Administering a Message Broker

Version 4.4.1
Sept. 2011

Administering a Message Broker

Version 4.4.1

Updated: 06 Jun 2013

Trademark Disclaimer

Third Party Acknowledgements

One or more products in the Red Hat JBoss Fuse release includes third party components covered by licenses that require that the following documentation notices be provided:

- JLine (http://jline.sourceforge.net) jline:jline:jar:1.0

  License: BSD (LICENSE.txt) - Copyright (c) 2002-2006, Marc Prud'hommeaux `<mwp1@cornell.edu>`

  All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  - Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
  - Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
  - Neither the name of JLine nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR

SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- Stax2 API (http://woodstox.codehaus.org/StAX2) org.codehaus.woodstox:stax2-api:jar:3.1.1

  License: The BSD License (http://www.opensource.org/licenses/bsd-license.php)

  Copyright (c) <YEAR>, <OWNER> All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  - Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
  - Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- jibx-run - JiBX runtime (http://www.jibx.org/main-reactor/jibx-run) org.jibx:jibx-run:bundle:1.2.3

  License: BSD (http://jibx.sourceforge.net/jibx-license.html) Copyright (c) 2003-2010, Dennis M. Sosnoski.

  All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  - Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
  - Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
  - Neither the name of JiBX nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON

ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- JavaAssist (http://www.jboss.org/javassist) org.jboss.javassist:com.springsource.javassist:jar:3.9.0.GA:compile

  License: MPL (http://www.mozilla.org/MPL/MPL-1.1.html)

- HAPI-OSGI-Base Module (http://hl7api.sourceforge.net/hapi-osgi-base/) ca.uhn.hapi:hapi-osgi-base:bundle:1.2

  License: Mozilla Public License 1.1 (http://www.mozilla.org/MPL/MPL-1.1.txt)

Table of Contents

List of Tables

List of Examples

Chapter 1. Setting up the Administration Tool

Fuse Message Broker's administrative tool requires the shell's environment be set up properly. This involves setting a number of environment variables and can typically be accomplished by creating a simple shell script.

Setting up the Windows Environment

Overview

The Windows Fuse Message Broker administrative tool uses four environment variables to determine where to locate configuration files, runtime libraries, and data files. In addition, it requires that your Java environment is properly set up.

Optionally, you may also want to add the administrative tool to the shell's path so that you can use the tool from anywhere.

Environment variables

Table 1.1 on page 12 describes the environment variables used by the administration tool on Windows platforms.

Table 1.1. Windows Administration Tool Environment Variables

Variable	Description
`ACTIVEMQ_HOME`	Specifies the directory where Fuse Message Broker is installed.
`ACTIVEMQ_BASE`	Specifies the directory containing files associated with the current broker instance. Default value is the value of `ACTIVEMQ_HOME`.
`ACTIVEMQ_CLASSPATH`	Specifies the classpath used by the current broker instance.
`SSL_OPTS`	Specifies Java properties required to support SSL/TLS protocols. See *ActiveMQ Security Guide* for details.

Java environment

You must set the `JAVA_HOME` environment variable to the location of the Java installation you want to use for running Fuse Message Broker.

Setup script

Example 1.1 on page 12 shows a sample script for setting up the administration tool's environment.

Example 1.1. Sample Windows Environment Script

```
@echo off
set ACTIVEMQ_HOME=ActiveMQInstallDir
```

```
set ACTIVEMQ_BASE=BrokerInstanceDir
set PATH=%PATH%;%ACTIVEMQ_HOME%\bin

REM Configure Java
set JAVA_HOME=JavaInstallDir
set PATH=%JAVA_HOME%\bin;%PATH%

set ACTIVEMQ_CLASSPATH=%ACTIVEMQ_HOME%\lib\optional\activemq-optional-5.5.1-fuse-00-xx.jar

echo Setting Apache ActiveMQ 5.5 environment
```

This sample script sets `JAVA_HOME` as well as the Fuse Message Broker specific environment variables. In addition, it adds the `activemq-optional-5.5.1-fuse-00-xx.jar` JAR file to your Fuse Message Broker classpath so that the optional feature of the broker are available.

Setting up the Unix/Linux/OS X Environment

Overview

The Fuse Message Broker administrative tool on Unix and Unix-like platforms uses a number of environment variables. The environment variables are used to configure the broker's runtime environment and configure the behavior of the administration tool. The tool can also call one or more start up scripts to effect its execution.

In addition, it requires that your Java environment is properly set up.

Optionally, you may also want to add the administrative tool to the shell's path so that you can use the tool from anywhere.

Environment variables

Table 1.2 on page 14 describes the environment variables used by the administration tool on Unix and Unix-like platforms.

Table 1.2. Unix Administration Tool Environment Variables

Variable	Description
`ACTIVEMQ_HOME`	Specifies the directory where Fuse Message Broker is installed.
`ACTIVEMQ_CLASSPATH`	Specifies the classpath used by the current broker instance.
`ACTIVEMQ_SSL_OPTS`	Specifies Java properties required to support SSL/TLS protocols. See *ActiveMQ Security Guide* for details.
`ACTIVEMQ_CONFIG_DIR`	Specifies the location of the broker's configuration directory. Default is `$ACTIVEMQ_HOME/conf`.
`ACTIVEMQ_DATA_DIR`	Specifies the location of the broker's data directory. Default is `$ACTIVEMQ_HOME/data`.
`ACTIVEMQ_PIDFILE`	Specifies the location of the file that holds the broker's process ID(PID).
`ACTIVEMQ_USER`	Specifies the user as which the broker daemon runs. The user should be a user with non-root privileges. Default value is a blank string, which implies that the broker does *not* change user.
`ACTIVEMQ_OPTS_MEMORY`	Specifies the JVM memory configuration. Default value is `-Xms256M -Xmx256M`.

Variable	Description
`ACTIVEMQ_QUEUEMANAGERURL`	Specifies the default connection URL that is used with the **browse** task. Default value is `--amqurl tcp://localhost:61616`.
`ACTIVEMQ_KILL_MAXSECONDS`	Specifies how many seconds the **stop** task will wait for an orderly shutdown to complete before killing the broker using `SIGKILL`.

Startup scripts

On Unix, the administration tool can find and source a *startup script* immediately before it executes the specified task. The startup script is a shell script, which is normally used to set some environment variables.

The administration tool looks for startup scripts in the following locations:

1. `/etc/default/activemq`
2. `/home/User/.activemqrc`

If both of these scripts exist, they will *both* be sourced by the administration tool, in the order shown.

Java environment

You must set the `JAVA_HOME` environment variable to the location of the Java installation you want to use for running Fuse Message Broker.

Chapter 2. Installing Fuse Message Broker as a Windows Service

Fuse Message Broker ships with wrapper that simplifies the process of installing it as a service on a machine running Windows.

Fuse Message Broker uses a wrapper, based on the Java service wrapper[1] from Tanuki Software Ltd., that provides a solution to the problem of launching a Java application as a Windows service. The wrapper provides a mechanism for specifying all of the required JVM and application specific parameters to the Windows' service interface.

The wrapper is located in the `bin/win32` folder of a Windows Fuse Message Broker installation. This folder contains the following:

- `activemq.bat`—script that starts the broker in the foreground
- `InstallService.bat`—script that installs the broker as a Windows service
- `UninstallService.bat`—script that uninstalls the broker from the service registry
- `wrapper.conf`—file that configures the wrapper service
- `wrapper.dll`
- `wrapper.exe`—wrapper service executable

Before using the wrapper to install Fuse Message Broker as a Windows service, you will want to check the wrappers configuration to ensure it is set up properly.

[1] http://www.tanukisoftware.com/en/wrapper.php

Configuring the Wrapper

Overview

The wrapper is configured by the `wrapper.conf` file, which is located under the *`InstallDir`*`/bin/win32/` directory.

Specifying the Fuse Message Broker's environment

A broker's environment is controlled by two environment variables:

- `ACTIVEMQ_HOME`—the location of the Fuse Message Broker install directory.
- `ACTIVEMQ_BASE`—the root directory containing the configuration, logging, and persistence data specific to the broker instance.

  The configuration for the broker instance is stored in the *`ACTIVEMQ_BASE`*`/conf` directory. The logging and persistence data is stored in the *`ACTIVEMQ_BASE`*`/data` directory.

It is likely that you will need to customize the values of the `ACTIVEMQ_HOME` and `ACTIVEMQ_BASE` properties in the `wrapper.conf` file. Example 2.1 on page 18 shows the default values.

Example 2.1. Default Environment Settings

```
set.default.ACTIVEMQ_HOME=../..
set.default.ACTIVEMQ_BASE=../..
```

Important

When specifying file and directory paths in `wrapper.conf`, the forward slash, `/`, is used.

Passing parameters to the JVM

If you want to pass parameters to the JVM, you do so by setting wrapper properties using the form `wrapper.java.additional.`*`<n>`*. *`<n>`* is a sequence number that must be distinct for each parameter.

Setting Java system properties

One of the most useful things you can do by passing additional parameters to the JVM is to set Java system properties. The syntax for setting a Java system property is `wrapper.java.additional.`*`<n>`*`=-D`*`PropName=PropValue`*.

Example 2.2 on page 19 shows the default Java properties.

Example 2.2. Default Java System Properties

```
# Java Additional Parameters
# note that n is the parameter number starting from 1.
wrapper.java.additional.1=-Dactivemq.home="%ACTIVEMQ_HOME%"
wrapper.java.additional.2=-Dactivemq.base="%ACTIVEMQ_BASE%"
wrapper.java.additional.3=-Djavax.net.ssl.keyStorePassword=password
wrapper.java.additional.4=-Djavax.net.ssl.trustStorePassword=password
wrapper.java.additional.5=-Djavax.net.ssl.keyStore="%ACTIVEMQ_BASE%/conf/broker.ks"
wrapper.java.additional.6=-Djavax.net.ssl.trustStore="%ACTIVEMQ_BASE%/conf/broker.ts"
wrapper.java.additional.7=-Dcom.sun.management.jmxremote
wrapper.java.additional.8=-Dorg.apache.activemq.UseDedicatedTaskRunner=true
wrapper.java.additional.9=-Djava.util.logging.config.file=logging.properties
```

Setting application parameters

You specify application parameters using the syntax `wrapper.app.parameter.<n>`. <*n*> is a sequence number that must follow the pattern:

- `1`—specifies the name of the Java class executed by the wrapper
- `>=2`—specify properties passed to the application

Example 2.3 on page 19 shows the default settings.

Example 2.3. Default Wrapper Application Settings

```
# Application parameters.  Add parameters as needed starting from 1
wrapper.app.parameter.1=org.apache.activemq.console.Main
wrapper.app.parameter.2=start
```

Adding classpath entries

You add classpath entries using the syntax `wrapper.java.classpath.<n>`. <*n*> is a sequence number that must be distinct for each classpath entry.

Example 2.4 on page 19 shows the default classpath entries.

Example 2.4. Default Wrapper Classpath

```
# Java Classpath (include wrapper.jar)  Add class path elements as
#  needed starting from 1
wrapper.java.classpath.1=%ACTIVEMQ_HOME%/bin/wrapper.jar
wrapper.java.classpath.2=%ACTIVEMQ_HOME%/bin/run.jar
```

Reference

For a complete description of all the properties you can set in the `wrapper.conf` configuration file, see Configuration Property Overview[2] at the Tanuki Web site.

[2] http://wrapper.tanukisoftware.com/doc/english/properties.html

Installing the Windows Service

Overview

Fuse Message Broker provides the `InstallService.bat` and `UninstallService.bat` scripts to install and uninstall the broker as a Windows service.

Installing the broker as a service

After customizing the `wrapper.conf` configuration file as described in ????, change directory to *`InstallDir`*`\bin\win32`, and install the broker as a Windows service, using `InstallService.bat`, as follows:

```
C:\apache-activemq5.5.1-fuse-00-xx\bin\win32> InstallService
wrapper  | ActiveMQ installed.
```

Once installed the Fuse Message Broker service will automatically start up when Windows is launched.

Starting the broker manually

You can manually start the installed Fuse Message Broker service using the Windows **net start** tool. The service is registered under the name `ActiveMQ`.

You start the service as follows:

```
c:\> net start ActiveMQ
The ActiveMQ service is starting.........
The ActiveMQ service was started successfully.
```

Stopping the broker manually

You can manually stop the installed Fuse Message Broker service using the Windows **net stop** tool. The service is registered under the name `ActiveMQ`.

You stop the service as follows:

```
c:\> net stop ActiveMQ
The ActiveMQ service is stopping...
The ActiveMQ service was stopped successfully.
```

Uninstalling the broker

To remove the broker from the Windows service registry, run **UninstallService.bat** as follows:

```
C:\apache-activemq5.5.1-fuse-00-xx\bin\win32> UninstallService
wrapper  | ActiveMQ removed.
```

Chapter 3. Starting a Broker

You can start a broker from the command line using either the administration tool or Maven. You specify the configuration for the started broker using a configuration URI.

Specifying the Broker's Configuration

Overview

There are three ways to pass configuration to a broker instance at start-up:

- XBean URI—specifies the location a Spring XML configuration file
- Broker URI—specifies the broker configuration as part of the URI
- Properties URI—specifies the location of properties file containing Fuse Message Broker configuration

XBean configuration

XBean configuration uses a Fuse Message Broker XML configuration file to configure the broker. You pass the location of the configuration file to the administrative tool using a URI prefixed with `xbean:`.

There are three ways to specify the location of the XML configuration file using an XBean URI:

- on the Java classpath using the syntax `xbean:`*`fileName`*

  For example the XBean URI `xbean:activemq.xml` specifies the file `activemq.xml` that is located somewhere on the Java classpath.

- on the file system using the syntax `xbean:file:`*`filePath`*

  For example the XBean URI `xbean:file:activemq.xml` specifies the file `activemq.xml` that is located in the current directory.

Tip

You can specify both relative and absolute paths to the file.

- on a resource path using the syntax `xbean:`*`resourcePath`*

For more information on the Fuse Message Broker XML configuration see the XML Configuration Reference[1].

[1] http://fusesource.com/docs/broker/5.5/xmlref/index.html

Broker URI

You can specify a broker's configuration entirely on the command line using a broker URI of the form shown in Example 3.1 on page 25.

Example 3.1. Broker URI Syntax

```
broker:(transportURI,[network:networkURI])/[brokerName]?brokerOptions
```

- *`transportURI`*—a comma separated list of URIs at which the broker listens for client connections

  See *Broker Client Connectivity Guide* for more information.

- *`networkURI`*—a comma separated list of URIs at which the broker listens for connections from other brokers

  See "Broker Networks" in *Clustering Guide* for more information.

- *`brokerName`*—the broker's name

- *`brokerOptions`*—an ampersand(*&*) separated list configuration properties

  See ???? for a description of the properties.

For example, the URI shown in Example 3.2 on page 25 starts up a broker that accepts connections on port 61616, establishes a network connection to `remotehost:61616`, and disables persistence.

Example 3.2. Broker URI

```
broker:(tcp://localhost:61616,network:static:tcp://remotehost:61616)?persistent=false&use
Jmx=true
```

Property configuration

Property configuration uses a Java properties file to configure the broker. You pass the location of the configuration file to the administrative tool using a URI prefixed with `properties:`. As with the XBean URI, you can specify a location on the classpath, on the local file system, or using a resource path. For example the URI `properties:file:activemq.properties` specifies the file `activemq.properties` that is located in the current directory.

The properties file shown in Example 3.3 on page 26 starts up a broker that accepts connections on port 61616, is named Cheese, and disables persistence and JMX.

Example 3.3. Broker Properties File

```
useJmx = false
persistent = false
brokerName = Cheese
```

See ???? for a description of all of the available properties.

Starting a Broker on Windows

Overview

On Windows the command to start a Fuse Message Broker broker is simple. It allows you to start a broker instance as a foreground process using the specified configuration.

The Unix start command provides more options. See ????.

Starting a foreground broker

On Windows, the **activemq** tool to starts an instance of Fuse Message Broker in the foreground. The syntax for the command is shown in Example 3.4 on page 27.

Example 3.4. Syntax for Starting a Broker on Windows

`activemq` {[xbean:file:*`confURI`* [?validate= {[true] | [false]}]] | [broker:*`brokerURI`*] | [properties:*`propURI`*]}

Tip

For a full description of the parameters see ????.

For example, to start a broker using the default broker configuration, enter:

```
c:\fmb> activemq xbean:file:conf/activemq.xml
```

You can optionally disable schema validation of the configuration file using the `validate` flag, as follows:

```
c:\fmb> activemq xbean:file:conf/activemq.xml?validate=false
```

Starting a background broker

On Windows the **activemq** command does not provide a mechanism for launching a broker instance as a background process. To start a broker in the background you can either:

- install the broker instance as a Windows service. See ????.
- Use the Windows **start** command to spawn the broker in a new command window.

Starting a Broker on Unix/Linux/OS X

Overview

The command to start a broker on Unix and Unix-like platforms provides more flexibility than the Windows start-up command. On Unix, you can start a broker in either the foreground or in the background as a daemon process. In addition, the Unix start-up command can run pre-start scripts and associate an effective user to the broker process.

Starting a background broker

On Unix, the **activemq start** command to starts an instance of Fuse Message Broker in the background. The syntax for the command is shown in Example 3.5 on page 28.

Example 3.5. Syntax for Starting a Daemon Broker on Unix

`activemq start` {[xbean:file:*`confURI`* [?validate= {[true] | [false]}]] | [broker:*`brokerURI`*] | [properties:*`propURI`*]}

Tip

For a full description of the parameters see ????.

For example, to start a daemon broker using the default broker configuration, enter:

```
% activemq start xbean:file:conf/activemq.xml
```

You can optionally disable schema validation of the configuration file using the `validate` flag, as follows:

```
% activemq start xbean:file:conf/activemq.xml?validate=false
```

By default, the daemon broker runs as the current user. It is good practice to specify a special effective user for the broker. This special effective user should be given just enough privileges to run the broker. This provides added security by ensuring that the broker process has limited ability to corrupt sensitive part of your system.

The `ACTIVEMQ_USER` environment variable specifies the effective user for the broker instances. By default, the broker will run as the currently logged in user. See ????.

Starting a broker in the foreground

On Unix, the **activemq console** command starts an instance of Fuse Message Broker as a foreground process. The syntax for the command is shown in Example 3.6 on page 29.

Example 3.6. Syntax for Starting a Foreground Broker on Unix

`activemq console` {[xbean:file:*`confURI`* [?validate= {[true] | [false]}]] | [broker:*`brokerURI`*] | [properties:*`propURI`*]}

Tip

For a full description of the parameters see ????.

For example, you can start a broker in the foreground using the default broker configuration, `conf/activemq.xml`, as follows:

```
% activemq console xbean:conf/activemq.xml
```

Starting a Broker with Maven

Overview

When developing messaging applications it can be helpful to have a test broker start up automatically when running the application. This can be accomplished using Fuse Message Broker's Apache Maven plug-in. The plug-in can be used directly from the command line or it can be included as part of a project's POM.

The Maven plug-in treats the broker as a part of the project's build and test environment. Maven will download all of the Fuse Message Broker libraries from the configured Maven repositories. The plug-in will store all of the broker's data in the project's `target` folder.

Note

Starting broker in a production environment from Maven is not recommended.

Using the command line

If a basic broker will suffice for testing an application, the Fuse Message Broker Maven plug-in can be loaded from the command line as shown in Example 3.7 on page 30.

Example 3.7. Using the Maven Plug-in from the Command Line

```
mvn org.apache.activemq.tooling:maven-activemq-plugin:5.5.1-fuse-00-xx:run
```

This will create a broker that:

- listens for client connections on `tcp://localhost:61616`
- does not use JMX
- does not persist messages

Using the POM

If the basic broker provided by adding the plug-in on the command line is insufficient for test an application the plug-in can be included in the project's POM. When included in a POM, the Fuse Message Broker plug-in uses a configuration URI to configure the broker. The plug-in can also set system properties when starting the broker.

Table 3.1 on page 31 describes the configuration properties used by the Fuse Message Broker Maven plug-in.

Table 3.1. Maven Configuration Properties

Property	Default	Description
`configUri`	broker:(tcp://localhost:6166)?useJmx=false&persistent=false	Specifies the configuration URI used to configure the broker. See ???? for more information.
`fork`	false	Specifies whether the broker is started in a separate thread. Having the broker start in a separate thread can be useful for integration testing.
`systemProperties`		Specifies a collection of system properties that will be set.

To start the configured broker when running the project use **mvn activemq:run**.

Example 3.8 on page 31 shows a POM fragment that configures the Fuse Message Broker plug-in to start a broker using the `activemq.xml` configuration file on the project's class path.

Example 3.8. Using the Maven Plug-in from a POM

```
<project ... >

  ...

  <repositories>
  <!-- FuseSource maven repositories -->
    <repository>
      <id>fusesource.releases</id>
      <name>FuseSoure releases repository</name>
      <url>http://repo.fusesource.com/maven2/</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </repository>
    <repository>
      <id>fusesource.snapshots</id>
      <name>FuseSource Snapshot Repository</name>
      <url>http://repo.fusesource.com/maven2-snapshot</url>
      <snapshots>
        <enabled>true</enabled>
      </snapshots>
```

```
    <releases>
      <enabled>false</enabled>
    </releases>
  </repository>
</repositories>

<pluginRepositories>
<!-- FuseSource maven repositories -->
  <pluginRepository>
    <id>fusesource.releases</id>
    <name>FuseSoure releases repository</name>
    <url>http://repo.fusesource.com/maven2/</url>
    <releases>
      <enabled>true</enabled>
    </releases>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </pluginRepository>
  <pluginRepository>
    <id>fusesource.snapshots</id>
    <name>FuseSource Snapshot Repository</name>
    <url>http://repo.fusesource.com/maven2-snapshot</url>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
    <releases>
      <enabled>false</enabled>
    </releases>
  </pluginRepository>
</pluginRepositories>

...

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.activemq.tooling</groupId>
      <artifactId>maven-activemq-plugin</artifactId>
      <version>5.5</version>
      <configuration>
        <configUri>xbean:activemq.xml</configUri>
        <fork>false</fork>
      </configuration>
      <dependencies>
        <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring</artifactId>
          <version>2.5.5</version>
```

```
          </dependency>
          <dependency>
            <groupId>org.mortbay.jetty</groupId>
            <artifactId>jetty-xbean</artifactId>
            <version>6.1.11</version>
          </dependency>
          <dependency>
            <groupId>org.apache.camel</groupId>
            <artifactId>camel-activemq</artifactId>
            <version>1.1.0</version>
          </dependency>
        </dependencies>
      </plugin>
    </plugins>
  </build>
</project>
```

Chapter 4. Shutting Down a Broker

The Fuse Message Broker's shutdown tool uses JMX to locate and gracefully shutdown brokers running as daemon processes.

Important

The administration tool requires that brokers have JMX enabled. By default, JMX is enabled.

Shutting Down a Broker on Windows

Overview

On Windows platforms brokers are run as either foreground processes or as system services. You can shutdown running brokers using operating system actions. For example, typing **CTRL**+**C** in a window in which a foreground broker is running will shutdown the broker. Closing the window will also shutdown a foreground broker.

However, if you want to properly shutdown a broker use the administration tool's **stop** task. It ensures that the broker is gracefully shutdown. It also allows you to shutdown brokers remotely.

The stop task

On Windows, the command to shutdown a broker is **activemq-admin stop**. The syntax for the command is shown in Example 4.1 on page 36.

Example 4.1. Syntax for Stopping a Broker on Windows

`activemq-admin stop` {[*`brokerName`*] | [--all]} [--jmxurl *`JMXUrl`*] [-jmxuser *`userName`*] [-jmxpassword *`password`*]

Tip

For a full description of the parameters see ????.

If you do not use the *`--jmxurl`* parameter, the default JMX url, `service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi`, is used. This is the JMX url specified in the default Fuse Message Broker configuration.

If you configured a broker to use a different JMX url, you must use the *`--jmxurl`* parameter and provide the JMX url for connecting to the broker.

If you have secured JMX access to the broker, you will need to use the *`-jmxuser`* and *`-jmxpassword`* parameters. They specify the user name and password required to access the broker's JMX context. For information about using security with JMX see "JMX Security" in *ActiveMQ Security Guide*.

Examples

The command shown in Example 4.2 on page 37 shuts down a broker named `gatewayEast` that is running in the default JMX context.

Example 4.2. Shutting Down a Broker on Windows

```
c:\activemq-admin stop gatewayEast
```

The command shown in Example 4.3 on page 37 shuts down all of the brokers running in the default JMX context.

Example 4.3. Shutting Down All Broker on Windows

```
c:\activemq-admin stop --all
```

The command shown in Example 4.4 on page 37 shuts down a broker named `gatewayEast` that is running in the JMX context `service:jmx:rmi:///jndi/rmi://localhost:8050/jmxrmi`.

Example 4.4. Shutting Down a Broker on Windows in a Non-default JMX Context

```
c:\activemq-admin stop gatewayEast --jmxurl service:jmx:rmi:///jndi/rmi://localhost:8050/jmxrmi
```

Shutting Down a Broker on Unix/Linux/OS X

Overview

The administration tool's **stop** task uses JMX to locate and shutdown brokers on Unix and Unix-like platforms. This saves you from remembering a broker's PID each time it starts up.

The Unix **stop** task can also be configured to force a shutdown if an orderly shutdown hangs. It does this by waiting a predetermined amount of time and then killing the broker process.

The stop task

On Windows, the command to shutdown a broker is **activemq stop**. The syntax for the command is shown in Example 4.5 on page 38.

Example 4.5. Syntax for Stopping a Broker on Unix/Linux/OS X

`activemq stop` {[*brokerName*] | [--all]} [--jmxurl *JMXUrl*] [-jmxuser *userName*] [-jmxpassword *password*]

Tip

For a full description of the parameters see ????.

If you do not use the `--jmxurl` parameter, the default JMX url, `service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi`, is used. This is the JMX url specified in the default Fuse Message Broker configuration.

If you configured a broker to use a different JMX url, you must use the `--jmxurl` parameter and provide the JMX url for connecting to the broker.

If you have secured JMX access to the broker, you will need to use the `-jmxuser` and `-jmxpassword` parameters. They specify the user name and password required to access the broker's JMX context. For information about using security with JMX see "JMX Security" in *ActiveMQ Security Guide*.

Shutdown timeout

When you execute the **stop** task, the administration tool initiates an orderly shutdown of the broker (or brokers) by sending a shutdown request to the JMX port. If the broker fails to shut down within a specified timeout, the administration tool sends a `SIGKILL` signal to the broker process.

The duration of the kill timeout is set by the `ACTIVEMQ_KILL_MAXSECONDS` environment variable. For more information about setting up your environment for the administration tool see ????.

Examples

The command shown in Example 4.6 on page 39 shuts down a broker named `gatewayEast` that is running in the default JMX context.

Example 4.6. Shutting Down a Broker on Unix/Linux/OS X

```
>activemq stop gatewayEast
```

The command shown in Example 4.7 on page 39 shuts down all of the brokers running in the default JMX context.

Example 4.7. Shutting Down All Broker on Unix/Linux/OS X

```
>activemq stop --all
```

The command shown in Example 4.8 on page 39 shuts down a broker named `gatewayEast` that is running in the JMX context `service:jmx:rmi:///jndi/rmi://localhost:8050/jmxrmi`.

Example 4.8. Shutting Down a Broker in a Non-default JMX Context on Unix/Linux/OS X

```
>activemq stop gatewayEast --jmxurl service:jmx:rmi:///jndi/rmi://localhost:8050/jmxrmi
```

Appendix A. Administration Tool Commands

Name

activemq-admin create — creates a new instance of the broker's file structure

Synopsis

`activemq-admin create` [--amqconf *`confFile`*] [--version] [[--help] | [-h] | [-?]] {*`brokerPath`*}

Arguments

Argument	Description
--amqconf *`confFile`*	Specifies the configuration file to be copied to the new broker instance.
--version	Displays the version of the tool.
--help, -?, -h	Displays the help for the command.
`brokerPath`	Specifies the path of the new broker instance.

Name

activemq — starts a foreground broker on Windows

Synopsis

`activemq` [-D*name*=*value*...] [--version] [[--help] | [-h] | [-?]] {[xbean:file:*confURI* [?validate= {[true] | [false]}]] | [broker:*brokerURI*] | [properties:*propURI*]}

Arguments

Argument	Description
-D*name*=*value*	Specifies a system property to be interpreted by the VM.
--version	Displays the version of the tool.
--help, -?, -h	Displays the help for the command.
xbean:file:*confURI*	Specifies the path of the broker configuration file to use.
broker:*brokerURI*	Specifies the broker URI to use.
properties:*propURI*	Specifies the path to a properties file to configure the broker.

Name

activemq start — starts a daemon broker on Unix

Synopsis

`activemq start` [-D*name*=*value*...] [--version] [[--help] | [-h] | [-?]] {[xbean:file:*confURI* [?validate= {[true] | [false]}]] | [broker:*brokerURI*] | [properties:*propURI*]}

Arguments

Argument	Description
-D*name*=*value*	Specifies a system property to be interpreted by the VM.
--version	Displays the version of the tool.
--help, -?, -h	Displays the help for the command.
xbean:file:*confURI*	Specifies the path of the broker configuration file to use.
broker:*brokerURI*	Specifies the broker URI to use.
properties:*propURI*	Specifies the path to a properties file to configure the broker.

Name

activemq console — starts a foreground broker on Unix

Synopsis

`activemq console` [-D*name*=*value*...] [--version] [[--help] | [-h] | [-?]] {[xbean:file:*confURI*] | [broker:*brokerURI*] | [properties:*propURI*]}

Arguments

Argument	Description
-D*name*=*value*	Specifies a system property to be interpreted by the VM.
--version	Displays the version of the tool.
--help, -?, -h	Displays the help for the command.
xbean:file:*confURI*	Specifies the path of the broker configuration file to use.
broker:*brokerURI*	Specifies the broker URI to use.
properties:*propURI*	Specifies the path to a properties file to configure the broker.

Name

activemq stop — shuts down a running broker on Unix

Synopsis

`activemq stop` {[*brokerName*] | [--all]} [--jmxurl *JMXUrl*] [-jmxuser *userName*] [-jmxpassword *password*] [-D*name*=*value*...] [--version] [[--help] | [-h] | [-?]]

Arguments

Option	Description
brokerName	Specifies the name of the broker to be stopped.
`--all`	Stop all brokers running in the default JMX context.
`--jmxurl <`*URL*`>`	Sets the JMX URL use to locate the broker. The default is `service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi`
`--jmxuser <`*user*`>`	Sets the JMX user, used for authentication.
`--jmxpassword <`*password*`>`	Sets the JMX password, used for authentication.
`-D<`*prop*`>=<`*value*`>`	Sets a Java system property. For example, `-Dactivemq.home=C:/ActiveMQ`.
`--version`	Displays the version information.
`-h, -?, --help`	Displays the online help for this command

Name

activemq restart — restarts a running broker on Unix

Synopsis

`activemq restart` {[*`brokerName`*] | [--all] | [--jmxurl *`JMXUrl`*]} [-jmxuser *`userName`*] [-jmxpassword *`password`*] [-D*`name`*=*`value`*...] [--version] [[--help] | [-h] | [-?]]

Arguments

Option	Interpretation
`--all`	Restart all brokers running in the default JMX context.
`--jmxurl <URL>`	Sets the JMX URL to connect to.
`--jmxuser <user>`	Sets the JMX user, used for authentication.
`--jmxpassword <password>`	Sets the JMX password, used for authentication.
`-D<prop>=<value>`	Sets a Java system property. For example, `-Dactivemq.home=C:/ActiveMQ`.
`--version`	Displays the version information.
`-h, -?, --help`	Displays the online help for this command

Name

activemq-admin stop — shuts down a running broker on Windows

Synopsis

`activemq-admin stop` {[*`brokerName`*] | [--all]} [--jmxurl *`JMXUrl`*] [-jmxuser *`userName`*] [-jmxpassword *`password`*] [-D*`name`*=*`value`*...] [--version] [[--help] | [-h] | [-?]]

Arguments

Option	Interpretation
`brokerName`	Specifies the name of the broker to be stopped.
`--all`	Stop all brokers running in the default JMX context.
`--jmxurl <`*`URL`*`>`	Sets the JMX URL use to locate the broker. The default is `service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi`
`--jmxuser <`*`user`*`>`	Sets the JMX user, used for authentication.
`--jmxpassword <`*`password`*`>`	Sets the JMX password, used for authentication.
`-D<`*`prop`*`>=<`*`value`*`>`	Sets a Java system property. For example, `-Dactivemq.home=C:/ActiveMQ`.
`--version`	Displays the version information.
`-h, -?, --help`	Displays the online help for this command

Name

activemq-admin list — lists the available brokers

Synopsis

`activemq-admin list` [--jmxurl *`JMXUrl`*] [-jmxuser *`userName`*] [-jmxpassword *`password`*] [-D*`name`*=*`value`*...] [--version] [[--help] | [-h] | [-?]]

Arguments

Option	Interpretation
`--jmxurl <`*`URL`*`>`	Sets the JMX URL to connect to.
`--jmxuser <`*`user`*`>`	Sets the JMX user, used for authentication.
`--jmxpassword <`*`password`*`>`	Sets the JMX password, used for authentication.
`-D<`*`prop`*`>=<`*`value`*`>`	Sets a Java system property. For example, `-Dactivemq.home=C:/ActiveMQ`.
`--version`	Displays the version information.
`-h, -?, --help`	Displays the online help for this command

Name

activemq-admin query — queries the broker for information on specific objects

Synopsis

`activemq-admin query` [-Q*MBeanType*=*name*] [-xQ*MBeanType*=*name*] [--objname *query*] [--xobjname *query*] [--view {*attr*...}] [--jmxurl *JMXUrl*] [-jmxuser *userName*] [-jmxpassword *password*] [-D*name*=*value*...] [--version] [[--help] | [-h] | [-?]]

Arguments

Option	Interpretation
`-Q <type>=<name>`	Adds to the search list the specific object type matched by the defined object identifier.
`-xQ <type>=<name>`	Removes from the search list the specific object type matched by the object identifier.
`--objname <query>`	Adds to the search list objects matched by the query similar.
`--xobjname <query>`	Removes from the search list objects matched by the query.
`--view <attr1>, <attr2>,...`	Selects the specific attribute of the object to view; by default, all attributes are displayed.
`--jmxurl <URL>`	Sets the JMX URL to connect to.
`--jmxuser <user>`	Sets the JMX user, used for authentication.
`--jmxpassword <password>`	Sets the JMX password, used for authentication.
`--jmxlocal`	Use the local JMX server instead of a remote server.
`-D<prop>=<value>`	Sets a Java system property. For example, `-Dactivemq.home=C:/ActiveMQ`.
`--version`	Displays the version information.
`-h, -?, --help`	Displays the online help for this command.

Name

activemq-admin bstat — summarizes the statistics for a broker

Synopsis

`activemq-admin bstat` [--jmxurl *`JMXUrl`*] [-jmxuser *`userName`*] [-jmxpassword *`password`*] [-D*`name`*=*`value`*...] [--version] [[--help] | [-h] | [-?]]

Arguments

Option	Interpretation
`--jmxurl <`*`URL`*`>`	Sets the JMX URL to connect to.
`--jmxuser <`*`user`*`>`	Sets the JMX user, used for authentication.
`--jmxpassword <`*`password`*`>`	Sets the JMX password, used for authentication.
`--jmxlocal`	Use the local JMX server instead of a remote server.
`-D<`*`prop`*`>=<`*`value`*`>`	Sets a Java system property. For example, `-Dactivemq.home=C:/ActiveMQ`.
`--version`	Displays the version information.
`-h, -?, --help`	Displays the online help for this command

Name

activemq-admin browse — browse the contents of a destination

Synopsis

`activemq-admin browse` {--amqurl *`brokerURL`*} [--msgsel {msgsel...}] [--view {*`attr`*...}] [[-Vheader] | [-Vcustom] | [-Vbody]] [--version] [[--help] | [-h] | [-?]] *`destName`*

Arguments

Option	Interpretation
`--amqurl <`*`brokerURL`*`>`	Sets the URL for the broker you are connecting to.
`--msgsel <`*`msgsel1, msgsel12`*`>`	Adds to the search list messages matched by the query similar to the message selector format.
`-Vheader`, `-Vcustom`, `-Vbody`	A predefined view that enables you to view the message header, custom message header, or message body.
`--view <`*`attr1`*`>,<`*`attr1`*`>,...`	Selects the specific attribute of the message to view.
`-D<`*`prop`*`>=<`*`value`*`>`	Sets a Java system property. For example, `-Dactivemq.home=C:/ActiveMQ`.
`--version`	Displays the version information.
`-h, -?, --help`	Displays the online help for this command.

Name

activemq-admin journal-audit — audits the log files of the persistent journal

Synopsis

`activemq-admin journal-audit` [--message-format=*`VelocityTemplate`*]
[--topic-ack-format=*`VelocityTemplate`*] [--queue-ack-format=*`VelocityTemplate`*]
[--transaction-format=*`VelocityTemplate`*] [--trace-format=*`VelocityTemplate`*] [--where=*`JoSQLExp`*]
[-D*`name`*=*`value`*...] [--version] [[--help] | [-h] | [-?]] journalDir

Arguments

Option	Interpretation
`--message-format=`*`VelocityTemplate`*	Specifies the Apache Velocity[1] template used to format the displayed messages.
`--topic-ack-format=`*`VelocityTemplate`*	Specifies the Apache Velocity[2] template used to display topic ack. messages.
`--queue-ack-format=`*`VelocityTemplate`*	Specifies the Apache Velocity[3] template used to display queue ack. messages.
`--transaction-format=`*`VelocityTemplate`*	Specifies the Apache Velocity[4] template used to display transaction records.
`--trace-format=`*`VelocityTemplate`*	Specifies the Apache Velocity[5] template used to display trace records.
`--where=`*`JoSQLExp`*	Select the records to display, using a SQL-like syntax implemented by JoSQL[6].
`-D<`*`prop`*`>=<`*`value`*`>`	Sets a Java system property. For example, `-Dactivemq.home=C:/ActiveMQ`.
`-h, -?, --help`	Displays the online help for this command.

[1] http://velocity.apache.org/
[2] http://velocity.apache.org/
[3] http://velocity.apache.org/
[4] http://velocity.apache.org/
[5] http://velocity.apache.org/
[6] http://josql.sourceforge.net/

Name

activemq-admin purge — purges messages from a destination

Synopsis

`activemq-admin purge` [--jmxurl *`JMXUrl`*] [-jmxuser *`userName`*] [-jmxpassword *`password`*] [--msgsel {msgsel...}] [-D*`name`*=*`value`*...] [--version] [[--help] | [-h] | [-?]] destName

Arguments

Option	Interpretation
`--jmxurl <`*`URL`*`>`	Sets the JMX URL to connect to.
`--jmxuser <`*`user`*`>`	Sets the JMX user, used for authentication.
`--jmxpassword <`*`password`*`>`	Sets the JMX password, used for authentication.
`--msgsel <`*`msgsel1, msgsel12`*`>`	Adds to the search list messages matched by the query similar to the message selector format.
`-D<`*`prop`*`>=<`*`value`*`>`	Sets a Java system property. For example, `-Dactivemq.home=C:/ActiveMQ`.
`--version`	Displays the version information.
`-h, -?, --help`	Displays the online help for this command

Appendix B. Broker Properties

Global properties

Table B.1 on page 55 describes the broker properties you can set as options on a broker URI or in a broker properties file.

Table B.1. Broker URI Properties

Property	Default	Description
`useJmx`	true	Specifies if the broker will connect to a JMX server.
`persistent`	true	Specifies whether the broker uses persistent storage
`populateJMSXUserID`	false	Specifies whether the broker populates the `JMSXUserID` property of messages to indicate the authenticated sender username of the person who sent the message
`useShutdownHook`	true	Specifies if the broker installs a shutdown hook so that it can properly shut itself down on a JVM kill.
`brokerName`	localhost	Specifies the name of the broker
`deleteAllMessagesOnStartup`	false	Specifies if all the messages in the persistent store will be deleted on broker startup.
`enableStatistics`	true	Specifies if statistics gathering is enabled.

Property file specific

Table B.2 on page 56 describes additional properties that you can set in a broker properties file.

Table B.2. Property File Properties

Property	Default	Description
`transportConnectors`	`tcp:\\localhost:61616`	Specifies a comma separated list of transport URIs on which the broker will listen for client connections. See *Broker Client Connectivity Guide* for more information.
`networkConnectors`		Specifies a comma separated list of URIs on which the broker will listen for connections with other brokers. See "Broker Networks" in *Clustering Guide* for more information.

Index