queries JMX attributes.

```
import java.util.TimerTask;

import com.scalagent.jorammq.mqtt.JmxHelper;
import com.scalagent.jorammq.mqtt.MqttServer;
import com.scalagent.jorammq.mqtt.MqttTopicPath;

public class AnalyticsTaskExample extends TimerTask {
  …
```

This example monitors two JMX attributes of a single MQTT client: “ToDeliverMessageCount”, and the flag “Connected”. The JmxHelper class provides some constants and also methods like 'createClientName' to obtain the JMX ObjectName of a client MBean. You can find the Java documentation of JmxHelper in the directory 'javadoc'.

```
  private MqttServer mqttServer;
  private String clientId;
  private ObjectName clientJmxName;
  private List<String> attributeList;
  private List<MqttTopicPath> topicPathList;

  public AnalyticsTaskExample(MqttServer mqttServer) throws Exception {
    this.mqttServer = mqttServer;
    clientId = "MQTT client to monitor";

    attributeList = new ArrayList<String>();
    attributeList.add(JmxHelper.JMX_ATT_CLIENT_TO_DELIVER_MESSAGE_COUNT);
    attributeList.add(JmxHelper.JMX_ATT_CLIENT_CONNECTED);

    topicPathList = new ArrayList<MqttTopicPath>(attributeList.size());
```

```
    for (String att : attributeList) {
      StringBuffer buf = new StringBuffer();
      buf.append("$SYS/monitor/");
      buf.append(clientId);
      buf.append('/');
      buf.append(att);
      topicPathList.add(MqttTopicPath.parse(buf.toString()));
    }
  }
```

The client MBean is enabled in a method 'start' (called by the OSGi bundle activator). If the client has not connected yet, then 'enableClientMBean' fails and will be retried periodically by the TimerTask.

```
  void start() throws Exception {
    if (!mqttServer.isStarted())
      throw new Exception("MQTT server not started");
    String hostname = mqttServer.getHostname();
    clientJmxName = JmxHelper.createClientName(hostname, clientId);
    enableClientMBean();
  }
```

In the TimerTask method 'run', JMX attributes are queried and their values are printed and published to an MQTT topic. If the client MBean is not found, then 'enableClientMBean' is retried.

```
  @Override
  public void run() {
    try {
      for (int i = 0; i < attributeList.size(); i++) {
        printAttributeValue(attributeList.get(i), topicPathList.get(i));
      }
    } catch (InstanceNotFoundException exc) {
      System.err.println("InstanceNotFound: " + clientId);
      enableClientMBean();
    } catch (Exception exc) {
      System.err.println("AnalyticsTaskExample: " + exc);
    }
  }

  private void enableClientMBean() {
    // Enable the MBean of a particular MQTT client
    // The hostname is contained in the JMX ObjectName
    try {
```

```
      String hostname = mqttServer.getHostname();
      JmxHelper.enableClientMBean(hostname, clientId);
    } catch (Exception exc) {
      System.err.println("AnalyticsTaskExample: " + exc);
    }
  }

  private void printAttributeValue(String attName, MqttTopicPath topicPath)
    throws Exception {
    Object value = JmxHelper.getAttribute(clientJmxName, attName);

    // Print value
    System.out.println(attName + ": " + value);

    if (value != null) {
      // Publish value to a $SYS topic with a UTF-8 encoding
      mqttServer.publishMqttMessage(topicPath, value.toString().getBytes());
    }
  }
```

Finally a method 'stop' allows to disable the monitored MBean.

```
  void stop() throws Exception {
    // Disable the MBean of a particular MQTT client
    // The hostname is contained in the JMX ObjectName
    JmxHelper.disableClientMBean(mqttServer.getHostname(), clientId);
  }
```

22.2.2 Activator

The OSGi bundle activator is in charge of instantiating and scheduling the analytics task.

The reference of the Timer is returned by the MqttServer.

```
public class AnalyticsActivator implements BundleActivator {

  private AnalyticsTaskExample analyticsTask;

  public void start(BundleContext context) throws Exception {
    ServiceTracker mqttServerTracker = new ServiceTracker(context,
        MqttServer.class.getName(), null);
    mqttServerTracker.open();
    MqttServer mqttServer = (MqttServer) mqttServerTracker.waitForService(0);
```

```
    analyticsTask = new AnalyticsTaskExample(mqttServer);
    analyticsTask.start();

    // Schedule an analytics report every 10s.
    mqttServer.getTimer().schedule(analyticsTask, 10000, 10000);
  }

  public void stop(BundleContext context) throws Exception {
    if (analyticsTask != null) {
      analyticsTask.cancel();
      analyticsTask.stop();
    }
  }

}
```

23 HawtIO Web console

A HawtIO plugin[34], “jorammq-hawtio.war”, is delivered with JoramMQ. It provides access to a JoramMQ monitoring console via a web browser.

HawtIO console runs in a web front-end and accesses every MQTT server through a JSON/HTTP connection made to an embedded Jolokia agent (see section 23.1 below). In this way, a single console monitors and controls several MQTT servers.

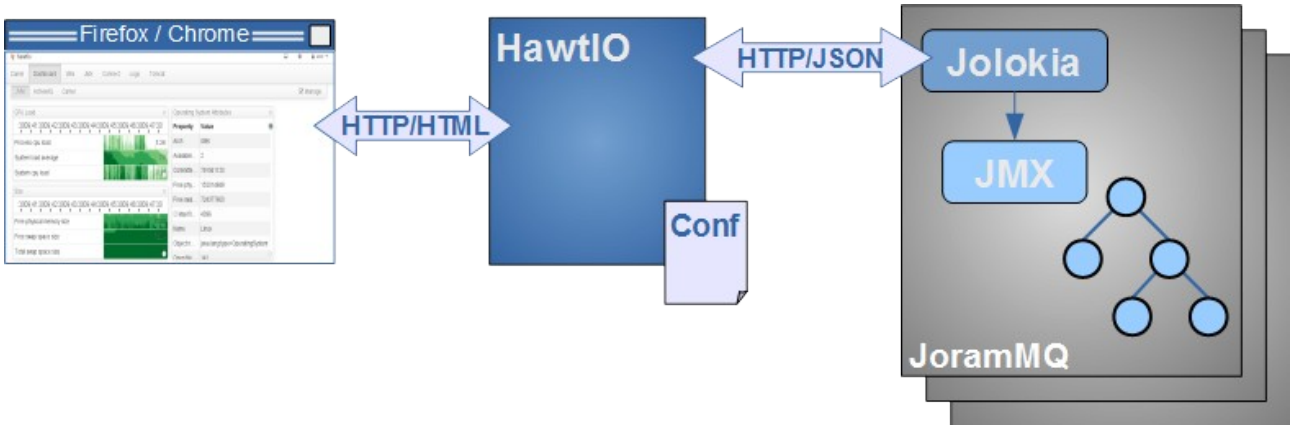

Figure 6: HawtIO / Jolokia architecture

Depending on its use, the Jolokia plugin may decrease the performance of the MQTT server. It should therefore be used with care in production. The use of a JMX console instead of this plugin should then be considered.

23.1 Installation of the Jolokia agent

JoramMQ servers are reachable through HawtIO only if they own an active Jolokia agent. Check that the JoramMQ servers have a Jolokia bundle installed and activated.

By default the Jolokia agent is not activated, in that case, you need to modify the “conf/felix.properties” file and add the Jolokia bundle to the list of installed bundles as follows:

```
felix.auto.start.1= \
  file:./bundle/monolog.jar \
  ...
  file:./bundle/jorammq-ssh.jar \
  file:./bundle/jolokia-osgi-bundle.jar
```

Be careful the jolokia bundle starts the jetty web server, so you have to remove the jetty.jar in the list of installed bundles to avoid a warning.

In this same file you must also uncomment and configure the Jolokia parameters for this server:

[34] HawtIO is a modular web console for managing Java stuff.

```
  org.osgi.service.http.port=7050
  org.jolokia.user admin
  org.jolokia.password adminpass
  org.jolokia.agentContext /jolokia
```

Then you need to stop the server, delete Felix bundle cache 'data/felix/', and restart the server[35].

The Jolokia agent should start and display the following messages:

```
  [INFO] servlet_1: No access restrictor found at classpath:/jolokia-access.xml,
access to all MBeans is allowed
  [INFO] Started jetty 6.1.x at port(s) HTTP:7050
```

23.1.1 Direct connection to the Jolokia agent

Jolokia is a remote JMX over HTTP bridge that exposes JMX Mbeans through a REST API. Once deployed it allows the access to the Mbean attributes through HTTP. For a full guide as to how to use, you can refer to the Jolokia documentation or use HawtIO as described below. However, you can also use a simple browser to query a Mbean's attribute, for example the URL below returns the number of connected clients:

```
http://localhost:7050/jolokia/read/
com.scalagent.jorammq:host=localhost,manager=ClientManager/ConnectCount
```

After authentication[36] this would give you back something like the following:

```
{"request":
{"mbean":"com.scalagent.jorammq:host=localhost,manager=ClientManager","attribute":
"ConnectCount","type":"read"},"value":0,"timestamp":1455548959,"status":200}
```

Note: You can use the "wget" utility to test this easily. Simply execute a command like this:

```
wget --user admin --password adminpass
http://192.168.1.101:7050/jolokia/read/com.scalagent.jorammq:host=localhost,manage
r=ClientManager/ConnectCount
-----
{
  "request": {
      "mbean":"com.scalagent.jorammq:host=localhost,manager=ClientManager",
      "attribute":"ConnectCount",
      "type":"read"
    },
  "value":0,
  "timestamp":1455549378,
  "status":200
}
```

[35] It resets only the code cache and there is no data lost.
[36] By default admin/adminpass as configured in the Felix configuration file.

23.2 Installation of the web front-end

Hawtio consists of 2 parts: an AngularJS application and a Java backend, which proxies the communication between the frontend and Jolokia endpoints. The frontend has access to all JMX attributes and operations available in Java applications running locally and remotely.

There are several ways to launch it:

- Running an executable JAR (described below).
- Running a Spring Boot app.
- Deploying on a Servlet container.
- Deploying on an application server.
- etc.

You can find details about these different modes in the HawtIO documentation:

- https://hawt.io/docs/get-started

23.2.1 Running an executable JAR

The simplest way to use the JoramMQ HawtIO Web console is to start HawtIO using the hawtio-app executable JAR:

- Download the HawtIO executable jar:
  - https://oss.sonatype.org/content/repositories/public/io/hawt/hawtio-app/1.5.11/hawtio-app-1.5.11.jar
- Make a directory named “plugins” and copy the “jorammq-hawtio.war” file from the JoramMQ installation.
- Just run this from the command line: java -jar hawtio-app-1.5.11.jar
- Open a web browser and enter the URL: http://<hawtio host>:8080/hawtio

23.2.2 Connect to the MQTT server

Back in the web browser, there is a 'Connect' tab on the top panel of the HawtIO screen. Upon clicking it, you can enter in host, port and path informations corresponding to the Jolokia agent owned by your JoramMQ server.

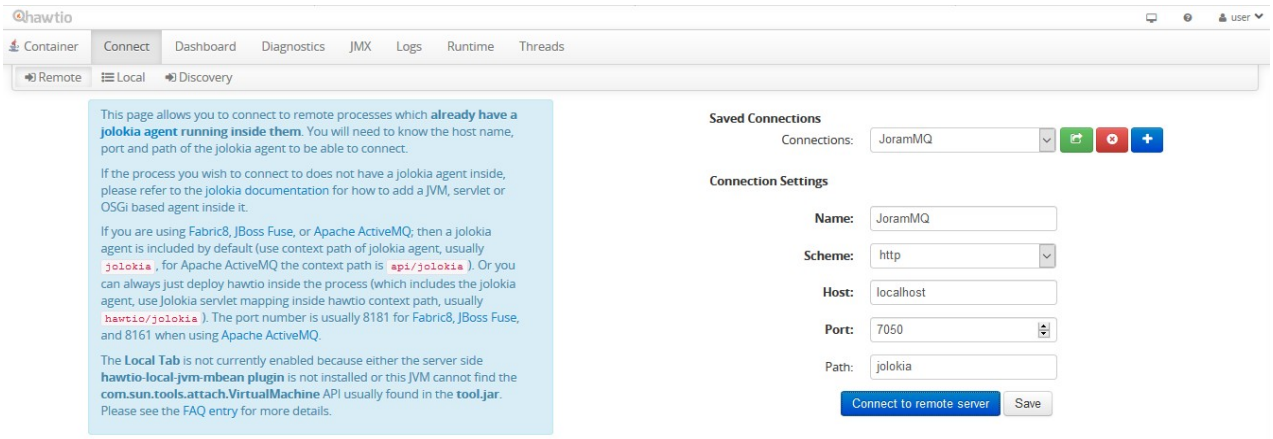

Figure 7 - HawtIO Connect tab

Then click on the button “Connect to remote server” to connect the HawtIO console to the Jolokia agent. A user name and a password may be required depending on JoramMQ configuration (see “jorammq/conf/felix.properties”, see section 23.1 above).

23.2.3 JoramMQ tab

Once connected to a JoramMQ server, you can click the 'JoramMQ' tab to obtain a synthetic status of the server (see Figure 8). Two additional tabs provide specific informations about JMS and MQTT connectors.

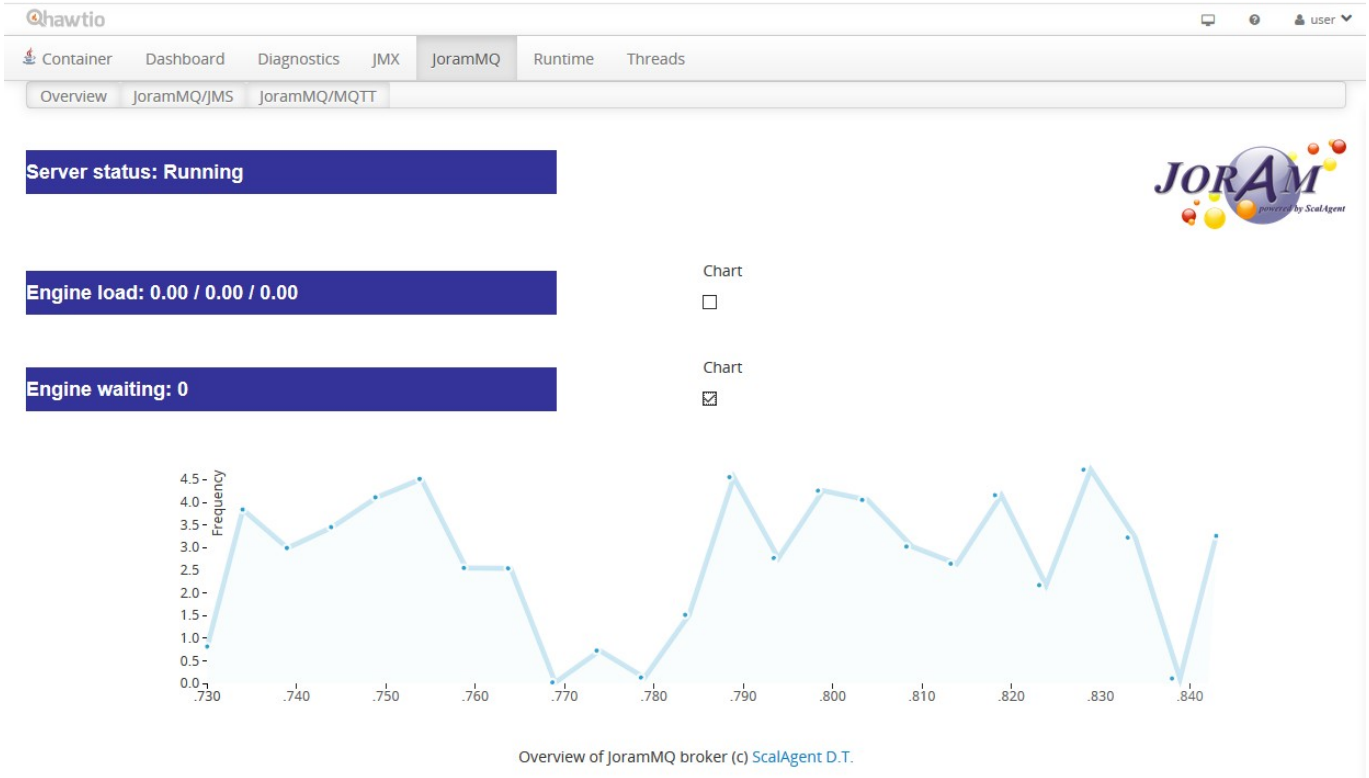

Figure 8 - JoramMQ HawtIO Overview tab

The Overview tab (Figure 8) displays:

- Server status
- Average server load
  - Average number of tasks waiting in the server execution queue during the last minute, the last 5 minutes and the last 15 minutes.
  - Optionally a curve is available for the average load during the last minute.
- Number of waiting tasks in the server execution queue.
  - Optionally a curve is available.

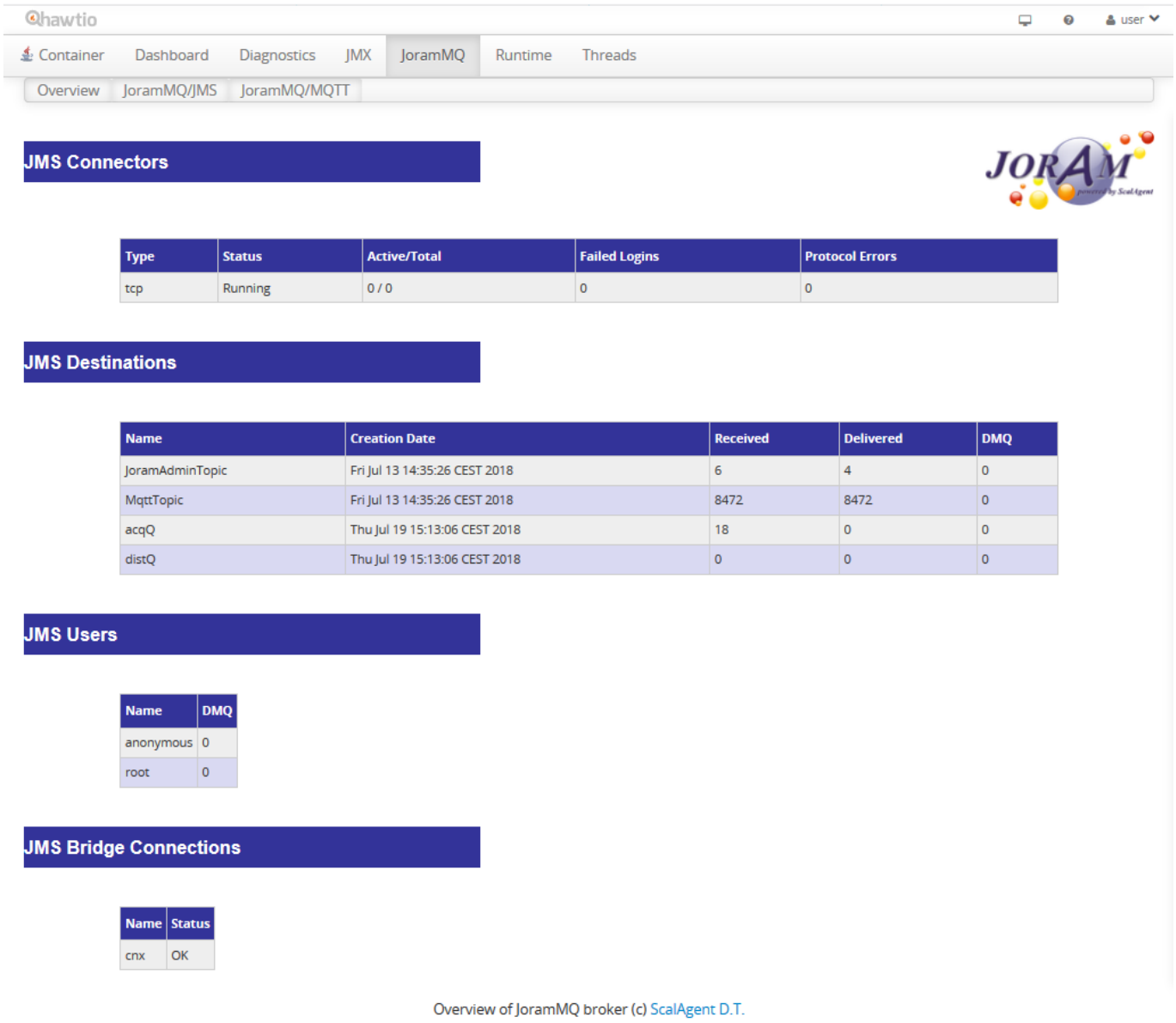

Figure 9 - JoramMQ HawtIO JMS tab

The “JoramMQ/JMS” tab (Figure 9) provides informations about the JMS part of JoramMQ:

- List and status of each running JMS connector. For each connector:

- The protocol used (TCP for example).
- The state of the connector: Running / Stopped.
- The number of active connections / The number of connections initiated since the last boot.
- The number of connections rejected following an authentication error, or a protocol error.

- List and status of all existing destinations (queue and topic, including bridge destinations). For each destination:
  - Its name, and the date of creation.
  - The number of messages received, and the number of messages delivered.
  - The number of error messages sent to the DMQ.
- List and status of created users. For each user:
  - Its name, and the date of creation.
  - The number of error messages sent to the DMQ.
- Name and status of each JMS bridge connections.

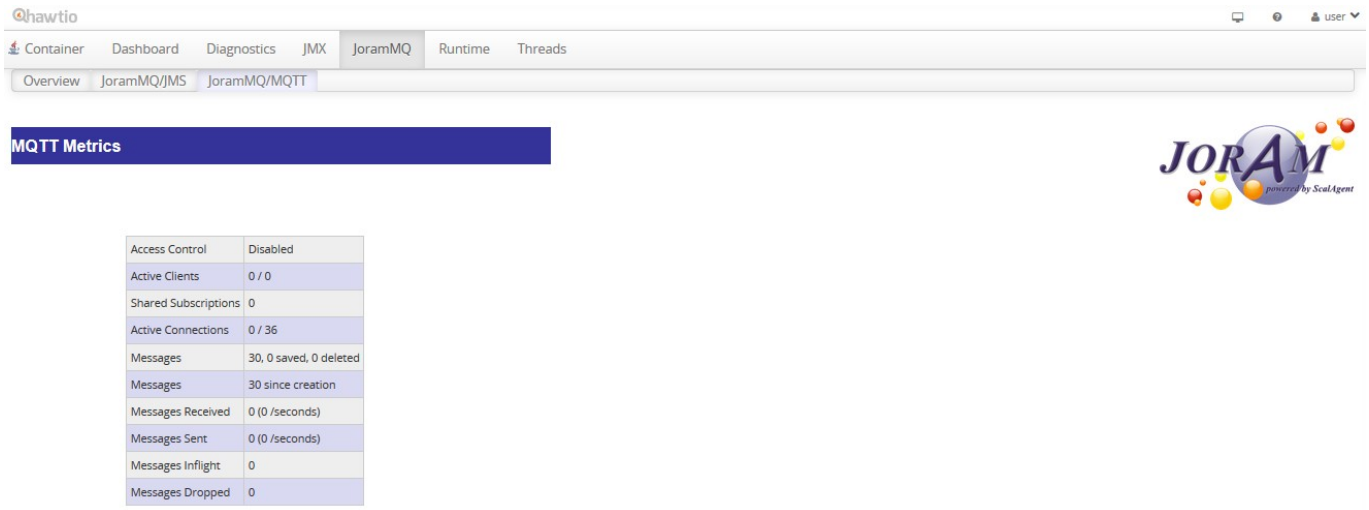

Figure 10 - JoramMQ HawtIO MQTT tab (1)

The “JoramMQ/MQTT” tab provides informations about the MQTT broker:

- Global metrics (Figure 10) from the ClientManager (section 8.3) and ConnectorManager (section 8.6) MBeans:
  - Access control status.
  - The number of connected clients / the number of clients.
  - The number of active shared subscriptions.
  - The number of actives connections / the total number of connections since the last boot.

- The number of messages, the number of messages saved / destroyed since the last boot.
- The number of messages since the first boot.
- The number of messages sent and received, and the corresponding rate per second.
- The number of messages waiting for acknowledgments (qos> 0).
- The number of messages dropped.

- List and status of each running MQTT connectors (Figure 11).For each connector:
  - From the “MQTT Connectors” table: its name, the number of active connections, and its configuration.
  - From the “MQTT Connectors – Health Check” table: its status, the date of the last successful connection test, and the latency of connection and transfer of a message during this last test.

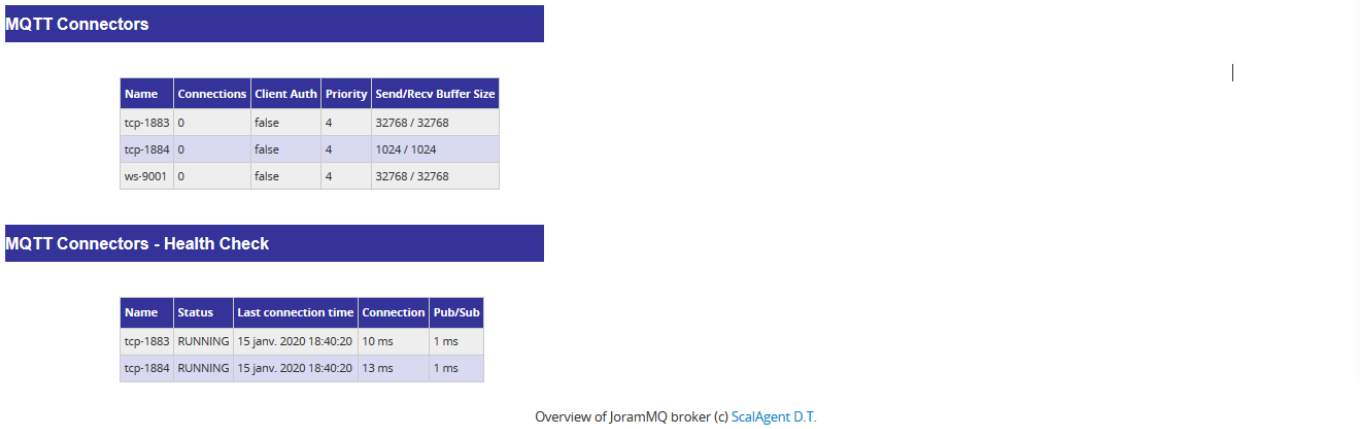

Figure 11 - JoramMQ HawtIO MQTT tab (2)

There may be additional tables depending on the optional components deployed in the JoramMQ server:

- Bridge MQTT (see MBean description section 14.5)
  - For each stream:
    - Name and current status.
    - The date of the last connection / disconnection.
    - The number of connections since the server start.
    - The number of attempts to connect if disconnected.
    - The number of messages transmitted.
    - If the “details” option is selected additional informations are showed.
- Bridge MQTT/JMS bridge (see MBean description section 15.3):
  - The component status.

- If the “details” option is selected, a table shows configuration informations of the component.
- For each stream:
  - Name and current status.
  - The date of the last connection / disconnection.
  - The number of connections since the server start.
  - The number of attempts to connect if disconnected.
  - The number of forwarded messages.
  - If the “details” option is selected additional informations about stream configuration are showed.

- Bridge JMS to File (see MBean description section 16.3):
  - The component status.
  - If the “details” option is selected, a table shows configuration informations of the component.
  - For each stream:
    - Name and current status.
    - The date of the last connection / disconnection.
    - The number of connections since the server start.
    - The number of attempts to connect if disconnected.
    - The number of downloaded files.
- Bridge File to JMS (see MBean description section 17.3):
  - The component status.
  - If the “details” option is selected, a table shows configuration informations of the component.
  - For each stream:
    - Name and current status.
    - The date of the last connection / disconnection.
    - The number of connections since the server start.
    - The number of attempts to connect if disconnected.
    - The number of uploaded files.

23.2.4 JMX Tab

From this tab, you can monitor attributes and run operations provided by MBeans, i.e. the JMX monitoring beans (see chapter 8).

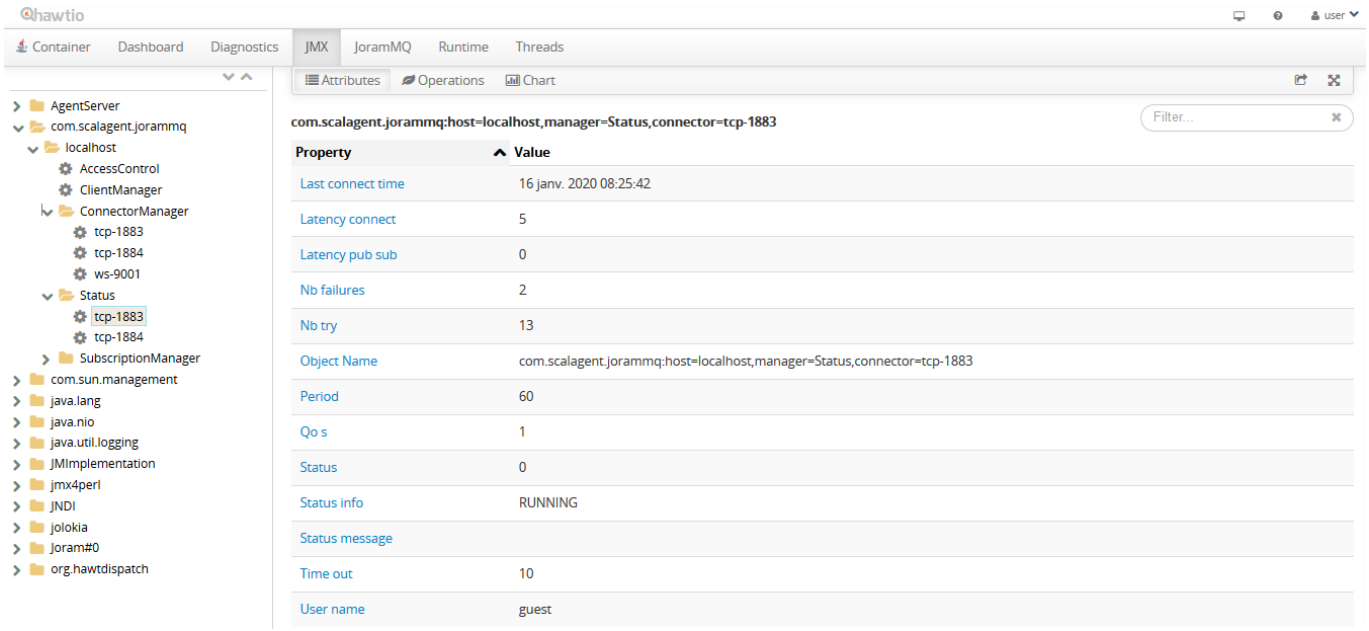

Figure 12 - Mbeans view in JMX tab

24 IoT MQTT Simulator

The JoramMQ IoT simulator is an environment allowing the creation, deployment and execution of tests around the MQTT protocol. The purpose of the tool is therefore to enable smart load simulation for MQTT environments, as well as observation of the functioning of the system. The system is generic, it can be used with different MQTT brokers, and for different types of applications. The system is easily extensible and can take into account other protocols than MQTT.

The system allows the simulation of an IoT application in order to be able on the one hand to evaluate the performance of the broker(s) of messages used, and on the other hand to be able to size the environment (hardware, operating system, application and server settings). The application consists of a set of connected objects and servers interacting through one or more MQTT brokers. It allows the production of message flows similar to those produced by the simulated application.

The JoramMQ IoT simulator is shipped in the jorammq-mqtt-simulator.zip file, to install it you just need to uncompress this file.

The JoramMQ IoT simulator is documented in the jorammq-mqtt-simulator.pdf file:

- Section 2 describes the concepts of the simulator.
- Section 3 describes the rules for extending the simulator.
- Section 4 contains the user guide for the simulator. It describes the syntax of the XML language used, and presents the different subsystems (actions, events) delivered.
- Section 5 includes 2 commented examples.
- Section 6 describes an experiment to offer a graphical view of scenarios with Sirius.
- Section 7 contains the XML schema for the scenario description language.

25 Tools

This chapter is dedicated to the various utilities delivered with JoramMQ.

25.1 Client JMS

This utility is used to verify the correct functioning of the JMS connector and the JNDI service. The corresponding jar (clientjms.jar) is in the lib directory of the delivery, it is used to send and receive JMS messages.

The help option allows you to print a short manual of the tool with its various parameters:

```
$ java -jar lib/clientjms.jar help
usage: java -DJNDI_FILE=./jndi.properties -DADMIN=joramAdmin.xml -DSILENT=true
         -DCF=cf -DDESTINATION=queue -DSEND=1 -DRECEIVE=1 -DTIMEOUT=5000
        -jar clientjms.jar [help]

Options, set by Java environment variable ("-Dproperty=value" in command line)
  - JNDI_FILE: Path of JNDI properties file. If not defined, Joram's default are
    used. "fr.dyade.aaa.jndi2.client.NamingContextFactory" for JNDI Factory,
    "localhost", and 16400 for host and port.
    These values can be overloaded by specific properties below.
  - JNDI_FACTORY: Classname of the JNDI factory (cf java.naming.factory.initial
    property). By default "fr.dyade.aaa.jndi2.client.NamingContextFactory".
  - JNDI_URL: URL of JNDI server (cf java.naming.provider.url). By default
    "scn://localhost:16400".
  - ADMIN: Path of administration file to execute before the test.
  - CF: JNDI name of the ConnectionFactory to use, by default "cf".
  - DESTINATION: JNDI name of JMS destination, by default "queue".
  - USER: User name for authentication, by default "anonymous".
  - PASS: Password for authentication, by default "anonymous".
  - SILENT: If defined true, the output is minimum.
  - SEND: number of messages to send by the test, by default 10. If less
    than 0, no messages are sent.
  - RECEIVE: number of messages to receive by the test, by default 10. If less
    than 0, no messages are received.
  - TIMEOUT: Maximum amount of time to wait the messages, by default 10000ms.
```

25.2 Client MQTT

This utility is used to check the correct operation of the MQTT connector. The corresponding jar (clientmqtt.jar) is located in the lib directory of the distribution, it is used to send and receive MQTT messages.

The help option allows you to print a short manual of the tool with its various parameters:

```
$ java -jar lib/clientmqtt.jar help
usage: java -DURI=tcp://localhost:1883 -DTOPIC=topic -DCLIENTID=client_0 -DQOS=1
        -DUSER=user -DPASS=pass -DNOCLEAN=true -DSILENT=true -DSEND=10
        -DRECEIVE=10 -DTIMEOUT=10000
        -jar clientmqtt.jar [help]

Options, set by Java environment variable ("-Dproperty=value" in command line)
  - URI: URI to access the broker, by default "tcp://localhost:1883".
  - USER: User name for authentication (optional).
  - PASS: Password for authentication (optional).
  - CLIENTID: Identifier of the session, by default "client_" + random.
  - TOPIC: Topic path, by default "topic".
  - NOCLEAN: if defined true, cleanSession flag is set to false.
  - SILENT: If defined true, the output is minimum.
  - SEND: number of messages to send by the test, by default 10. If less
    than 0, no messages are sent.
  - RECEIVE: number of messages to receive by the test, by default 10. If less
    than 0, no messages are received.
  - TIMEOUT: Maximum amount of time to wait the messages, by default 10000ms.

Options for SSL/TLS connections:
  -Dprotocol=protocol | the standard name of the requested protocol, by
                      | default "TLSv1.2".
  -Dalgo=algo         | the standard name of the requested algorithm, by
                      | default "SunX509".
  -Dcatype=catype     | the type of keystore for client certificate, by
                      | default "PKCS12".
  -Dcafile=cafile     | the sources of authentication keys.
  -Dkstype=kstype     | the type of keystore for server certificates, by
                      | default "JKS".
  -Dksfile=ksfile     | the sources of peer authentication trust decisions.
  -Dpassphrase=xxxs   | the password used to check the integrity of
                      | the keystore
```

25.3 JMS Tool

This tool allows you to perform various JMS administration operations. The corresponding jar (joram-tools-jmstool .jar) is located in the lib directory of the distribution.

The help option allows you to print a short manual of the tool with its various parameters:

```
$ java -jar lib/joram-tools-jmstool.jar help
usage: java -DJNDI_FILE=./jndi.properties -DSILENT=true -DALL=true
    -DCF=cf -DQUEUE=queue -DFORWARD=forward -jar jmstool.jar
    <command> [parameters]

Options and parameters, set by Java environment variable ("-Dproperty=value" in
  command line):
  - JNDI_FILE: Path of JNDI properties file. If not defined, Joram's default
    are used. "fr.dyade.aaa.jndi2.client.NamingContextFactory" for JNDI
    Factory, "localhost", and 16400 for host and port.
    These values can be overloaded by specific properties below.
  - JNDI_FACTORY: Classname of the JNDI factory (cf java.naming.factory.initial
    property). By default "fr.dyade.aaa.jndi2.client.NamingContextFactory".
  - JNDI_URL: URL of JNDI server (cf java.naming.provider.url). By default
    "scn://localhost:16400".
  - CF: JNDI name of the ConnectionFactory to use, by default "cf".
  - QUEUE: JNDI name of JMS queue to handle, by default "queue".
  - ADMIN: User name for authentication, by default "root.
  - PASS: Password for authentication, by default "root".
  - FORWARD: JNDI Name of queue to forward messages, no default.
  - MSGID: Unique identifier of message to handle, no default.
  - INPUT: Pathname of file containing additionnal parameters (see
     Parameters section below). Each line specfies a parameter.
     The additional command line parameters, and "GUI" option
     will then be ignored.
  - GUI: If set to true use a GUI dialog to get password, by default false.
  - ALL: If defined true, all messages are automatically handled.
  - DUMP: If defined true, try to dump (hex+ascii) the content of Stream and
    Object messages.
  - SILENT: If defined true, the output is minimum.

Commands:
```

```
  - help: prints the usage message.
  - browse:prints successively each message of the queue, allowing to delete
    or forward it. If ALL option is defined, all messages are automatically
    handled.
  - purge: removes all messages in the queue.
  - forward: forwards the message identified by MSGID option.
  - remove: removes the message identified by MSGID option.
  - pause_on/pause_off: pause (resp. restart) the queue.
  - create: creates the queue and registers it in JNDI. The rights set allow
    any user to send or receive messages on this queue.
  - delete: deletes the queue in the JMS server.
  - statistics statistics: prints all statistics about the queue.
  - lookup: gets the registered object in JNDI (name asked on terminal).
  - unbind: removes the binding in JNDI (name asked on terminal).
  - password: updates a user password (parameters asked on terminal).

Parameters: Some commands require additional parameters normally entered by
  the user at the terminal. These parameters can be given in the command line
  in the order requested by the command:
  - lookup/unbind: <name>
    These commands can take multiple parameters, to stipulate the last
    parameter and avoid a request on the terminal you can use the ""
    empty parameter.
  - password: <user> <password>
    If a parameter contains white space characters it must be surrounded by ".
```

25.4 JMS connector health-check

This component periodically monitors the health of the JMS TCP connectors and publish indicators related to the status of these conenctors. It can be used in 2 ways:

- First, integrated into the OSGi configuration (jar bundle/joram-tools-jmscheck.jar). A global JMX MBean is then published in the Joram JVM (Joram#0:type=healthcheck) as well as an MBean for each monitored connector. In this case all the necessary properties must be present in the OSGi configuration file: conf / felix.properties.
- Second by explicitly using the tool (jar lib/joram-tools-jmscheck.jar) as in the command below. The Mbeans ar then published on the tool's JVM (Joram:type=healthcheck).

The root MBean publishes the following attributes:

- Period: the test period in seconds.

- TimeOut: the timeout in seconds.
- Theshold: the minimum number of failures before generating a dump.
- Delay: the minimal delay between 2 dumps.
- Status: the global healthcheck indication, 0 if all connectors are running, 1 otherwise. StatusInfo gives a "user friendly" information for this attribute: RUNNING or UNREACHABLE.

Each connector's MBean publishes the following attributes:

- Name: the name of the ConnectionFactory used.
- Period: the test period in seconds.
- TimeOut: the timeout in seconds.
- Status: the number of failed attempts in progress, 0 if the last attempts were successful (the component is OK), StatusInfo gives a "user friendly" information for this attribute.
- LastConnectTime: The date of the last successful connection.
- LatencyConnect and LatencyPubSub: the latency times (in ms) of the last connection and the last message exchange.
- ErrorMsg: the error message corresponding to the last attempt (empty if it was successful).
- NbFailures: the total number of failures since launch.
- NbTry: the total number of tests since launch.

The help option allows you to print a short manual of the tool with its various parameters:

```
$ java -jar target/joram-tools-jmscheck.jar help
usage: java -DJNDI_FILE=./jndi.properties -DPERIOD=1 -DTIMEOUT=5000
        -DCF=cf -DQUEUE=queue
        -jar joram-tools-jmscheck.jar [help]

Options, set by Java environment variable ("-Dproperty=value" in command line)
  - JNDI_FILE: Path of JNDI properties file. If not defined, Joram's default are
    used. "fr.dyade.aaa.jndi2.client.NamingContextFactory" for JNDI Factory,
    "localhost", and 16400 for host and port.
    These values can be overloaded by specific properties below.
  - JNDI_FACTORY: Classname of the JNDI factory (cf java.naming.factory.initial
    property).
  - JNDI_HOST: Hostname ou IP address of JNDI server.
  - JNDI_PORT: Listening port of JNDI server.

  - PERIOD: Period between 2 checks, by default 60s.
```

```
  - TIMEOUT: Maximum amount of time to wait connecting and receiving messages,
    by default 10s.

For each JMS connector to monitor there is 4 properties to define:
  - CF: JNDI name of the ConnectionFactory to use.
  - QUEUE: Internal name of JMS destination.
  - USER: User name for authentication, if no defined uses the ConnectionFactory
    default.
  - PASS: Password for authentication, if no defined uses the ConnectionFactory
    default.
If there are multiple connectors to monitor, suffix each property with 1, 2,
    3, etc.

All these properties can be defined in a file whose name is given by the
CONF_FILE property. In this case the other properties defined in the command
line are ignored.
```

25.5 BackupTool

This tool is used to handle JoramMQ backup files.

The help option allows you to print a short manual of the tool with its various parameters:

```
$ java -jar lib/backuptool.jar
usage:
java -DEXTRACT_DIRECTORY=<path> -DSILENT=true -jar backuptool.jar
  [help|list|extract|dump] <path to backup file>

Options and parameters, set by Java environment variable ("-Dproperty=value" in
command line):
  - EXTRACT_DIRECTORY: Path to directory where to restore the data. If it is not
    defined uses the current directory.
  - FILTER: Regular expression allowing to filter objets. If it is not
    defined all objects are processed by the command.

Commands:
  - help: prints the usage message.
  - list: lists the objects contained in the backup file.
  - extract: extracts the objects contained in the backup file.
  - dump: dumps the objects contained in the backup file.
```

25.6 LevelDBTool

This tool allows you to handle a LevelDB repository. It should be used with care.

The help option allows you to print a short manual of the tool with its various parameters:

```
$ java -jar lib/leveldbtool.jar
usage:
java -DEXTRACT_DIRECTORY=<path> -DSILENT=true -jar leveldbtool.jar
  [help|list|extract|dump|delete|compact|repair] <path to leveldb directory>

Options and parameters, set by Java environment variable ("-Dproperty=value" in
command line):
  - EXTRACT_DIRECTORY: Path to directory where to restore the data. If it is not
    defined uses the current directory.
  - PREFIX: Prefix of objects to process. If it is not defined all objects
    are processed by the command.
  - KEY: key of object to delete.

Commands:
  - help: prints the usage message.
  - list: lists the objects contained in the repository.
  - dump: dumps the objects contained in the repository.
  - extract: extracts the objects contained in the repository.
  - delete: deletes the object associated to the key defined by KEY property.
  - compact: compacts the repository.
  - repair: repairs the repository.

  - Use -DLevelDB.useJavaImpl=true to force the use of Java version of LevelDB
    to replace JNI version.
```

The repair command is only accessible if the native version of LevelDB is working (see section 2.11).

🛠 **Warning :** Be careful, leveldbtool handles the database, you need to run it with the appropriate user. This user must be the same as the user who starts the JoramMQ server. This is in order to have access rights to the database files and not to change them so that they can again be handled by the JoramMQ server. It is also recommended to use the same version of the LevelDB library as that used by the broker (LevelDB.useJavaImpl property).

26 Logging configuration

JoramMQ logging is based on Java Logging APIs, contained in the package java.util.logging. Basically, Java Logging includes support for delivering plain text or XML-formatted log records to memory, output streams, consoles, files, and sockets. In addition, the logging APIs are capable of interacting with logging services that already exist on the host operating system.

This document describes the essential configuration functions when using JoramMQ. For advanced usage you are advised to refer to the Java documentation.

26.1 Introduction

By default, configuration of JoramMQ logging is done using the "conf/log.properties" properties file.
A standard version of this file is provided with the JoramMQ shipping.

The configuration is described by several properties. The main concepts defined are:

- Levels: Each message has a severity level. The level gives an indication of the importance and urgency of the message.
- Loggers: The loggers represent the points of interest of the application, they are organized hierarchically. Each application message is sent to a particular logger.
- Handlers: An handler represents an output, ConsoleHandler or FileHandler for example.
- Formatters: The formatters are used to format the content of the messages before they are sent to the handlers.

26.2 Configuration

26.2.1 Levels

Each log message has an associated log Level object. The Level gives a rough guide to the importance and urgency of a log message.

Traditionally Java Logging defines seven standard log levels, ranging from FINEST (the lowest priority, with the lowest value) to SEVERE (the highest priority, with the highest value). In order of priority: SEVERE, WARNING, INFO, CONFIG, FINE, FINER and FINEST. Additionally, level OFF Turns off logging, and level ALL enables logging of all messages.

JoramMQ defines and uses 5 different levels:

- FATAL: It characterizes a very high error message. These messages are rare and reflect a serious malfunction of the application, the outcome of which is normally a shutdown.

- ERROR: It characterizes an error message. These messages are rare and indicate an important error leading to the malfunction of one or more components.
- WARN: It characterizes a warning message. These messages indicate an abnormal situation requiring the user's attention even if the integrity of the application is not compromised.
- INFO: It characterizes a information message. These messages provide information about the operation of the application. It may be important to filter subjects correctly to limit the size of logs.
- DEBUG: It characterizes a debugging message. These messages give detailed information about the operation of the application, they are normally reserved for developers.

26.2.2 Loggers

Logger objects are organized in a hierarchical namespace and child Logger objects inherits some logging properties from their parent. The root logger (named "") has no parent. In particular, a logger may inherit:

- Logging level: If a logger's level is not defined, then the logger will use the level recursively inherited from its parent.
- Handlers: By default, a Logger will log any output messages to its parent's handler, and so on recursively up the tree.

26.2.3 Handlers

Java provides the following Handler classes:

- StreamHandler: A simple handler for writing formatted records to an OutputStream.
- ConsoleHandler: A simple handler for writing formatted records to System.err
- FileHandler: A handler that writes formatted log records either to a single file, or to a set of rotating log files.
- SocketHandler: A handler that writes formatted log records to remote TCP ports.
- MemoryHandler: A handler that buffers log records in memory.

ConsoleHandler

This Handler publishes log records to stderr. By default the SimpleFormatter is used to generate brief summaries.

The ConsoleHandler is initialized using the following properties. If properties are not defined (or have invalid values) then the specified default values are used.

- java.util.logging.ConsoleHandler.level specifies the default level for the Handler (defaults to Level.INFO).
- java.util.logging.ConsoleHandler.filter specifies the name of a Filter class to use (defaults to no Filter).

- java.util.logging.ConsoleHandler.formatter specifies the name of a Formatter class to use (defaults to java.util.logging.SimpleFormatter).
- java.util.logging.ConsoleHandler.encoding the name of the character set encoding to use (defaults to the default platform encoding).

FileHandler

The FileHandler can either write to a specified file, or it can write to a rotating set of files.

For a rotating set of files, as each file reaches a given size limit, it is closed, rotated out, and a new file opened.

Successively older files are named by adding "0", "1", "2", etc. into the base filename. So the most recent file is the one with index "0".

The FileHandler is initialized using the following properties. If properties are not defined (or have invalid values) then the specified default values are used.

- java.util.logging.FileHandler.level specifies the default level for the Handler (defaults to Level.ALL).
- java.util.logging.FileHandler.filter specifies the name of a Filter class to use (defaults to no Filter).
- java.util.logging.FileHandler.formatter specifies the name of a Formatter class to use (defaults to java.util.logging.XMLFormatter)
- java.util.logging.FileHandler.encoding the name of the character set encoding to use (defaults to the default platform encoding).
- java.util.logging.FileHandler.limit specifies an approximate maximum amount to write (in bytes) to each file. If this is zero, then there is no limit (default).
- java.util.logging.FileHandler.count specifies how many output files to cycle through (defaults to 1).
- java.util.logging.FileHandler.pattern specifies a pattern for generating the output file name. See below for details. (Defaults to "%h/java%u.log").
- java.util.logging.FileHandler.append specifies whether the FileHandler should append onto any existing files (defaults to false).

A pattern consists of a string that includes the following special components that will be replaced at runtime:

- "/" the local pathname separator
- "%t" the system temporary directory
- "%h" the value of the "user.home" system property
- "%g" the generation number to distinguish rotated logs
- "%u" a unique number to resolve conflicts
- "%%" translates to a single percent sign "%"

If no "%g" field has been specified and the file count is greater than one, then the generation number will be added to the end of the generated filename, after a dot.

Normally the "%u" unique field is set to 0. However, if the FileHandler tries to open the filename and finds the file is currently in use by another process it will increment the unique number field and try again. This will be repeated until FileHandler finds a file name that is not currently in use. If there is a conflict and no "%u" field has been specified, it will be added at the end of the filename after a dot. (This will be after any automatically added generation number.)

26.2.4 Formatters

Java also includes two standard Formatter classes:

- SimpleFormatter: Writes brief "human-readable" summaries of log records.
- XMLFormatter: Writes detailed XML-structured information.

JoramMQ offers an extended formatter with the class:

- "org.objectweb.util.monolog.api.LogFormatter".

SimpleFormatter

Print a configurable summary of the LogRecord in a human readable format. The summary will typically be 1 or 2 lines.

The log message can be customized by a format string specified in the "java.util.logging.SimpleFormatter.format" property. This format is defined by the java.util.Formatter class in a C language printf-style. The parameters are:

1. a Date object representing event time of the log record.
2. a string representing the caller, if available; otherwise, the logger's name.
3. the logger's name.
4. the log level.
5. the log message.
6. a string representing the throwable associated with the log record and its stacktrace beginning with a newline character, if any; otherwise, an empty string.

For example, the format string "%1$tc %2$s %4$s: %5$s%6$s%n" prints a message including the timestamp (1$), the source (2$), the log level (4$) and the log message (5$) followed with the throwable and its backtrace (6$), if any:

```
Tue Mar 22 13:11:31 PDT 2011 MyClass SEVERE: message without an exception
```

XMLFormatter

Format a LogRecord into a standard XML format.

LogFormatter

As the SimpleFormatter, it prints a configurable summary of the LogRecord.

This formatter is specific to JoramMQ, it offers an extended format which makes it possible to obtain more complete and precise messages.

The log message can be customized by a format string specified in the "java.util.logging.SimpleFormatter.format" property, or by the format parameter of the handler. This format is defined by the java.util.Formatter class in a C language printf-style.

The parameters are:

1. Date/Time in "dd/MM/yyyy hh:mm:ss.SSS" format representing event time of the log record.
2. a Date object representing event time of the log record.
3. Full class name of the caller, if available; otherwise, an empty string.
4. Class name of the caller, if available; otherwise, an empty string.
5. Method name of the caller, if available; otherwise, an empty string.
6. Line number of the caller, if available; otherwise, an empty string.
7. the logger's name.
8. the name of Level.
9. the localized name of Level.
10. the log message.
11. a unique Thread identifier.
12. the thread name.
13. a string representing the throwable associated with the log record and its stacktrace beginning with a newline character, if any; otherwise, an empty string.

For example, we detail below the 'short' and 'long' formats used in the default JoramMQ configuration.

- The short format (`%1$s %7$s %8$s: %10$s%n`) writes the time of event, the source logger, the level and the message, for example:

```
10/02/2022 07:30:54.460 com.scalagent.jorammq.mqtt.adapter.KeyStoreStatus WARN:
Checking MQTT adapter key store .\conf\keystore: found no valid X509 trusted
certificate
```

- The long format (`%1$s %7$s %8$s [%12$s %4$s.%5$s(%6$s)]: %10$s%13$s%n`) adds information about the event source (the thread name, the class, method and line number of the caller) and prints an eventual throwable associated with the event. For example:

```
10/02/2022 07:30:54.460 com.scalagent.jorammq.mqtt.adapter.KeyStoreStatus WARN
[FelixStartLevel KeyStoreStatus.doUpdate(444)]: Checking MQTT adapter key store .\
conf\keystore: found no valid X509 trusted certificate
```

26.3 Default configuration

In this section we explained the content of the JoramMQ default configuration. There are 2 parts in this configuration, the first is dedicated to handlers, the second to loggers.

26.3.1 Handlers configuration

In the first part below, we declare two handlers, a ConsoleHandler and a FileHandler,and we configure them

- ConsoleHandler:
  - It uses the specific JoramMQ formatter classes with a short message format.
  - Only messages with a priority greater than or equal to WARN are printed.
- FileHandler:
  - It uses the specific JoramMQ formatter classes with a long message format.
  - All messages are printed.
  - Messages are written to a rotating set of 10 files of 10Mb.

26.3.2 Loggers configuration

This part is divided in 6 sections:

- First the definition of the level of the root logger (WARN). This level will be inherited by all its descendants if they do not define a specific level. If there is no other logger definition, only messages with a priority higher to WARN would be written.
- The four next sections define loggers of different components:
  - 'fr.dyade.aaa' subtree concerns the base runtime of JoramMQ.
  - 'fr.dyade.aaa.jndi2' subtree concerns the JNDI implementation.
  - 'org.objectweb.joram' subtree concerns the JMS implementation.
  - 'com.scalagent' subtree concerns the JoramMQ MQTT implementation.
- The last section defines loggers of third party software embedded with JoramMQ.

By default, all loggers are configured with the WARN level. Many other loggers are commented out to allow detailed logging of some JoramMQ components.

26.3.3 "conf/logging.properties" file

```
# (C) 2021 - 2022 ScalAgent Distributed Technologies
# All rights reserved

handlers = java.util.logging.FileHandler, java.util.logging.ConsoleHandler

# ------------------------------------------------------------------------
# Console logging
# ------------------------------------------------------------------------
java.util.logging.ConsoleHandler.formatter =
org.objectweb.util.monolog.api.LogFormatter
java.util.logging.ConsoleHandler.format = %1$s %7$s %8$s: %10$s%n
java.util.logging.ConsoleHandler.level = WARN

# ------------------------------------------------------------------------
# File logging
# ------------------------------------------------------------------------
java.util.logging.FileHandler.pattern = ./log/server-%u.%g.log
java.util.logging.FileHandler.limit = 10000000
java.util.logging.FileHandler.count = 10
java.util.logging.FileHandler.formatter =
org.objectweb.util.monolog.api.LogFormatter
java.util.logging.FileHandler.format = %1$s %7$s %8$s [%12$s %4$s.%5$s(%6$s)]:
%10$s%13$s%n
java.util.logging.FileHandler.level = ALL
java.util.logging.FileHandler.append = true

java.util.logging.SimpleFormatter.format = %1$tY-%1$tm-%1$td %1$tH:%1$tM:%1$tS
%4$s %2$s %5$s%6$s%n
org.objectweb.util.monolog.api.LogFormatter.format = %1$s %7$s %8$s [%12$s %4$s.
%5$s(%6$s)]: %10$s%13$s%n

.level = WARN

#=================================================
# A3 Runtime
fr.dyade.aaa.level = WARN

#fr.dyade.aaa.agent.Agent.level = DEBUG
#fr.dyade.aaa.agent.Engine.level = DEBUG
```

```
#fr.dyade.aaa.util.Transaction.level = DEBUG
#fr.dyade.aaa.agent.Network.level = DEBUG
#fr.dyade.aaa.agent.Service.level = DEBUG

#================================================
# Joram/JNDI component
#fr.dyade.aaa.jndi2.client.level = DEBUG
#fr.dyade.aaa.jndi2.server.level = DEBUG

#================================================
# Joram/JMS component
org.objectweb.joram.level = WARN

#org.objectweb.joram.mom.level = DEBUG
#org.objectweb.joram.client.jms.level = DEBUG
#org.objectweb.joram.client.connector.level = DEBUG
#org.objectweb.joram.shared.level = DEBUG

#org.objectweb.joram.client.jms.Session.Message.level = INFO
#org.objectweb.joram.client.jms.Connection.tracker.level = DEBUG
#org.objectweb.joram.client.jms.Session.tracker.level = DEBUG

org.ow2.joram.level = WARN

#org.ow2.joram.jmxconnector.level = DEBUG

#org.objectweb.joram.tools.rest.level = DEBUG
#com.scalagent.joram.mom.dest.rest.level = DEBUG
#org.objectweb.joram.tools.rest.jms.level = DEBUG

# Needed to avoid useless warning message about clock synchronization
# 'TcpConnectionListener.acceptConnection :  -> bad clock synchronization between
client and server'
org.objectweb.joram.mom.proxies.tcp.TcpConnectionListener.level = ERROR

#================================================
# JORAMMQ
com.scalagent.level = WARN

#com.scalagent.txlog.level = DEBUG
```

```
#com.scalagent.batchengine.level = DEBUG
#com.scalagent.batchnetwork.level = DEBUG
#com.scalagent.jorammq.mqtt.adapter.level = DEBUG
# Allows to trace in and out MQTT frames.
#com.scalagent.jorammq.mqtt.adapter.MqttConnection.dump.level = INFO
#com.scalagent.jorammq.mqtt.accesscontrol.level = DEBUG
#com.scalagent.jorammq.mom.tcp.level = DEBUG

# Loggers allowing to control Jetty logging (WARN by default, or FINEST)
#org.eclipse.jetty.level = DEBUG
#org.eclipse.jetty.websocket.level = DEBUG

com.scalagent.tracker.level = ERROR
#com.scalagent.tracker.MqttAgent.level = WARN
#com.scalagent.tracker.MqttBridge.level = WARN
#com.scalagent.tracker.MqttTopic.level = WARN
#com.scalagent.tracker.JoramMqttSession.level = WARN
#com.scalagent.tracker.MqttConnection.level = WARN

#com.scalagent.tracker.exitOnBadOrder.level = DEBUG

#================================================
# Third-party software
org.apache.sshd.level = WARN
org.eclipse.jetty.level = WARN
org.glassfish.jersey.level = WARN
```

26.4 Migration

Until version 1.15 JoramMQ's logging was based on the OW2/Monolog wrapper. This wrapper offered a common interface to the different logging APIs: Log4J, JUL (Java Util Logging), etc. It also allowed a universal configuration regardless of the layout used.

The OW2/Monolog project being now poorly supported, it was decided to replace it in version 1.16 of JoramMQ. The choice fell on the logging component integrated into Java: JUL (Java Util Logging).

The choice of logging component normally has little impact on the Joram user, the main concern is its configuration. In this section we will detail these changes. A logging configuration file can be divided in 2 parts:

- The first part dedicated to handlers is strongly impacted by the implementation change. We recommend that you rebuild it following the instructions above (or go back to the default configuration).
- The second part dedicated to loggers has very few changes, there are essentially two:
  - The "logger" prefix must be removed from logger names.
  - You must add the symbol '=' between the name of the logger and the desired logging level.

for example, for the loggers dedicated to the A3 runtime:

JoramMQ 1.15	**JoramMQ 1.16**
logger.fr.dyade.aaa.level WARN	fr.dyade.aaa.level = WARN
#logger.fr.dyade.aaa.agent.Agent.level DEBUG	#fr.dyade.aaa.agent.Agent.level = DEBUG
#logger.fr.dyade.aaa.agent.Engine.level DEBUG	#fr.dyade.aaa.agent.Engine.level = DEBUG
#logger.fr.dyade.aaa.util.Transaction.level DEBUG	#fr.dyade.aaa.util.Transaction.level = DEBUG
#logger.fr.dyade.aaa.util.Network.level DEBUG	#fr.dyade.aaa.agent.Network.level = DEBUG
#logger.fr.dyade.aaa.agent.Service.level DEBUG	#fr.dyade.aaa.agent.Service.level = DEBUG

26.5 Use of other logging backend

As mentioned above, JoramMQ's logging now relies on JUL. However, JoramMQ exposes an API (package org.objectweb.util.monolog.api) and allows implementations based on other logging backend. The choice of the implementation used is made through the environment variable "fr.dyade.aaa.DEBUG_LOGGING_FACTORY", the default value is:

- "org.objectweb.util.monolog.jul.SimpleLoggingFactory".

For example, a contribution allows to use slf4j with JoramMQ. In this case the configuration must be specific and must be carried out directly on the backend used (slf4j is itself a wrapper). To use this implementation, you must set the configuration variable of the factory ("fr.dyade.aaa.DEBUG_LOGGING_FACTORY") to the value:

- "org.objectweb.util.monolog.slf4j.Slf4jMonologFactory".

Other implementations could be provided.

27 Compatibility between versions

Apart from the libraries (jar, bundle, etc.), the configuration files and the scripts are likely to evolve from one version to another. It is therefore important to check the modifications of the new versions of these files. In particular, it is common for external libraries to evolve for security reasons, among other things. It is therefore imperative to use the new version of the “conf/felix.properties” configuration file, or to integrate its modifications to your possibly customized file.

🛠 **Warning :** Despite all the precautions taken to ensure backward compatibility, when changing the version, it is advisable to first back up the JoramMQ persistence database and to realize a test.

27.1 JoramMQ 1.13 (04/2020)

There is no known incompatibility between JoramMQ versions 1.12 and 1.13.

27.2 JoramMQ 1.14 (02/2021)

There is no known incompatibility between JoramMQ versions 1.13 and 1.14.

27.2.1 Fichier “conf/felix.properties”

The only notable change is the replacement of the “mina-core” and “sshd-core” libraries by the “sshd-osgi” library

JoramMQ 1.13	JoramMQ 1.14
file:./bundle/mina-core.jar \ file:./bundle/sshd-core.jar \	file:./bundle/sshd-osgi.jar \

27.3 JoramMQ 1.15 (12/2021)

There is no known incompatibility between JoramMQ versions 1.14 and 1.15.

The “conf.jorammq.xml” has also evolved to optimize and simplify the default configuration, and to introduce new functions.

Be careful, there are several developments to pay attention to:

- The launch scripts have been modified to take into account any updated configuration before each start[37]. If you want to keep the old behavior you can set the UPDATE_CONF_AUTO variable in the jorammq-server script to any value other than OK.
- There are 2 new variables in the setenv script:

[37] Previously this operation had to be explicitly done via the “jorammq-admin -update” command.

- JORAMMQ_TMP_DIR allows setting the directory to be used for temporary files (by default the OS temporary directory).
- JORAMMQ_JAVA_OPTS allows to specify the Java options to add when launching JoramMQ. For example it can be used to configure detailed output of SSL handshaking.

- In order to avoid potential problems, now the Java version of LevelDB will never be used if it has not explicitly been allowed.
  - The JNI version is used by default. If you want to use the Java version it is necessary to set the property Transaction.LevelDBRepository.useJavaImpl to true in the conf/jorammq.xml file configuration file.
  - Successive use of these 2 versions is not recommended: if you previously used the Java version you must now force the use of this one via the property defined above. To determine the version used you can examine the server launch traces. If the Java version is used there is the message “Cannot use LevelDB with JNI. Using LevelDB Java version instead”.

27.3.1 Fichier “conf/felix.properties”

This version has undergone significant changes with on the one hand the removal of the jleveldb library, but above all the modification of the Web-Services and Web-Socket libraries.

JoramMQ 1.14	JoramMQ 1.15
file:./bundle/jleveldb.jar \	
file:./bundle/geronimo-servlet_3.0_spec.jar \ file:./bundle/jetty-http.jar \ file:./bundle/jetty-continuation.jar \ file:./bundle/jetty-io.jar \ file:./bundle/jetty-util.jar \ file:./bundle/jetty-security.jar \ file:./bundle/jetty-server.jar \ file:./bundle/jetty-servlet.jar \ file:./bundle/jetty-websocket.jar \	file:./bundle/org.apache.felix.http.servlet-api.jar \ file:./bundle/jetty-http.jar \ file:./bundle/jetty-continuation.jar \ file:./bundle/jetty-io.jar \ file:./bundle/jetty-util.jar \ file:./bundle/jetty-util-ajax.jar \ file:./bundle/jetty-security.jar \ file:./bundle/jetty-server.jar \ file:./bundle/jetty-servlet.jar \ file:./bundle/websocket-api.jar \ file:./bundle/websocket-common.jar \ file:./bundle/websocket-server.jar \ file:./bundle/websocket-client.jar \ file:./bundle/websocket-servlet.jar \

file:./bundle/javax.annotation-api.jar \ file:./bundle/javax.inject.jar \ file:./bundle/activation.jar \ file:./bundle/jaxb-api.jar \ file:./bundle/javax.ws.rs-api.jar \ file:./bundle/validation-api.jar \ file:./bundle/jersey-container-servlet-core.jar \ file:./bundle/jersey-server.jar \ file:./bundle/hk2-api.jar \ file:./bundle/aopalliance-repackaged.jar \ file:./bundle/hk2-utils.jar \ file:./bundle/jersey-common.jar \ file:./bundle/jersey-hk2.jar \ file:./bundle/hk2-locator.jar \ file:./bundle/osgi-resource-locator.jar \ file:./bundle/jersey-client.jar \ file:./bundle/javassist.jar \	file:./bundle/jakarta.annotation-api.jar \ file:./bundle/jakarta.inject.jar \ file:./bundle/activation.jar \ file:./bundle/jaxb-api.jar \ file:./bundle/jakarta.ws.rs-api.jar \ file:./bundle/jakarta.validation-api.jar \ file:./bundle/jersey-container-servlet-core.jar \ file:./bundle/jersey-server.jar \ file:./bundle/hk2-api.jar \ file:./bundle/aopalliance-repackaged.jar \ file:./bundle/hk2-utils.jar \ file:./bundle/jersey-common.jar \ file:./bundle/jersey-hk2.jar \ file:./bundle/hk2-locator.jar \ file:./bundle/osgi-resource-locator.jar \ file:./bundle/jersey-client.jar \ file:./bundle/javassist.jar \
file:./bundle/servlet.jar \ file:./bundle/jetty.jar \	file:./bundle/org.apache.felix.http.jetty.jar \

27.4 JoramMQ 1.16 (06/2022)

There is no known incompatibility between JoramMQ versions 1.15 and 1.16. However, several external libraries have evolved among other things for security reasons. It is therefore imperative to use the new version of the “conf/felix.properties” configuration file, or to integrate its modifications to your possibly customized file.

The logging component has evolved, JoramMQ now uses the Java logging library (JUL) by default. The configuration file had to be modified. It is therefore imperative to use the new version of the “conf/log.properties” configuration file, or to integrate its modifications to your possibly customized file.

27.4.1 Fichier “conf/felix.properties”

The only notable change is the suppression of “monolog” library, and the insertion of properties related to OSGi HTTP service.

JoramMQ 1.15	JoramMQ 1.16
file:./bundle/monolog.jar \	

	##
	# Felix HTTP Service
	##
	## component need this bundles
	# file:./bundle/org.apache.felix.http.jetty.jar
	org.apache.felix.http.enable=true
	org.osgi.service.http.port=8989
	#org.apache.felix.https.enable=true
	#org.osgi.service.http.port.secure=8443
	#org.apache.felix.https.keystore=./conf/keystore
	#org.apache.felix.https.keystore.password=joramp
	ass

27.5 JoramMQ 1.17 (03/2023)

There is no known incompatibility between JoramMQ versions 1.16 and 1.17. However, several external libraries have evolved among other things for security reasons.

27.6 JoramMQ 1.18 (09/2023)

There is no known incompatibility between JoramMQ versions 1.17 and 1.18. However, several external libraries have evolved among other things for security reasons.

27.6.1 Fichier “conf/jorammq.xml”

Adds properties dedicated to regulating client MBean registration.

Adds properties dedicated to controlling MQTT codecs.

27.6.2 Fichier “conf/felix.properties”

Adds Jakarta/JMS API bundle: needed by JMS bridge and REST/JMS connector.

There are new properties linked to the transacted mode of the bridge MQTT/JMS (see chapter 15).