Red Hat AMQ 7.2

Managing AMQ Broker

For Use with AMQ Broker 7.2

Last Updated: 2018-11-27

Red Hat AMQ 7.2 Managing AMQ Broker

For Use with AMQ Broker 7.2

Legal Notice

Abstract

This guide describes how to monitor, manage, and upgrade AMQ Broker.

Table of Contents

CHAPTER 1. UPGRADING YOUR BROKER

1.1. ABOUT UPGRADES

Red Hat releases new versions of AMQ Broker to the Customer Portal. Update your brokers to the newest version to ensure that you have the latest enhancements and fixes. In general, Red Hat releases a new version of AMQ Broker in one of three ways:

Major Release

A major upgrade or migration is required when an application is transitioned from one major release to the next, for example, from AMQ Broker 6 to AMQ Broker 7. This type of upgrade is not addressed in this guide. For instructions on how to upgrade from previous releases of AMQ Broker, see Migrating to Red Hat AMQ 7.

Minor Release

AMQ Broker periodically provides minor releases, which are updates that include new features, as well as bug and security fixes. If you plan to upgrade from one AMQ Broker minor release to another, for example, from AMQ Broker 7.0 to AMQ Broker 7.1, code changes should not be required for applications that do not use private, unsupported, or tech preview components.

Micro Release

AMQ Broker also periodically provides micro releases that contain minor enhancements and fixes. Micro releases increment the minor release version by the last digit, for example from 7.0.1 to 7.0.2. A micro release should not require code changes, however, some releases may require configuration changes.

1.2. UPGRADING A BROKER INSTANCE FROM 7.0.X TO 7.0.Y

The procedure for upgrading AMQ Broker from one version of 7.0 to another is similar to the one for installation: you download an archive from the Customer Portal and then extract it. The following subsections describe how to upgrade a 7.0.x broker for different operating systems.

- Upgrading from 7.0.x to 7.0.y on Linux
- Upgrading from 7.0.x to 7.0.y on Windows

1.2.1. Upgrading from 7.0.x to 7.0.y on Linux

The name of the archive that you download could differ from what is used in the following examples.

Prerequisites

- Before upgrading AMQ Broker, review the release notes for the target release.
  The release notes describe important enhancements, known issues, and changes to behavior in the target release.

  For more information, see the AMQ Broker 7.0 Release Notes.

Procedure

1. Download the desired archive from the Red Hat Customer Portal by following the instructions provided in Downloading an AMQ Broker Archive.

2. Change the owner of the archive to the same user that owns the AMQ Broker installation to be upgraded.

   ```
   sudo chown amq-broker:amq-broker jboss-amq-7.x.x.redhat-1.zip
   ```

3. Move the archive to the directory created during the original installation of AMQ Broker. In the following example, the directory **`/opt/redhat`** is used.

   ```
   sudo mv jboss-amq-7.x.x.redhat-1.zip /opt/redhat
   ```

4. As the directory owner, extract the contents of the compressed archive. The archive is kept in a compressed format. In the following example, the user **`amq-broker`** extracts the archive by using the unzip command.

   ```
   su - amq-broker
   cd /opt/redhat
   unzip jboss-amq-7.x.x.redhat-1.zip
   ```

5. Stop the broker if it is running.

   ```
   BROKER_INSTANCE_DIR/bin/artemis stop
   ```

6. Back up the instance directory of the broker by copying it to the home directory of the current user.

   ```
   cp -r BROKER_INSTANCE_DIR ~/
   ```

7. (Optional) Note the current version of the broker. After the broker stops, a line similar to the one below is displayed at the end of its log file, which can be found at ***`BROKER_INSTANCE_DIR`*`/log/artemis.log`**.

   ```
   INFO  [org.apache.activemq.artemis.core.server] AMQ221002: Apache
   ActiveMQ Artemis Message Broker version 2.0.0.amq-700005-redhat-1
   [4782d50d-47a2-11e7-a160-9801a793ea45] stopped, uptime 28 minutes
   ```

8. Edit the ***`BROKER_INSTANCE_DIR`*`/etc/artemis.profile`** configuration file to set the **`ARTEMIS_HOME`** property to the new directory created when the archive was extracted.

   ```
   ARTEMIS_HOME='/opt/redhat/jboss-amq-7.x.x-redhat-1'
   ```

9. Restart the broker by entering the following command:

   ```
   BROKER_INSTANCE_DIR/bin/artemis run
   ```

10. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the log file ***`BROKER_INSTANCE_DIR`*`/log/artemis.log`** and find two lines similar to the ones below. Note the new version number that appears in the log after the broker is live.

    ```
    INFO  [org.apache.activemq.artemis.core.server] AMQ221007: Server is
    now live
    ...
    ```

```
INFO  [org.apache.activemq.artemis.core.server] AMQ221001: Apache
ActiveMQ Artemis Message Broker version 2.1.0.amq-700005-redhat-1
[0.0.0.0, nodeID=4782d50d-47a2-11e7-a160-9801a793ea45]
```

1.2.2. Upgrading from 7.0.x to 7.0.y on Windows

Prerequisites

- Before upgrading AMQ Broker, review the release notes for the target release.
  The release notes describe important enhancements, known issues, and changes to behavior in the target release.

  For more information, see the AMQ Broker 7.0 Release Notes.

Procedure

1. Download the desired archive from the Red Hat Customer Portal by following the instructions provided in Downloading an AMQ Broker Archive.
2. Use a file manager to move the archive to the folder you created during the last installation of AMQ Broker.
3. Extract the file contents into the directory by right-clicking on the zip file and choosing **Extract All**.
4. Stop the broker if it is running by entering the following command.

   ```
   BROKER_INSTANCE_DIR\bin\artemis-service.exe stop
   ```

5. Back up the broker by using a file manager.
   a. Right click on the *BROKER_INSTANCE_DIR* folder and select **Copy**.
   b. Right click in the same window and select **Paste**.
6. (Optional) Note the current version of the broker. After the broker stops, a line similar to the one below is displayed at the end of its log file, which can be found at ***BROKER_INSTANCE_DIR*\log\artemis.log**.

   ```
   INFO  [org.apache.activemq.artemis.core.server] AMQ221002: Apache
   ActiveMQ Artemis Message Broker version 2.0.0.amq-700005-redhat-1
   [4782d50d-47a2-11e7-a160-9801a793ea45] stopped, uptime 28 minutes
   ```

7. Edit the ***BROKER_INSTANCE_DIR*\etc\artemis.profile** configuration file to set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted.

   ```
   ARTEMIS_HOME=NEW_INSTALL_DIR
   ```

8. Restart the broker entering the following command:

   ```
   BROKER_INSTANCE_DIR\bin\artemis-service.exe start
   ```

9. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the log file ***BROKER_INSTANCE_DIR*\log\artemis.log** and find two lines

similar to the ones below. Note the new version number that appears in the log after the broker is live.

```
INFO  [org.apache.activemq.artemis.core.server] AMQ221007: Server is
now live
...
INFO  [org.apache.activemq.artemis.core.server] AMQ221001: Apache
ActiveMQ Artemis Message Broker version 2.1.0.amq-700005-redhat-1
[0.0.0.0, nodeID=4782d50d-47a2-11e7-a160-9801a793ea45]
```

1.3. UPGRADING A BROKER INSTANCE FROM 7.0.X TO 7.1.0

AMQ Broker 7.1.0 includes configuration files and settings that were not included with previous versions. Upgrading a broker instance from 7.0.x to 7.1.0 requires adding these new files and settings to your existing 7.0.x broker instances. The following subsections describe how to upgrade a 7.0.x broker instance to 7.1.0 for different operating systems.

IMPORTANT

Starting with AMQ Broker 7.1.0, you can access the AMQ Console only from the local host by default. You must modify the configuration in ***BROKER_INSTANCE_DIR*/etc/jolokia-access.xml** to enable remote access. For more information, see Securing AMQ Console and AMQ Broker Connections.

- Upgrading from 7.0.x to 7.1.0 on Linux
- Upgrading from 7.0.x to 7.1.0 on Windows

1.3.1. Upgrading from 7.0.x to 7.1.0 on Linux

Before you can upgrade a 7.0.x broker, you need to install Red Hat AMQ Broker 7.1.0 and create a temporary broker instance. This will generate the 7.1.0 configuration files required to upgrade a 7.0.x broker.

Prerequisites

- Before upgrading AMQ Broker, review the release notes for the target release.
  The release notes describe important enhancements, known issues, and changes to behavior in the target release.

  For more information, see the AMQ Broker 7.1 Release Notes.

- Before upgrading your 7.0.x brokers, you must first install version 7.1.
  For steps on installing 7.1 on Linux, see Installing on Linux.

Procedure

1. If it is running, stop the 7.0.x broker you want to upgrade:

   ```
   $ BROKER_INSTANCE_DIR/bin/artemis stop
   ```

2. Back up the instance directory of the broker by copying it to the home directory of the current user.

```
cp -r BROKER_INSTANCE_DIR ~/
```

3. Open the file **`artemis.profile`** in the ***`BROKER_INSTANCE_DIR`*****`/etc/`** directory of the 7.0.x broker.

    a. Update the **`ARTEMIS_HOME`** property so that its value refers to the installation directory for AMQ Broker 7.1.0:

    ```
    ARTEMIS_HOME="7.1.0_INSTALL_DIR"
    ```

    b. On the line below the one you updated, add the property **`ARTEMIS_INSTANCE_URI`** and assign it a value that refers to the 7.0.x broker instance directory:

    ```
    ARTEMIS_INSTANCE_URI="file://7.0.x_BROKER_INSTANCE_DIR"
    ```

    c. Update the **`JAVA_ARGS`** property by adding the **`jolokia.policyLocation`** parameter and assigning it the following value:

    ```
    -Djolokia.policyLocation=${ARTEMIS_INSTANCE_URI}/etc/jolokia-
    access.xml
    ```

4. Create a 7.1.0 broker instance. The creation procedure generates the configuration files required to upgrade from 7.0.x to 7.1.0. In the following example, note that the instance is created in the directory **`upgrade_tmp`**:

```
$ 7.1.0_INSTALL_DIR/bin/artemis create --allow-anonymous --user
admin --password admin upgrade_tmp
```

5. Copy configuration files from the **`etc`** directory of the temporary 7.1.0 instance into the ***`BROKER_INSTANCE_DIR`*****`/etc/`** directory of the 7.0.x broker.

    a. Copy the **`management.xml`** file:

    ```
    $ cp TEMPORARY_7.1.0_BROKER_INSTANCE_DIR/etc/management.xml
    7.0_BROKER_INSTANCE_DIR/etc/
    ```

    b. Copy the **`jolokia-access.xml`** file:

    ```
    $ cp TEMPORARY_7.1.0_BROKER_INSTANCE_DIR/etc/jolokia-access.xml
    7.0_BROKER_INSTANCE_DIR/etc/
    ```

6. Open up the **`bootstrap.xml`** file in the ***`BROKER_INSTANCE_DIR`*****`/etc/`** directory of the 7.0.x broker.

    a. Comment out or delete the following two lines:

    ```
    <app url="jolokia" war="jolokia.war"/>
    <app url="hawtio" war="hawtio-no-slf4j.war"/>
    ```

    b. Add the following to replace the two lines removed in the previous step:

    ```
    <app url="console" war="console.war"/>
    ```

7. Start the broker that you upgraded:

```
$ BROKER_INSTANCE_DIR/bin/artemis run
```

Additional Resources

For more information about creating an instance of the broker, see Creating a Broker Instance.

1.3.2. Upgrading from 7.0.x to 7.1.0 on Windows

Before you can upgrade a 7.0.x broker, you need to install Red Hat AMQ Broker 7.1.0 and create a temporary broker instance. This will generate the 7.1.0 configuration files required to upgrade a 7.0.x broker.

Prerequisites

- Before upgrading AMQ Broker, review the release notes for the target release.
  The release notes describe important enhancements, known issues, and changes to behavior in the target release.

  For more information, see the AMQ Broker 7.1 Release Notes.

- Before upgrading your 7.0.x brokers, you must first install version 7.1.
  For steps on installing 7.1 on Windows, see Installing on Windows.

Procedure

1. If it is running, stop the 7.0.x broker you want to upgrade:

   ```
   > BROKER_INSTANCE_DIR\bin\artemis-service.exe stop
   ```

2. Back up the instance directory of the broker by using a file manager.

   a. Right click on the *BROKER_INSTANCE_DIR* folder and select **Copy**.

   b. Right click in the same window and select **Paste**.

3. Open the file **`artemis.profile`** in the ***`BROKER_INSTANCE_DIR`*`/etc/`** directory of the 7.0.x broker.

   a. Update the **`ARTEMIS_HOME`** property so that its value refers to the installation directory for AMQ Broker 7.1.0:

      ```
      ARTEMIS_HOME="7.1.0_INSTALL_DIR"
      ```

   b. On the line below the one you updated, add the property **`ARTEMIS_INSTANCE_URI`** and assign it a value that refers to the 7.0.x broker instance directory:

      ```
      ARTEMIS_INSTANCE_URI="file://7.0.x_BROKER_INSTANCE_DIR"
      ```

   c. Update the **`JAVA_ARGS`** property by adding the **`jolokia.policyLocation`** parameter and assigning it the following value:

      ```
      -Djolokia.policyLocation=${ARTEMIS_INSTANCE_URI}/etc/jolokia-
      access.xml
      ```

4. Create a 7.1.0 broker instance. The creation procedure generates the configuration files required to upgrade from 7.0.x to 7.1.0. In the following example, note that the instance is created in the directory **`upgrade_tmp`**:

   ```
   > 7.1.0_INSTALL_DIR/bin/artemis create --allow-anonymous --user
   admin --password admin upgrade_tmp
   ```

5. Copy configuration files from the **`etc`** directory of the temporary 7.1.0 instance into the ***`BROKER_INSTANCE_DIR`*****`/etc/`** directory of the 7.0.x broker.

   a. Copy the **`management.xml`** file:

      ```
      > cp TEMPORARY_7.1.0_BROKER_INSTANCE_DIR/etc/management.xml
      7.0_BROKER_INSTANCE_DIR/etc/
      ```

   b. Copy the **`jolokia-access.xml`** file:

      ```
      > cp TEMPORARY_7.1.0_BROKER_INSTANCE_DIR/etc/jolokia-access.xml
      7.0_BROKER_INSTANCE_DIR/etc/
      ```

6. Open up the **`bootstrap.xml`** file in the ***`BROKER_INSTANCE_DIR`*****`/etc/`** directory of the 7.0.x broker.

   a. Comment out or delete the following two lines:

      ```
      <app url="jolokia" war="jolokia.war"/>
      <app url="hawtio" war="hawtio-no-slf4j.war"/>
      ```

   b. Add the following to replace the two lines removed in the previous step:

      ```
      <app url="console" war="console.war"/>
      ```

7. Start the broker that you upgraded:

   ```
   > BROKER_INSTANCE_DIR\bin\artemis-service.exe start
   ```

Additional Resources
For more information about creating an instance of the broker, see Creating a Broker Instance.

1.4. UPGRADING A BROKER INSTANCE FROM 7.1.X TO 7.2.0

AMQ Broker 7.2.0 includes configuration files and settings that were not included with 7.0.x versions. If you are running 7.0.x instances, you must first upgrade those broker instances from 7.0.x to 7.1.0 before upgrading to 7.2.0. The following subsections describe how to upgrade a 7.1.x broker instance to 7.2.0 for different operating systems.

IMPORTANT

Starting with AMQ Broker 7.1.0, you can access the AMQ Console only from the local host by default. You must modify the configuration in ***`BROKER_INSTANCE_DIR`*****`/etc/jolokia-access.xml`** to enable remote access. For more information, see Securing AMQ Console and AMQ Broker Connections.

- Upgrading from 7.1.x to 7.2.0 on Linux
- Upgrading from 7.1.x to 7.2.0 on Windows

1.4.1. Upgrading from 7.1.x to 7.2.0 on Linux

NOTE

The name of the archive that you download could differ from what is used in the following examples.

Procedure

1. Download the desired archive from the Red Hat Customer Portal by following the instructions provided in Downloading an AMQ Broker Archive.

2. Change the owner of the archive to the same user that owns the AMQ Broker installation to be upgraded.

   ```
   sudo chown amq-broker:amq-broker amq-7.x.x.redhat-1.zip
   ```

3. Move the archive to the directory created during the original installation of AMQ Broker. In the following example, the directory **`/opt/redhat`** is used.

   ```
   sudo mv amq-7.x.x.redhat-1.zip /opt/redhat
   ```

4. As the directory owner, extract the contents of the compressed archive. In the following example, the user **`amq-broker`** extracts the archive by using the unzip command.

   ```
   su - amq-broker
   cd /opt/redhat
   unzip jboss-amq-7.x.x.redhat-1.zip
   ```

5. Stop the broker if it is running.

   ```
   BROKER_INSTANCE_DIR/bin/artemis stop
   ```

6. Back up the instance directory of the broker by copying it to the home directory of the current user.

   ```
   cp -r BROKER_INSTANCE_DIR ~/
   ```

7. (Optional) Note the current version of the broker. After the broker stops, a line similar to the one below is displayed at the end of its log file, which can be found at ***`BROKER_INSTANCE_DIR`*****`/log/artemis.log`**.

   ```
   INFO  [org.apache.activemq.artemis.core.server] AMQ221001: Apache
   ActiveMQ Artemis Message Broker version 2.5.0.amq-720001-redhat-1
   [0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]
   ```

8. Edit the ***`BROKER_INSTANCE_DIR`*****`/etc/artemis.profile`** configuration file to set the **`ARTEMIS_HOME`** property to the new directory created when the archive was extracted.

```
ARTEMIS_HOME='/opt/redhat/amq-7.x.x-redhat-1'
```

9. Restart the broker by entering the following command:

```
BROKER_INSTANCE_DIR/bin/artemis run
```

10. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the log file ***BROKER_INSTANCE_DIR*/log/artemis.log** and find two lines similar to the ones below. Note the new version number that appears in the log after the broker is live.

```
INFO  [org.apache.activemq.artemis.core.server] AMQ221007: Server is
now live
...
INFO  [org.apache.activemq.artemis.core.server] AMQ221001: Apache
ActiveMQ Artemis Message Broker version 2.5.0.amq-720001-redhat-1
[0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]
```

Additional Resources

- For more information about creating an instance of the broker, see Creating a Broker Instance.
- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the ***BROKER_INSTANCE_DIR*/etc/artemis.profile** file, update the **ARTEMIS_INSTANCE_ETC_URI** property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **etc/** and **data/** directories within the broker instance's directory.

1.4.2. Upgrading from 7.1.x to 7.2.0 on Windows

Procedure

1. Download the desired archive from the Red Hat Customer Portal by following the instructions provided in Downloading an AMQ Broker Archive.
2. Use a file manager to move the archive to the folder you created during the last installation of AMQ Broker.
3. Extract the file contents into the directory by right-clicking on the zip file and choosing **Extract All**.
4. Stop the broker if it is running by entering the following command.

```
BROKER_INSTANCE_DIR\bin\artemis-service.exe stop
```

5. Back up the broker by using a file manager.
   a. Right click on the *BROKER_INSTANCE_DIR* folder and select **Copy**.
   b. Right click in the same window and select **Paste**.

6. (Optional) Note the current version of the broker. After the broker stops, a line similar to the one below is displayed at the end of its log file, which can be found at ***BROKER_INSTANCE_DIR*\log\artemis.log**.

```
INFO  [org.apache.activemq.artemis.core.server] AMQ221002: Apache
ActiveMQ Artemis Message Broker version 2.0.0.amq-700005-redhat-1
[4782d50d-47a2-11e7-a160-9801a793ea45] stopped, uptime 28 minutes
```

7. Edit the ***BROKER_INSTANCE_DIR*\etc\artemis.profile** configuration file to set the **ARTEMIS_HOME** property to the new directory created when the archive was extracted.

```
ARTEMIS_HOME=NEW_INSTALL_DIR
```

8. Restart the broker entering the following command:

```
BROKER_INSTANCE_DIR\bin\artemis-service.exe start
```

9. (Optional) Confirm that the broker is running and that the version has changed. After starting the broker, open the log file ***BROKER_INSTANCE_DIR*\log\artemis.log** and find two lines similar to the ones below. Note the new version number that appears in the log after the broker is live.

```
INFO  [org.apache.activemq.artemis.core.server] AMQ221007: Server is
now live
...
INFO  [org.apache.activemq.artemis.core.server] AMQ221001: Apache
ActiveMQ Artemis Message Broker version 2.5.0.amq-720001-redhat-1
[0.0.0.0, nodeID=554cce00-63d9-11e8-9808-54ee759954c4]
```

Additional Resources

- For more information about creating an instance of the broker, see Creating a Broker Instance.
- You can now store a broker instance's configuration files and data in any custom directory, including locations outside of the broker instance's directory. In the ***BROKER_INSTANCE_DIR*\etc\artemis.profile** file, update the **ARTEMIS_INSTANCE_ETC_URI** property by specifying the location of the custom directory after creating the broker instance. Previously, these configuration files and data could only be stored in the **\etc** and **\data** directories within the broker instance's directory.

CHAPTER 2. MANAGEMENT

AMQ Broker provides both a graphical as well as a programming interface to help you manage your brokers.

2.1. USING AMQ CONSOLE

If you prefer to use a graphic interface to manage AMQ, you can use AMQ Console. AMQ Console is a web console included in the AMQ Broker installation, and it enables you to use a web browser to manage AMQ Broker and AMQ Interconnect.

For more information, see Using AMQ Console.

2.2. USING THE MANAGEMENT API

AMQ Broker 7.2 has an extensive management API that allows a user to modify a broker's configuration, create new resources (for example, addresses and queues), inspect these resources (for example, how many messages are currently held in a queue), and interact with them (for example, to remove messages from a queue). Using the management API, clients can also manage the broker and subscribe to management notifications.

There are two ways to manage the broker:

1. Using JMX — JMX is the standard way to manage Java applications
2. Using the JMS API — management operations are sent to the broker using JMS messages and the AMQ JMS client

Although there are two different ways to manage the broker, each API supports the same functionality. If it is possible to manage a resource using JMX it is also possible to achieve the same result by using JMS messages and the AMQ JMS client.

This choice depends on your particular requirements, application settings, and environment.

Regardless of the way you invoke management operations, the management API is the same.

For each managed resource, there exists a Java interface describing what can be invoked for this type of resource.

The broker exposes its managed resources in the **`org.apache.activemq.artemis.api.core.management`** package.

The way to invoke management operations depends on whether JMX messages or JMS messages and the AMQ JMS client is used.

NOTE

A few management operations require a **`filter`** parameter to choose which messages are affected by the operation. Passing **`null`** or an empty string means that the management operation will be performed on *all messages*.

2.2.1. Managing the Broker

Listing, creating, deploying, and destroying queues

A list of deployed queues can be retrieved using the **getQueueNames()** method.
Queues can be created or destroyed using the management operations **createQueue()**, **deployQueue()**, or **destroyQueue()** on the **ActiveMQServerControl** (with the **ObjectName org.apache.activemq.artemis:broker="*BROKER_NAME*"** or the resource name **server**).

createQueue will fail if the queue already exists while **deployQueue** will do nothing.

Pausing and resuming queues

The **QueueControl** can pause and resume the underlying queue. When a queue is paused, it will receive messages but will not deliver them. When it is resumed, it will begin delivering the queued messages, if any.

Listing and closing remote connections

- Retrieve a client's remote addresses by using **listRemoteAddresses()**. It is also possible to close the connections associated with a remote address using the **closeConnectionsForAddress()** method.
- Alternatively, list connection IDs using **listConnectionIDs()** and list all the sessions for a given connection ID using **listSessions()**.

Managing transactions

In case of a broker crash, when the broker restarts, some transactions might require manual intervention. Use the the following methods to help resolve issues you encounter.

- List the transactions which are in the prepared states (the transactions are represented as opaque Base64 Strings) using the **listPreparedTransactions()** method lists.
- Commit or rollback a given prepared transaction using **commitPreparedTransaction()** or **rollbackPreparedTransaction()** to resolve heuristic transactions.
- List heuristically completed transactions using the **listHeuristicCommittedTransactions()** and **listHeuristicRolledBackTransactions** methods.

Enabling and resetting message counters

- Enable and disable message counters using the **enableMessageCounters()** or **disableMessageCounters()** method.
- Reset message counters by using the **resetAllMessageCounters()** and **resetAllMessageCounterHistories()** methods.

Retrieving broker configuration and attributes

The **ActiveMQServerControl** exposes the broker's configuration through all its attributes (for example, **getVersion()** method to retrieve the broker's version, and so on).

Listing, creating, and destroying Core Bridge and diverts

- List deployed Core Bridge and diverts using the **getBridgeNames()** and **getDivertNames()** methods respectively.
- Create or destroy using bridges and diverts using **createBridge()** and **destroyBridge()** or **createDivert()** and **destroyDivert()** on the **ActiveMQServerControl** (with the **ObjectName**

org.apache.activemq.artemis:broker="*BROKER_NAME*" or the resource name **server**).

Stopping the broker and forcing failover to occur with any currently attached clients

Use the **forceFailover()** on the **ActiveMQServerControl** (with the **ObjectName** **org.apache.activemq.artemis:broker="*BROKER_NAME*"** or the resource name **server**)

NOTE

Since this method actually stops the broker you will probably receive some sort of error depending on which management service you use to call it.

2.2.2. Managing Addresses

Manage addresses using the **AddressControl** class (with the **ObjectName** **org.apache.activemq.artemis:broker="*BROKER_NAME*",component=addresses,address="*ADDRESS_NAME*"** or the resource name **address.*ADDRESS_NAME***).

- Modify roles and permissions for an address using the **addRole()** or **removeRole()** methods. You can list all the roles associated with the queue with the **getRoles()** method.

2.2.3. Managing Queues

The bulk of the core management API deals with queues. The **QueueControl** class defines the queue management operations (with the **ObjectName** **org.apache.activemq.artemis:broker="*BROKER_NAME*",component=addresses,address="*BOUND_ADDRESS*",subcomponent=queues,routing-type="*ROUTING_TYPE*",queue="*QUEUE_NAME*"** or the resource name **queue.*QUEUE_NAME***).

Most of the management operations on queues take either a single message ID (for example, to remove a single message) or a filter (for example, to expire all messages with a given property.)

Expiring, sending to a dead letter address, and moving messages

- Expire messages from a queue using the **expireMessages()** method. If an expiry address is defined, messages will be sent to it, otherwise they are discarded. The queue's expiry address can be set with the **setExpiryAddress()** method.

- Send messages to a dead letter address with the **sendMessagesToDeadLetterAddress()** method. It returns the number of messages which are sent to the dead letter address. If a dead letter address is not defined, messages are removed from the queue and discarded. The queue's dead letter address can be set with the **setDeadLetterAddress()** method.

- Move messages from one queue to another by using the **moveMessages()** method.

Listing and removing messages

- List messages from a queue using the **listMessages()** method. It will return an array of **Map**, one **Map** for each message.

- Remove messages from a queue using the **removeMessages()** method, which returns a

boolean for the single message ID variant or the number of removed messages for the filter variant. This method takes a **filter** argument to remove only filtered messages. Setting the filter to an empty string will in effect remove all messages.

- Counting messages
  The number of messages in a queue is returned by the **getMessageCount()** method. Alternatively, the **countMessages()** will return the number of messages in the queue which match a given filter.

- Changing message priority
  The message priority can be changed by using the **changeMessagesPriority()** method which returns a **boolean** for the single message ID variant or the number of updated messages for the filter variant.

- Message counters
  Message counters can be listed for a queue with the **listMessageCounter()** and **listMessageCounterHistory()** methods (see the Message Counters section). The message counters can also be reset for a single queue using the **resetMessageCounter()** method.

- Retrieving the queue attributes
  The **QueueControl** exposes queue settings through its attributes (for example, **getFilter()** to retrieve the queue's filter if it was created with one, **isDurable()** to know whether the queue is durable, and so on).

- Pausing and resuming queues
  The **QueueControl** can pause and resume the underlying queue. When a queue is paused, it will receive messages but will not deliver them. When it is resumed, it will begin delivering the queued messages, if any.

2.2.3.1. Managing Other Resources

You can start and stop the broker's remote resources (acceptors, diverts, bridges, and so on) so that a broker can be taken offline for a given period of time without stopping it completely (for example, if other management operations must be performed, such as resolving heuristic transactions). These resources are:

- Acceptors
  Start or stop an acceptor using the **start()** or. **stop()** method on the **AcceptorControl** class (with the **ObjectName** **org.apache.activemq.artemis:broker="*BROKER_NAME*",component=acceptors,name="*ACCEPTOR_NAME*"** or the resource name **acceptor.*ADDRESS_NAME***). Acceptor parameters can be retrieved using the **AcceptorControl** attributes. See Network Connections: Acceptors and Connectors for more information about Acceptors.

- Diverts
  Start or stop a divert using the **start()** or **stop()** method on the **DivertControl** class (with the **ObjectName** **org.apache.activemq.artemis:broker="*BROKER_NAME*",component=diverts,name="*DIVERT_NAME*"** or the resource name **divert.*DIVERT_NAME***). Divert parameters can be retrieved using the **DivertControl** attributes.

- Bridges

Start or stop a bridge using the **`start()`** (resp. **`stop()`**) method on the **`BridgeControl`** class (with the **`ObjectName`** **`org.apache.activemq.artemis:broker="`*`BROKER_NAME`*`",component=bridge,name="`*`BRIDGE_NAME`*`"`** or the resource name **`bridge.`*`BRIDGE_NAME`***). Bridge parameters can be retrieved using the **`BridgeControl`** attributes. See Clustering for more information.

- Broadcast groups
  Start or stop a broadcast group using the **`start()`** or **`stop()`** method on the **`BroadcastGroupControl`** class (with the **`ObjectName`** **`org.apache.activemq.artemis:broker="`*`BROKER_NAME`*`",component=broadcast-group,name="`*`BROADCAST_GROUP_NAME`*`"`** or the resource name **`broadcastgroup.`*`BROADCAST_GROUP_NAME`***). Broadcast group parameters can be retrieved using the **`BroadcastGroupControl`** attributes. See Clustering for more information.

- Discovery groups
  Start or stop a discovery group using the **`start()`** or **`stop()`** method on the **`DiscoveryGroupControl`** class (with the **`ObjectName`** **`org.apache.activemq.artemis:broker="`*`BROKER_NAME`*`",component=discovery-group,name="`*`DISCOVERY_GROUP_NAME`*`"`** or the resource name **`discovery.`*`DISCOVERY_GROUP_NAME`***). Discovery groups parameters can be retrieved using the **`DiscoveryGroupControl`** attributes. See Clustering for more information.

- Cluster connections
  Start or stop a cluster connection using the **`start()`** or **`stop()`** method on the **`ClusterConnectionControl`** class (with the **`ObjectName`** **`org.apache.activemq.artemis:broker="`*`BROKER_NAME`*`",component=cluster-connection,name="`*`CLUSTER_CONNECTION_NAME`*`"`** or the resource name **`clusterconnection.`*`CLUSTER_CONNECTION_NAME`***). Cluster connection parameters can be retrieved using the **`ClusterConnectionControl`** attributes. See Clustering for more information.

2.2.4. Managing the Broker Using JMX

The broker can be managed using JMX. The management API is exposed by the broker using MBeans interfaces. The broker registers its resources with the domain **`org.apache.activemq`**.

For example, the **`ObjectName`** to manage a queue named **`exampleQueue`** is:

```
org.apache.activemq.artemis:broker="__BROKER_NAME__",component=addresses,address="exampleQueue",subcomponent=queues,routingtype="anycast",queue="exampleQueue"
```

and the MBean is:

```
org.apache.activemq.artemis.api.management.QueueControl
```

The MBean's **`ObjectName`** is built using the helper class **`org.apache.activemq.artemis.api.core.management.ObjectNameBuilder`**. You can also use **`jconsole`** to find the **`ObjectName`** of the MBeans you want to manage.

Managing the broker using JMX is identical to management of any Java applications using JMX. It can be done by reflection or by creating proxies of the MBeans.

2.2.4.1. Configuring JMX Management

By default, JMX is enabled to manage the broker. It can be disabled by setting **`jmx-management-enabled`** to **`false`** in **`broker.xml`**:

```
<jmx-management-enabled>false</jmx-management-enabled>
```

If JMX is enabled, the broker can be managed locally using **`jconsole`**.

NOTE

Remote connections to JMX are not enabled by default for security reasons. Refer to Oracle's Java Management Guide to configure the broker for remote management. System properties must be set in the **`artemis`**, or **`artemis.cmd`** for Windows installations, shell script located under ***`INSTALL_DIR`*****`/bin`**.

By default, the broker uses the JMX domain "org.apache.activemq.artemis". To manage several brokers from the *same* MBeanServer, the JMX domain can be configured for each individual broker by setting **`jmx-domain`** in **`broker.xml`**:

```
<jmx-domain>my.org.apache.activemq</jmx-domain>
```

2.2.4.2. MBeanServer Configuration

When the broker is run in standalone mode, it uses the Java Virtual Machine's **`Platform MBeanServer`** to register its MBeans. By default Jolokia is also deployed to allow access to the MBean server using REST.

2.2.4.3. Exposing JMX Using Jolokia

The default Broker configuration ships with the Jolokia http agent deployed as a web application. Jolokia is a remote JMX over HTTP bridge that exposes MBeans. For more information see the Jolokia documentation.

NOTE

To use Jolokia, the user must belong to the role defined by the **`hawtio.role`** system property in the ***`BROKER_INSTANCE_DIR`*****`/etc/artemis.profile`** configuration file. By default, this role is **`amq`**. For more information about assigning a user to a role, seeAdding Users.

Example 2.1. Using Jolokia to Query the Broker's Version

This example uses a Jolokia REST URL to find the version of a broker.

```
$ curl
http://admin:admin@localhost:8161/console/jolokia/read/org.apache.active
mq.artemis:broker=\"0.0.0.0\"/Version
{"request":
{"mbean":"org.apache.activemq.artemis:broker=\"0.0.0.0\"","attribute":"V
ersion","type":"read"},"value":"2.4.0.amq-710002-redhat-
1","timestamp":1527105236,"status":200}
```

2.2.5. Managing the Broker Using JMS Messages and the AMQ JMS Client

The management queue is a special queue and needs to be instantiated directly by the client:

```
Queue managementQueue =
ActiveMQJMSClient.createQueue("activemq.management");
```

To invoke management operations using JMS messages and the AMQ JMS client:

1. Create a **`QueueRequestor`** to send messages to the management address and receive replies.
2. Create a **`Message`**.
3. Use the helper class **`org.apache.activemq.artemis.api.jms.management.JMSManagementHelper`** to fill the message with the management properties.
4. Send the message using the **`QueueRequestor`**.
5. Use the helper class **`org.apache.activemq.artemis.api.jms.management.JMSManagementHelper`** to retrieve the operation result from the management reply.

For example, to view the number of messages in the JMS queue **`exampleQueue`**:

```
Queue managementQueue =
ActiveMQJMSClient.createQueue("activemq.management");

QueueSession session = ...
QueueRequestor requestor = new QueueRequestor(session, managementQueue);
connection.start();
Message message = session.createMessage();
JMSManagementHelper.putAttribute(message, "queue.exampleQueue",
"messageCount");
Message reply = requestor.request(message);
int count = (Integer)JMSManagementHelper.getResult(reply);
System.out.println("There are " + count + " messages in exampleQueue");
```

2.2.5.1. Configuring Broker Management Using JMS Messages and the AMQ JMS Client

The management address to send management messages is configured in the broker.xml file: <management-address>queue.activemq.management</management-address>

By default, the address is **`queue.activemq.management`**. The management address requires a special user permission type, **`manage`**, to be able to receive and handle management messages. This permission type is specified in the broker.xml file:

<security-setting-match="queue.activemq.management"> <permission-type="manage" roles="admin"/> </security-setting>

2.2.6. Management Notifications

The broker sends notifications to inform listeners of events such as the creation of new resources, security violations, and other events.

There are two ways to receive these notifications:

- JMX notifications
- JMS messages

2.2.6.1. JMX Notifications

If JMX is enabled (see Configuring JMX Management), JMX notifications can be received by subscribing to **`ObjectName org.apache.activemq.artemis:broker="`*`BROKER_NAME`*`"`**.

2.2.6.2. Notification Types and Headers

Below is a list of all the different kinds of notifications as well as which headers are on the messages. Every notification has a **`_AMQ_NotifType`** (value noted in parentheses) and **`_AMQ_NotifTimestamp`** header. The timestamp is the unformatted result of a call to **`java.lang.System.currentTimeMillis()`**.

- **`BINDING_ADDED`** (0)

  > ```
  > `_AMQ_Binding_Type`, `_AMQ_Address`, `_AMQ_ClusterName`,
  > `_AMQ_RoutingName`, `_AMQ_Binding_ID`, `_AMQ_Distance`,
  > `_AMQ_FilterString`
  > ```

- **`BINDING_REMOVED`** (1)

  > ```
  > `_AMQ_Address`, `_AMQ_ClusterName`, `_AMQ_RoutingName`,
  > `_AMQ_Binding_ID`, `_AMQ_Distance`, `_AMQ_FilterString`
  > ```

- **`CONSUMER_CREATED`** (2)

  > ```
  > `_AMQ_Address`, `_AMQ_ClusterName`, `_AMQ_RoutingName`,
  > `_AMQ_Distance`, `_AMQ_ConsumerCount`, `_AMQ_User`,
  > `_AMQ_RemoteAddress`, `_AMQ_SessionName`, `_AMQ_FilterString`
  > ```

- **`CONSUMER_CLOSED`** (3)

  > ```
  > `_AMQ_Address`, `_AMQ_ClusterName`, `_AMQ_RoutingName`,
  > `_AMQ_Distance`, `_AMQ_ConsumerCount`, `_AMQ_User`,
  > `_AMQ_RemoteAddress`, `_AMQ_SessionName`, `_AMQ_FilterString`
  > ```

- **`SECURITY_AUTHENTICATION_VIOLATION`** (6)

  > ```
  > `_AMQ_User`
  > ```

- **`SECURITY_PERMISSION_VIOLATION`** (7)

  > ```
  > `_AMQ_Address`, `_AMQ_CheckType`, `_AMQ_User`
  > ```

- **`DISCOVERY_GROUP_STARTED`** (8)

> `` `name` ``

- **`DISCOVERY_GROUP_STOPPED`** (9)

  > `` `name` ``

- **`BROADCAST_GROUP_STARTED`** (10)

  > `` `name` ``

- **`BROADCAST_GROUP_STOPPED`** (11)

  > `` `name` ``

- **`BRIDGE_STARTED`** (12)

  > `` `name` ``

- **`BRIDGE_STOPPED`** (13)

  > `` `name` ``

- **`CLUSTER_CONNECTION_STARTED`** (14)

  > `` `name` ``

- **`CLUSTER_CONNECTION_STOPPED`** (15)

  > `` `name` ``

- **`ACCEPTOR_STARTED`** (16)

  > `` `factory`, `id` ``

- **`ACCEPTOR_STOPPED`** (17)

  > `` `factory`, `id` ``

- **`PROPOSAL`** (18)

  > `` `_JBM_ProposalGroupId`, `_JBM_ProposalValue`, `_AMQ_Binding_Type`, `_AMQ_Address`, `_AMQ_Distance` ``

- **`PROPOSAL_RESPONSE`** (19)

  > `` `_JBM_ProposalGroupId`, `_JBM_ProposalValue`, `_JBM_ProposalAltValue`, `_AMQ_Binding_Type`, `_AMQ_Address`, `_AMQ_Distance` ``

- **`CONSUMER_SLOW`** (21)

```
`_AMQ_Address`, `_AMQ_ConsumerCount`, `_AMQ_RemoteAddress`,
`_AMQ_ConnectionName`, `_AMQ_ConsumerName`, `_AMQ_SessionName`
```

2.2.7. Message Counters

Message counters can be used to obtain information on queues over time as the broker keeps a history on queue metrics.

They can be used to show trends on queues. For example, using the management API, it would be possible to query the number of messages in a queue at regular intervals. However, this would not be enough to know if the queue is used: the number of messages can remain constant because nobody is sending or receiving messages from the queue or because there are as many messages sent to the queue than messages consumed from it. The number of messages in the queue remains the same in both cases but its use is widely different.

Message counters gives additional information about the queues:

- **`count`**
  The *total* number of messages added to the queue since the broker was started
- **`countDelta`**
  The number of messages added to the queue *since the last message counter update*
- **`messageCount`**
  The *current* number of messages in the queue
- **`messageCountDelta`**
  The *overall* number of messages added/removed from the queue *since the last message counter update*. For example, if **`messageCountDelta`** is equal to **`-10`** this means that overall 10 messages have been removed from the queue (for example, 2 messages were added and 12 were removed)
- **`lastAddTimestamp`**
  The timestamp of the last time a message was added to the queue
- **`udpateTimestamp`**
  The timestamp of the last message counter update

  These attributes can be used to determine other meaningful data as well. For example, to know specifically how many messages were consumed from the queue since the last update simply subtract the **`messageCountDelta`** from **`countDelta`**.

2.2.7.1. Configuring Message Counters

By default, message counters are disabled as it might have a small negative effect on memory.

To enable message counters, you can set it to **`true`** in **`broker.xml`**:

```
<message-counter-enabled>true</message-counter-enabled>
```

Message counters keeps a history of the queue metrics (10 days by default) and samples all the queues at regular interval (10 seconds by default). If message counters are enabled, these values should be configured to suit your messaging use case in **`broker.xml`**:

```
<!-- keep history for a week -->
<message-counter-max-day-history>7</message-counter-max-day-history>
<!-- sample the queues every minute (60000ms) -->
<message-counter-sample-period>60000</message-counter-sample-period>
```

Message counters can be retrieved using the Management API. For example, to retrieve message counters on a JMS queue using JMX:

```
// retrieve a connection to the brokers MBeanServer
MBeanServerConnection mbsc = ...
JMSQueueControlMBean queueControl =
(JMSQueueControl)MBeanServerInvocationHandler.newProxyInstance(mbsc,
   on,
   JMSQueueControl.class,
   false);
// message counters are retrieved as a JSON String
String counters = queueControl.listMessageCounter();
// use the MessageCounterInfo helper class to manipulate message counters
more easily
MessageCounterInfo messageCounter = MessageCounterInfo.fromJSON(counters);
System.out.format("%s message(s) in the queue (since last sample: %s)\n",
messageCounter.getMessageCount(),
messageCounter.getMessageCountDelta());
```

APPENDIX A. COMMAND-LINE TOOLS

AMQ Broker includes a set of command-line interface (CLI) tools so you can manage your messaging journal. The table below lists the name for each tool and its description.

Tool	Description
exp	Exports the message data using a special and independent XML format.
imp	Imports the journal to a running broker using the output provided by **`exp`**.
data	Prints reports about journal records and compacts their data.
encode	Shows an internal format of the journal encoded to String.
decode	Imports the internal journal format from encode.

For a full list of commands available for each tool, use the **`help`** parameter followed by the tool's name. In the example below, the CLI output lists all the commands available to the **`data`** tool after the user entered the command **`./artemis help data`**.

```
$ ./artemis help data

NAME
        artemis data - data tools group
        (print|imp|exp|encode|decode|compact) (example ./artemis data
print)

SYNOPSIS
        artemis data
        artemis data compact [--broker <brokerConfig>] [--verbose]
                [--paging <paging>] [--journal <journal>]
                [--large-messages <largeMessges>] [--bindings <binding>]
        artemis data decode [--broker <brokerConfig>] [--suffix <suffix>]
                [--verbose] [--paging <paging>] [--prefix <prefix>] [--
file-size <size>]
                [--directory <directory>] --input <input> [--journal
<journal>]
                [--large-messages <largeMessges>] [--bindings <binding>]
        artemis data encode [--directory <directory>] [--broker
<brokerConfig>]
                [--suffix <suffix>] [--verbose] [--paging <paging>] [--
prefix <prefix>]
                [--file-size <size>] [--journal <journal>]
                [--large-messages <largeMessges>] [--bindings <binding>]
        artemis data exp [--broker <brokerConfig>] [--verbose]
                [--paging <paging>] [--journal <journal>]
                [--large-messages <largeMessges>] [--bindings <binding>]
        artemis data imp [--host <host>] [--verbose] [--port <port>]
                [--password <password>] [--transaction] --input <input>
[--user <user>]
        artemis data print [--broker <brokerConfig>] [--verbose]
```

```
                [--paging <paging>] [--journal <journal>]
                [--large-messages <largeMessges>] [--bindings <binding>]

COMMANDS
        With no arguments, Display help information

        print
            Print data records information (WARNING: don't use while a
            production server is running)

        ...
```

You can use the help at the tool for more information on how to execute each of the tool’s commands. For example, the CLI lists more information about the **`data print`** command after the user enters the **`./artemis help data print`**.

```
$ ./artemis help data print

NAME
        artemis data print - Print data records information (WARNING:
don't use
        while a production server is running)

SYNOPSIS
        artemis data print [--bindings <binding>] [--journal <journal>]
                [--paging <paging>]

OPTIONS
        --bindings <binding>
            The folder used for bindings (default ../data/bindings)

        --journal <journal>
            The folder used for messages journal (default ../data/journal)

        --paging <paging>
            The folder used for paging (default ../data/paging)
```

Revised on 2018-11-27 15:21:21 UTC