

FUSE[™] Message Broker

FUSE[™] Message Broker Configuration Guide

Version 5.2
December 2008

FUSETM Message Broker Configuration Guide

Progress Software

Version 5.2

Published 02 Dec 2008

Copyright © 2008 IONA Technologies PLC , a wholly-owned subsidiary of Progress Software Corporation.

Legal Notices

Progress Software Corporation and/or its subsidiaries may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this publication. Except as expressly provided in any written license agreement from Progress Software Corporation, the furnishing of this publication does not give you any license to these patents, trademarks, copyrights, or other intellectual property. Any rights not expressly granted herein are reserved.

Progress, IONA, IONA Technologies, the IONA logo, Orbix, High Performance Integration, Artix, FUSE, and Making Software Work Together are trademarks or registered trademarks of Progress Software Corporation and/or its subsidiaries in the US and other countries.

Java and J2EE are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. CORBA is a trademark or registered trademark of the Object Management Group, Inc. in the US and other countries. All other trademarks that appear herein are the property of their respective owners.

While the information in this publication is believed to be accurate Progress Software Corporation makes no warranty of any kind to this material including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Progress Software Corporation shall not be liable for errors contained herein, or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

All products or services mentioned in this manual are covered by the trademarks, service marks, or product names as designated by the companies who market those products.

No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, photocopying, recording or otherwise, without prior written consent of IONA Technologies PLC. No third-party intellectual property right liability is assumed with respect to the use of the information contained herein. IONA Technologies PLC assumes no responsibility for errors or omissions contained in this publication. This publication and features described herein are subject to change without notice. Portions of this document may include Apache Foundation documentation, all rights reserved.

Table of Contents

Preface	9
The FUSE Message Broker Library	10
Open Source Project Resources	11
Document Conventions	12
Configuring the Broker	15
XML Configuration	16
URI Configuration	20
Properties File Configuration	21
Configuring an Embedded Broker	22
Configuring Transports	25
Configuring Clients	27
Index	29

DRAFT

List of Tables

- 1. URI Configuration: Broker Options 20
- 2. Available URI Values for BrokerFactory Class 22

DRAFT

DRAFT

List of Examples

1. Example activemq.xml	16
2. Configuring Persistence	19
3. Configuring Startup Destinations	19
4. Example Properties URIs	21
5. Example Properties File	21
6. Configuring an Embedded Broker in Code	22

DRAFT

Preface

The FUSE Message Broker Library	10
Open Source Project Resources	11
Document Conventions	12

DRAFT

The FUSE Message Broker Library

The FUSE Message Broker documentation library consists of the following books:

- [Installing FUSE™ Message Broker](#) discusses the requirements and procedures for installing FUSE Message Broker
- [Getting Started with FUSE™ Message Broker](#) provides an overview of the central concepts behind FUSE Message Broker and walks you through a simple example.
- [Connectivity Guide](#) explains the different wire protocols and transports that FUSE Message Broker supports.
- [Using FUSE™ Message Broker's Persistence Features](#) describes how to enable message persistence using the AMQ Message Store or a relational database in FUSE Message Broker.

Open Source Project Resources

Apache CXF

Web site: <http://cxf.apache.org/>

User's list: <user@cxf.apache.org>

Apache Tomcat

Web site: <http://tomcat.apache.org/>

User's list: <users@tomcat.apache.org>

Apache ActiveMQ

Web site: <http://activemq.apache.org/>

User's list: <users@activemq.apache.org>

Apache Camel

Web site:
<http://activemq.apache.org/camel/enterprise-integration-patterns.html>

User's list: <camel-user@activemq.apache.org>

Apache ServiceMix

Web site: <http://servicemix.apache.org>

User's list: <users@servicemix.apache.org>

Document Conventions

Typographical conventions

This book uses the following typographical conventions:

<code>fixed width</code>	<p>Fixed width (Courier font) in normal text represents portions of code and literal names of items such as classes, functions, variables, and data structures. For example, text might refer to the <code>javax.xml.ws.Endpoint</code> class.</p> <p>Constant width paragraphs represent code examples or information a system displays on the screen. For example:</p> <pre>import java.util.logging.Logger;</pre>
<i>Fixed width italic</i>	<p>Fixed width italic words or characters in code and commands represent variable values you must supply, such as arguments to commands or path names for your particular system. For example:</p> <pre>% cd /users/YourUserName</pre>
<i>Italic</i>	Italic words in normal text represent <i>emphasis</i> and introduce <i>new terms</i> .
Bold	Bold words in normal text represent graphical user interface components such as menu commands and dialog boxes. For example: the User Preferences dialog.

Keying conventions






This book uses the following keying conventions:

No prompt	When a command's format is the same for multiple platforms, the command prompt is not shown.
%	A percent sign represents the UNIX command shell prompt for a command that does not require root privileges.
#	A number sign represents the UNIX command shell prompt for a command that requires root privileges.
>	The notation > represents the MS-DOS or Windows command prompt.
...	Horizontal or vertical ellipses in format and syntax descriptions indicate that material has been eliminated to simplify a discussion.
[]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list from which you must choose an item in format and syntax descriptions.

	In format and syntax descriptions, a vertical bar separates items in a list of choices enclosed in {} (braces).
--	---

Admonition conventions

This book uses the following conventions for admonitions:

	Notes display information that may be useful, but not critical.
	Tips provide hints about completing a task or using a tool. They may also provide information about workarounds to possible problems.
	Important notes display information that is critical to the task at hand.
	Cautions display information about likely errors that can be encountered. These errors are unlikely to cause damage to your data or your systems.
	Warnings display information about errors that may cause damage to your systems. Possible damage from these errors include system failures and loss of data.

DRAFT

Configuring the Broker

In FUSE Message Broker, there are two ways of configuring a broker: using an XML file and using a URI.

XML Configuration	16
URI Configuration	20
Properties File Configuration	21
Configuring an Embedded Broker	22

DRAFT

XML Configuration

The recommended way to configure FUSE Message Broker is by using an XBean XML configuration file. XBean is an extension of the Spring Framework that has allowed the developers of Apache ActiveMQ to develop a syntax that is less verbose and yet more expressive than basic Spring configuration.

By default, FUSE Message Broker uses the `activemq.xml` configuration file stored in the `InstallDir/conf` directory.

Specifying an alternative configuration file

You can specify an alternative XML configuration file when running FUSE Message Broker by using the `xbean:file` command line option, as follows:

```
activemq xbean:file:PathToXmlConfigFile
```

For example:

```
activemq
xbean:file:C:/iona/fuse-message-broker-5.1.0.0/conf/activemq.xml
```

Configuration file format

The broker XML configuration file is comprised of a root `beans` element followed by a nested `broker` element.

Example 1. Example `activemq.xml`

```
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:amq="http://activemq.org/config/1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
    http://activemq.org/config/1.0 http://activemq.apache.org/schema/activemq-core.xsd
    http://activemq.apache.org/camel/schema/spring http://activemq.apache.org/camel/schema/spring/camel-spring.xsd">

  <!-- Allows us to use system properties as variables in this
  configuration file -->
  <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"/>

  <broker xmlns="http://activemq.org/config/1.0" broker
```



```

Name="localhost" dataDirectory="${activemq.base}/data"

    <!-- The transport connectors ActiveMQ will listen to -->
    <transportConnectors>
      <transportConnector name="openwire" uri="tcp://local
host:61616"/>
      <transportConnector name="ssl"      uri="ssl://local
host:61617"/>
      <transportConnector name="stomp"    uri="stomp://local
host:61613"/>
      <transportConnector name="xmpp"    uri="xmpp://local
host:61222"/>
    </transportConnectors>

    <!-- The store and forward broker networks ActiveMQ will
listen to -->
    <networkConnectors>
      <!-- by default just auto discover the other brokers -->
      <networkConnector name="default-nc" uri="multicast://de
fault"/>
      <networkConnector name="host1 and host2" uri="stat
ic://(tcp://host1:61616,tcp://host2:61616)"/>
    </networkConnectors>

  </broker>
</beans>

```

Full details of elements and attributes that you may find in the configuration file can be found in the [XML Configuration Reference](#) [../configref/index.html]>. The following sections detail the most important tags.

The broker element

The `broker` element identifies and configures an instance of a broker.

Specifying more than one `broker` element results in multiple broker instances running in the same virtual machine.

`brokerName`

Specifies a unique name for the broker instance. Be sure not to use special characters, such as underscores, as they are converted to URIs which do not allow things like underscores in them.

`dataDirectory`

Specifies the directory where data files for the JDBC and Journal persistence adaptors are stored.

Transport connectors

The `transportConnectors` element contains one or more nested `transportConnector` elements, each of which specifies an IP address and port number on which the broker listens and accepts network connection requests from clients.

The `transportConnector` element takes the following attributes:

`uri`

Specifies the URI for this transport connector. For example, `tcp://localhost:61616`. Required.

`discoveryURI`

Enables a discovery agent for this transport connector. The broker will listen for discovery advertisements from other brokers using this URI.

`name`

The name assigned to the `transportConnector`.

Network connectors

The `networkConnectors` element contains one or more nested `networkConnector` elements, each of which specifies an IP address and port number on which the broker creates connections to other brokers. The following attributes are required.

`name`

The name of the network connector.

`uri`

The URI for this network connector. You must use a static, rendezvous, or multicast URI

Configuring persistence

You can configure a number of message persistence implementations using the `persistenceAdapter` element. See [Using FUSE™ Message Broker's Persistence Features](#) for details.

Example 2. Configuring Persistence

```
<persistenceAdapter>  
  <journalJDBC journalLogFiles="5" dataDirectory="{activemq.base}/activemq-data" dataSource="#postgres-ds"/>  
</persistenceAdapter>
```

Startup destinations

Typically in FUSE Message Broker, destinations are created on demand. However, you can configure which destinations are available when the broker starts up using the `destinations` element.

Example 3. Configuring Startup Destinations

```
<destinations>  
  <queue physicalName="FOO.BAR" />  
  <topic physicalName="SOME.TOPIC" />  
</destinations>
```

The `name` and `physicalName` attributes should match the corresponding names in the client's JNDI context.

URI Configuration

You can configure a broker by appending a URL to the **activemq** command using the following syntax:

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

For example:

```
activemq  
broker:(tcp://localhost:61616,network-static:tcp://remotehost:61616)?persistent=false&useJmx=true
```

Broker options

You can include the following broker options to a configuration URI.

Table 1. URI Configuration: Broker Options

Option	Default Value	Description
useJmx	true	Expose the broker to JMX
persistent	true	Enable the broker to use persistent storage
populateJMSXUserID	false	Have the broker populate the JMSXUserID property of messages to indicate the authenticated username of the sender
useShutdownHook	true	Have the broker install a shutdown hook so that it can properly shut itself down on a JVM kill
brokerName	localhost	The name of the broker
deleteAllMessagesOnStartup	false	Delete all the messages in the persistent store when the broker starts up
enableStatistics	true	Enable statistics gathering

Properties File Configuration

You can run a configured broker by referencing a properties file, using the following syntax:

activemq properties:*PropertiesFile*

The *PropertiesFile* value can point to a local file or to a URL resource.

Examples

Example 4. Example Properties URIs

```
properties:/brokers/broker1.properties  
properties:broker1.properties  
properties:http://example.com/broker1.properties
```

Example 5. Example Properties File

```
useJmx = false  
persistent = false  
brokerName = broker1
```

Configuring an Embedded Broker

Configuring through code

You can fully configure an embedded broker through application code, as shown in [Example 6 on page 22](#).

Example 6. Configuring an Embedded Broker in Code

```
BrokerService broker = new BrokerService();
broker.setBrokerName("broker1");
broker.setUseShutdownHook(false);
//Add plugin
broker.setPlugins(new BrokerPlugin[]{new JaasAuthentication
Plugin()});
//Add a network connection
NetworkConnector connector = answer.addNetworkConnector("stat
ic://"+"tcp://broker2:61616");
connector.setDuplex(true);
broker.addConnector("tcp://localhost:61616");
broker.start();
```



Note

Always add plug-ins before connectors, otherwise the plug-ins will not be initialized.

Using the BrokerFactory class

You can use the `BrokerFactory` class to create and configure a broker using the following syntax:

```
BrokerService broker = BrokerFactory.createBroker(new
URI (URI));
```

The available values of the URI are:

Table 2. Available URI Values for BrokerFactory Class

URI scheme	Example	Description
xbean:	xbean:activemq.xml	Searches the classpath for an XML document (activemq.xml) with the given URI, which is then

URI scheme	Example	Description
		used as the XML Configuration on page 16
file:	file:brokers/conf/activemq.xml	Loads the given file as the XML Configuration on page 16
broker:	broker:tcp://localhost:61616	Configures the broker using URI Configuration on page 20

DRAFT

Configuring Transports

In FUSE Message Broker all connectors whether a transport connector or a network connector, are configured through a uniform resource identifier (URI).

This chapter maps to the following section of the ActiveMQ Wiki:

<http://activemq.apache.org/configuring-version-5-transports.html>

DRAFT

DRAFT

Configuring Clients

In FUSE Message Broker, you configure clients using

DRAFT

DRAFT

Index

DRAFT

DRAFT