



FUSE Message Broker

Using FUSE Message Broker's Persistence Features

Version 5.1
July 2008

Making Software Work Together™

Using FUSE Message Broker's Persistence Features

IONA Technologies

Version 5.1

Published 16 Jul 2008

Copyright © 2001-2008 IONA Technologies PLC

Trademark and Disclaimer Notice

IONA Technologies PLC 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 IONA Technologies PLC, 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.

IONA, IONA Technologies, the IONA logo, Orbix, High Performance Integration, Artix, FUSE, and Making Software Work Together are trademarks or registered trademarks of IONA Technologies PLC and/or its subsidiaries.

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 United States 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, IONA Technologies PLC 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. IONA 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.

Copyright Notice

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	11
The FUSE Message Broker Library	12
Open Source Project Resources	13
Document Conventions	14
Introduction to FUSE Message Broker Persistence	17
Using the AMQ Message Store	19
Using JDBC to Connect to a Database Store	25
Basics of Using the JDBC Persistence Adapter	26
Using JDBC with the High Performance Journal	32
Using JDBC without the Journal	35
Message Cursors	37
Types of Cursors	38
Configuring the Type of Cursor Used by a Destination	42
Index	45

List of Figures

1. Overview of the AMQ Message Store	19
2. AMQ Message Store Directory Layout	20
3. Store-based Cursors for a Fast Consumer	38
4. Store-based Cursors for a Slow Consumer	39
5. VM Cursors	40
6. File-based Cursors	41

List of Tables

1. Setting a Broker's Persistence	17
2. Configuration Attributes for the AMQ Message Store	22
3. Statements for Configuring the SQL Statements Used by the JDBC Persistence Adapter	28
4. Attributes for Configuring the Journaled JDBC Persistence Adapter	32
5. Attributes for Configuring the Plain JDBC Persistence Adapter	35
6. Elements for Configuring the Type of Cursor to Use for Durable Subscribers	42
7. Elements for Configuring the Type of Cursor to Use for Transient Subscribers	43
8. Elements for Configuring the Type of Cursor to Use for a Queue	44

List of Examples

1. Turning Off a Broker's Persistence	18
2. Adding Persistence Adapter Configuration	18
3. Configuring the AMQ Message Store	22
4. Configuration for Using the Default Database	27
5. Configuration for the Oracle JDBC Driver	27
6. Fine Tuning the Database Schema	28
7. Configuring a Generic JDBC Provider	31
8. Configuring FUSE Message Broker to use the Journaled JDBC Persistence Adapter	33
9. Configuring FUSE Message Broker to use the Plain JDBC Persistence Adapter	36
10. Configuring a Topic's Cursor Usage	43
11. Configuring a Queue's Cursor Usage	44

Preface

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 on page 1](#) 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://incubator.apache.org/cxf/>

User's list: <cxf-user@incubator.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.org/site/home.html>

User's list: <servicemix-users@geronimo.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>	<p>Italic words in normal text represent <i>emphasis</i> and introduce <i>new terms</i>.</p>
Bold	<p>Bold words in normal text represent graphical user interface components such as menu commands and dialog boxes. For example: the User Preferences dialog.</p>

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.

Introduction to FUSE Message Broker Persistence

Message persistence allows for the recovery of undelivered messages in the event of a system failure. By default, FUSE Message Broker's persistence features are activated. The default set up is fast and scalable. It is easy to customize the set-up to use and JDBC compliant database.

Overview

Loss of messages is not acceptable in mission critical applications. FUSE Message Broker reduces the risk of message loss by using a persistent message store by default. Persistent messages are written to the persistent store when they are sent. The messages persist in the store until their delivery is confirmed. This means that in the case of a system failure, FUSE Message Broker can recover all of the undelivered messages at the time of the failure.

Persistent message stores

The default message store is embeddable and transactional. It both very fast and extremely reliable. In addition to the default message store, FUSE Message Broker offers a number of other persistent message store options. These include:


- AMQ Message Store
 - a journaled JDBC adapter
 - a non-journaled JDBC adapter
-

Activating and deactivating persistence

Persistence in FUSE Message Broker is controlled by a broker's XML configuration file. To change a broker's persistence behavior you modify the configuration's `broker` element's `persistent` attribute.

Table 1. Setting a Broker's Persistence

Value	Description
true	The broker will use message persistence.

Value	Description
false	The broker will not use message persistence.
	 Important If you add persistence adapters to a broker's configuration, this setting is ignored.



Tip

By default, a broker's `persistent` attribute is set to `true`.

[Example 1 on page 18](#) shows a configuration snip-it for turning off a broker's message persistence.

Example 1. Turning Off a Broker's Persistence

```
<broker persistent="false" ...>
  ...
</broker>
```

Configuring persistence adapter behavior

FUSE Message Broker offers a number of different persistence mechanisms aside from the default message store. To use one of the alternative message stores, or to modify the default behavior of the default message store, you need to configure the persistence adapter. This is done by adding a `persistenceAdapter` element to the broker's configuration file as shown in [Example 2 on page 18](#).

Example 2. Adding Persistence Adapter Configuration

```
<broker persistent="true" ...>
  ...
  <persistenceAdapter>
    <amqpPersistenceAdapter ... />
  </persistenceAdapter>
  ...
</broker>
```

The `persistenceAdapter` element has no attributes. The configuration for the persistence adapter is specified using a child element for the desired persistence adapter.

Using the AMQ Message Store

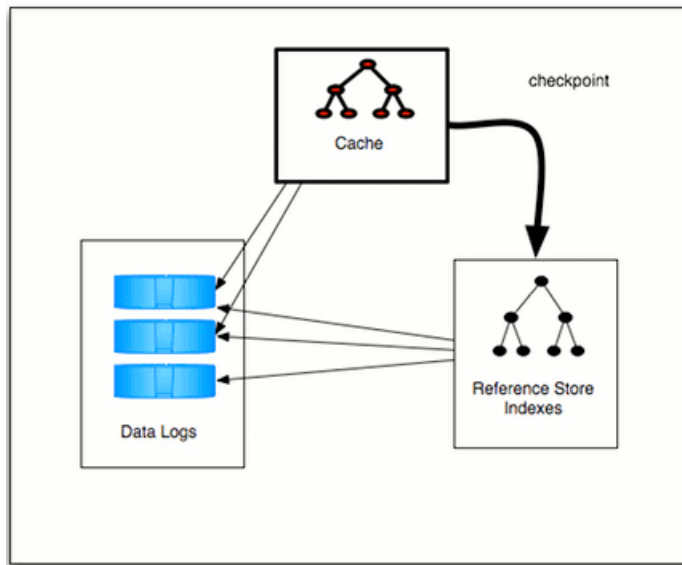
The default message store used by FUSE Message Broker is a light-weight transactional store that is fast and reliable. It is a hybrid system that couples a transactional journal for message storage and a reference store for quick retrieval. The AMQ message store is highly configurable.

Overview

By default, FUSE Message Broker uses the AMQ Message Store to persist message data. The AMQ message store is an embeddable, transactional message store that is extremely fast and reliable. It is an evolution of the Kaha system used by Active MQ 4.x. It uses a transactional journal to store message data and a Kaha-based index to store message locations for quick retrieval.

Figure 1 on page 19 shows a high-level view of the AMQ message store.

Figure 1. Overview of the AMQ Message Store



Messages are stored in file-based data logs. When all of the messages in a data log have been successfully consumed, the data log is marked as ready to be deleted. At a predetermined clean-up interval, logs marked as deletable are removed from the system.



Note

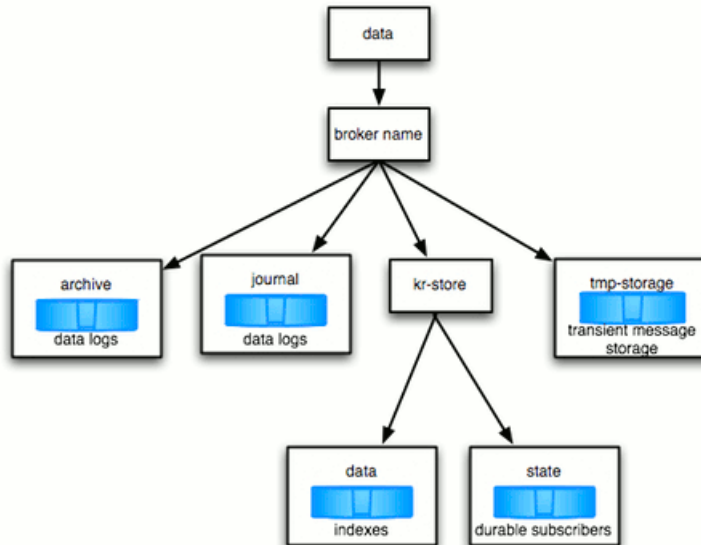
Message logs can also be archived.

An index of message locations is cached in memory to facilitate quick retrieval of message data. At configurable checkpoint intervals, the references are inserted into the persistent reference store.

Data structure

The AMQ message store is a file-based message store and uses a layered directory structure to store its data. [Figure 2 on page 20](#) shows the layout of the AMQ message store's files.

Figure 2. AMQ Message Store Directory Layout



The top-level directory of a broker's message store is identified by the name of the broker. For example a broker configured with the name `JoeFred` would have a message store folder named `JoeFred`. Beneath the message store's top-level folder are four folders:

`archive`

The `archive` folder stores archived message logs.



Note

This folder only exists when log archiving is activated.



Tip

You can change the name of this folder by setting the AMQ persistence adapter's `directoryArchive` attribute. See [Configuration on page 22](#).

`journal`

The `journal` folder stores the active message logs.

`kr-store`

The `kr-store` folder is used by the Kaha reference store when it saves message references to disk. It has two sub-folders:

`data`

The `data` folder stores the indexes for referencing the logged messages.

`state`

The `state` folder maintains state information regarding the message store. The information contained here includes the name of durable subscribers and information about transactions.



Note

This folder is only used if the AMQ persistence adapter's `persistentIndex` attribute is set to `true`. See [Configuration on page 22](#).

tmp-storage

The tmp-storage folder stores transient messages that are cached to free up memory. For example non-persistent messages may be stored here while awaiting consumption by an active, but slow consumer.

Configuration

FUSE Message Broker comes preconfigured to use the AMQ message store. However, you can modify how the message store behaves by explicitly defining its persistence adapter using the amqPersistenceAdapter element as shown in [Example 3 on page 22](#).




Example 3. Configuring the AMQ Message Store

```
<broker brokerName="broker" persistent="true" useShutdownHook="false">
  ...
  <persistenceAdapter>
    <amqPersistenceAdapter directory="activemq-data" maxFileLength="32mb"/>
  </persistenceAdapter>
</broker>
```

[Table 2 on page 22](#) describes all of the attributes that can be used to configure the AMQ message store.

Table 2. Configuration Attributes for the AMQ Message Store

Attribute	Default Value	Description
directory	activemq-data	Specifies the path to the top-level folder used to hold the message store's data files.
useNIO	true	Specifies whether or not NIO is used to write messages to the data logs.
syncOnWrite	false	Specifies whether or not to sync every write to a data log to the disk.
maxFileLength	32mb	Sets the maximum size of the logs used to store message data. Any string value provided, in kilobytes, megabytes, or gigabytes, is converted to a long value in bytes at runtime. The conversion process ignores whitespace, is case-insensitive, and accepts one- or two-letter acronyms as units of measurement. Values written without units are treated as bytes. For example, the following values are all valid: 1024kb, 256 MB, 512m, 1G, 10000000.

Attribute	Default Value	Description
<code>persistentIndex</code>	<code>true</code>	Specifies whether or not to write the reference index to a persistent store. If set this attribute is set to <code>false</code> the reference index will be maintained in non-persistent memory.
<code>checkpointInterval</code>	6000	Specifies how often, in milliseconds, that data from the journal is synchronized with the indexes.
<code>maxCheckpointMessageAddSize</code>	4096	Specifies the maximum number of messages to keep in a transaction before automatically committing them to the message store.
<code>cleanupInterval</code>	30000	Specifies the interval, in milliseconds, between clean-up sweeps of the message store.
<code>indexBinSize</code>	1024	<p>Specifies the default number of bins used by the reference index.</p> <div>  Tip Increasing the number of bins in use increases the relative performance of the index. </div>
<code>indexKeySize</code>	96	<p>Specifies the size of the index key to use.</p> <div>  Note The index key is the message id. </div>
<code>indexPageSize</code>	16kb	<p>Specifies the size of the page file to use for the reference index.</p> <div>  Tip Increasing the page file size increases the index's write performance. </div>
<code>archiveDataLogs</code>	<code>false</code>	Specifies whether or not old message logs are archived or deleted. Setting this attribute to <code>true</code> specifies that old message logs are copied to the message store's archive folder instead of being deleted.

Attribute	Default Value	Description
directoryArchive	archive	Specifies the name of the folder into which archived message logs are stored.

Failure recovery

If the message broker does not shutdown properly, the reference store indexes are cleaned and the message data files are replayed to rebuild the message store's state information. It is possible to force automatic recovery by deleting the `kr-store/state/index-store-state` file.

Using JDBC to Connect to a Database Store

FUSE Message Broker supports the use of relational databases as a message store through JDBC. You can use the JDBC persistence adapter either coupled with a high performance journal or standalone.

Basics of Using the JDBC Persistence Adapter	26
Using JDBC with the High Performance Journal	32
Using JDBC without the Journal	35

Basics of Using the JDBC Persistence Adapter

Overview

For long term persistence you may want to use a relational database as your persistent message store. FUSE Message Broker's default database when using the JDBC persistence adapter is Apache Derby. FUSE Message Broker also supports most major SQL databases. You can enable other databases by properly configuring the JDBC connection in the broker's configuration file.

You can use the JDBC persistence adapter either with or without journaling. Using the journal provides two main benefits. First, it improves the speed of the message store. Second, it provides support for JMS transactions.

Supported databases

FUSE Message Broker is known to work with the following databases:

- Apache Derby
- Axion
- DB2
- HSQL
- Informix
- MaxDB
- MySQL
- Oracle
- Postgresql
- SQLServer
- Sybase

In addition, FUSE Message Broker supports a number of generic JDBC providers.

Configuring your JDBC driver

FUSE Message Broker autodetects the JDBC driver that is in use at start-up. For the supported databases, the JDBC adapter automatically adjusts the SQL statements and JDBC driver methods to work with the driver. If you wish to customize the names of the database tables or work with an unsupported

database, you can modify both the SQL statements and the JDBC driver methods. See [Customizing the SQL statements used by the adapter on page 28](#) for information about modifying the SQL statements. See [Using generic JDBC providers on page 31](#) for information about changing the JDBC methods.

The default configuration shipped with FUSE Message Broker (*InstallDir/conf/activemq.xml*) includes configuration examples for a number of the supported databases. [Example 4 on page 27](#) shows the configuration for using the default database.

Example 4. Configuration for Using the Default Database

```
<beans ...>
  <broker xmlns="http://activemq.org/config/1.0" brokerName="localhost">
    ...
    <persistenceAdapter>
      <jdbcPersistenceAdapter dataDirectory="activemq-data"/>
    </persistenceAdapter>
    ...
  </broker>
</beans>
```

[Example 5 on page 27](#) shows the configuration for using the Oracle JDBC driver. The persistence adapter configuration refers to the Spring `bean` element that configures the JDBC driver.

Example 5. Configuration for the Oracle JDBC Driver

```
<beans ...>
  <broker xmlns="http://activemq.org/config/1.0" brokerName="localhost">
    ...
    <persistenceAdapter>
      <journalJDBC dataSource="#oracle-ds" />
    </persistenceAdapter>
    ...
  </broker>
  ...
  <bean id="oracle-ds" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">

    <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
    <property name="url" value="jdbc:oracle:thin:@localhost:1521:AMQDB"/>
    <property name="username" value="scott"/>
    <property name="password" value="tiger"/>
    <property name="poolPreparedStatements" value="true"/>
  </bean>
  ...
</beans>
```

The JDBC drivers are configured using a Spring `bean` element. The `id` attribute specifies the name by which you will refer to the driver when configuring the JDBC persistence adapter. The `class` attribute specifies the class that implements the data source used to interface with the JDBC driver. The `destroy-method` attribute specifies the name of the method to call when the JDBC driver is shutdown.

In addition to the `bean` element, the JDBC driver configuration includes a number of `property` elements. Each `property` element specifies a property required by the JDBC driver. For information about the configurable properties refer to your JDBC driver's documentation.

Customizing the SQL statements used by the adapter


You can configure the SQL statements used to create the message store. This is done by adding a `statements` element to your JDBC persistence adapters configuration. [Example 6 on page 28](#) shows a configuration snip-it that specifies that long strings are going to be stored as VARCHAR(128).

Example 6. Fine Tuning the Database Schema

```
<persistenceAdapter>
  <journaledJDBC ...>
    <statements>
      <statements stringIdDataType ="VARCHAR(128)"/>
    </statements>
  </journaledJDBC>
</persistenceAdapter>
```

The first `statements` element is a wrapper for one or more `statements` elements. Each internal `statements` element specifies a single configuration statement. [Table 3 on page 28](#) describes the configurable properties.

Table 3. Statements for Configuring the SQL Statements Used by the JDBC Persistence Adapter

Attribute	Default	Description
tablePrefix		Specifies a prefix that is added to every table name. <div> Tip The prefix should be unique per broker if multiple brokers will be sharing the same database.</div>

Attribute	Default	Description
messageTableName	ACTIVEMQ_MSGS	Specifies the name of the table in which persistent messages are stored.
durableSubAcksTableName	ACTIVEMQ_ACKS	Specifies the name of the database table used to store acknowledgment messages from durable subscribers.
lockTableName	ACTIVEMQ_LOCK	Specifies the name of the lock table used to determine the master in a master/slave scenario.
binaryDataType	BLOB	Specifies the data type used to store the messages.
containerNameDataType	VARCHAR(250)	Specifies the data type used to store the destination name.
msgIdDataType	VARCHAR(250)	Specifies the data type used to store a message id.
sequenceDataType	INTEGER	Specifies the datatype used to store the sequence id of a message.
longDataType	BIGINT	Specifies the data type used to store a Java long.
stringIdDataType	VARCHAR(250)	Specifies the data type used to store long strings like client ids, selectors, and broker names.

The properties listed in [Table 3 on page 28](#) configure the default SQL statements used by the JDBC adapter and work with all of the supported databases. If you need to override the default statements to work with an unsupported database, there are a number of other properties that can be used. These include:

- `addMessageStatement`
- `updateMessageStatement`
- `removeMessageStatement`
- `findMessageSequenceIdStatement`
- `findMessageStatement`
- `findAllMessagesStatement`
- `findLastSequenceIdInMsgsStatement`
- `findLastSequenceIdInAcksStatement`

- `createDurableSubStatement`
- `findDurableSubStatement`
- `findAllDurableSubsStatement`
- `updateLastAckOfDurableSubStatement`
- `deleteSubscriptionStatement`
- `findAllDurableSubMessagesStatement`
- `findDurableSubMessagesStatement`
- `findAllDestinationsStatement`
- `removeAllMessagesStatement`
- `removeAllSubscriptionsStatement`
- `deleteOldMessagesStatement`
- `lockCreateStatement`
- `lockUpdateStatement`
- `nextDurableSubscriberMessageStatement`
- `durableSubscriberMessageCountStatement`
- `lastAckedDurableSubscriberMessageStatement`
- `destinationMessageCountStatement`
- `findNextMessageStatement`
- `createSchemaStatements`

- `dropSchemaStatements`

Using generic JDBC providers

To use a JDBC provider not natively supported by FUSE Message Broker you can typically configure the JDBC persistence adapter to work, by setting the persistence adapter's `adapter` attribute to one of the following values:

- `org.activemq.store.jdbc.adapter.BlobJDBCAdapter`
- `org.activemq.store.jdbc.adapter.BytesJDBCAdapter`
- `org.activemq.store.jdbc.adapter.DefaultJDBCAdapter`
- `org.activemq.store.jdbc.adapter.ImageJDBCAdapter`

The different settings change how the JDBC adapter stores and accesses BLOB fields in the database. To determine the proper setting consult the documentation for your JDBC driver and your database.

[Example 7 on page 31](#) shows a configuration snippet configuring the journaled JDBC persistence adapter to use the blob JDBC adapter.

Example 7. Configuring a Generic JDBC Provider

```
<broker persistent="true" ...>
  ...
  <persistenceAdapter>
    <journaledJDBC adapter="org.activemq.store.jdbc.adapter.BlobJDBCAdapter" ... />
  </persistenceAdapter>
  ...
</broker>
```

Using JDBC with the High Performance Journal

Overview

Using the JDBC persistence adapter with FUSE Message Broker's high performance journal boosts the performance of the persistence adapter in two ways:

1. In applications where message consumers keep up with the message producers, the journal makes it possible to lower the number of messages that need to be committed to the data base. For example a message producer could publish 10,000 messages between journal checkpoints. If the message consumer pops 9,900 messages off of the queue during the same interval, only 100 messages will be committed to the database through the JDBC adapter.
2. In applications where the message consumers cannot keep up with the message producers, or in applications where messages must persist for long periods, the journal boosts performance by committing messages in large batches. This means that the JDBC driver can optimize the writes to the external database.

In addition to the performance gains, the high performance journal also makes it possible to ensure the consistency of JMS transactions in the case of a system failure.

Configuration

To configure FUSE Message Broker to use the JDBC persistence adapter with the high performance journal you add the `journalJDBC` element to the `persistenceAdapter` element in your broker's configuration as shown in [Example 8 on page 33](#).

[Table 4 on page 32](#) describes the attributes used to configure the journaled JDBC persistence adapter.

Table 4. Attributes for Configuring the Journaled JDBC Persistence Adapter

Attribute	Default Value	Description
<code>adapter</code>		Specifies the strategy to use when accessing a non-supported database. For more information see Using generic JDBC providers on page 31 .
<code>createTablesOnStartup</code>	<code>true</code>	Specifies whether or not new database tables are created when the broker starts. If the database tables already exist, the existing tables are reused.

Attribute	Default Value	Description
dataDirectory	activemq-data	Specifies the directory into which the default Derby database writes its files.
dataSource	#derby	Specifies the id of the Spring bean storing the JDBC driver's configuration. For more information see Configuring your JDBC driver on page 26 .
journalArchiveDirectory		Specifies the directory used to store archived journal log files.
journalLogFiles	2	Specifies the number of log files to use for storing the journal.
journalLogFileSize	20MB	Specifies the size for a journal's log file.
journalThreadPriority	10	Specifies the thread priority of the thread used for journaling.
useDatabaseLock	true	Specifies whether or not an exclusive database lock should be used to enable JDBC Master/Slave.
useJournal	true	Specifies whether or not to use the journal.

Example

[Example 8 on page 33](#) shows a configuration snippet that configures the journaled JDBC adapter to use a MySQL database.

Example 8. Configuring FUSE Message Broker to use the Journaled JDBC Persistence Adapter

```

<beans ...>
  <broker ...>
    ...
    ❶ <persistenceAdapter>
    ❷   <journaledJDBC journalLogFiles="5" dataSource="#mysql-ds" />
    </persistenceAdapter>
    ...
  </broker>
  ...
  ❸<bean id="mysql-ds" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">

    <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
    <property name="url" value="jdbc:mysql://localhost/activemq?relaxAutoCommit=true"/>
    <property name="username" value="activemq"/>
    <property name="password" value="activemq"/>
    <property name="poolPreparedStatements" value="true"/>
  </bean>

```

The configuration in [Example 8 on page 33](#) show three noteworthy elements:

- ❶ The `persistenceAdapter` element wraps the configuration for the JDBC persistence adapter.
- ❷ The `journaledJDBC` element specifies that the broker will use the JDBC persistence adapter with the high performance journal. The element's attributes configure the following properties:
 - The journal will span five log files.
 - Quick journaling will be used. Therefore only message references will be written to the JDBC database.
 - The configuration for the JDBC driver is specified in a `bean` element with the id `mysql-ds`.
- ❸ The `bean` element specified the configuration for the MySQL JDBC driver.

Using JDBC without the Journal

Overview

For instances when journaling is not appropriate, or you wish to use your own journaling system, you can use the JDBC persistence adapter without the FUSE Message Broker high performance journal.

Configuration

To configure FUSE Message Broker to use the JDBC persistence adapter without the high performance journal you add the `jdbcpersistenceAdapter` element to the `persistenceAdapter` element in your broker's configuration as shown in [Example 9 on page 36](#).

[Table 5 on page 35](#) describes the attributes used to configure the journaled JDBC persistence adapter.

Table 5. Attributes for Configuring the Plain JDBC Persistence Adapter

Attribute	Default Value	Description
<code>adapter</code>		Specifies the strategy to use when accessing a non-supported database. For more information see Using generic JDBC providers on page 31 .
<code>cleanupPeriod</code>	<code>300000</code>	Specifies, in milliseconds, the interval at which acknowledged messages are deleted.
<code>createTablesOnStartup</code>	<code>true</code>	Specifies whether or not new database tables are created when the broker starts. If the database tables already exist, the existing tables are reused.
<code>dataDirectory</code>	<code>activemq-data</code>	Specifies the directory into which the default Derby database writes its files.
<code>dataSource</code>	<code>#derby</code>	Specifies the id of the Spring bean storing the JDBC driver's configuration. For more information see Configuring your JDBC driver on page 26 .
<code>useDatabaseLock</code>	<code>true</code>	Specifies whether or not an exclusive database lock should be used to enable JDBC Master/Slave.

Example

[Example 9 on page 36](#) shows a configuration snippet that configures the JDBC adapter to use the default database.

Example 9. Configuring FUSE Message Broker to use the Plain JDBC Persistence Adapter

```
<beans ...>
  <broker ...>
    ...
    ❶ <persistenceAdapter>
    ❷   <journalJDBC dataSource="derby-ds" />
    </persistenceAdapter>
    ...
  </broker>
  ...
  ❸<bean id="#derby-ds" class="org.apache.derby.jdbc.EmbeddedDataSource">
    <property name="databaseName" value="derbydb"/>
    <property name="createDatabase" value="create"/>
  </bean>
```

The configuration in [Example 9 on page 36](#) show three noteworthy elements:

- ❶ The `persistenceAdapter` element wraps the configuration for the JDBC persistence adapter.
- ❷ The `jdbcPersistenceAdapter` element specifies that the broker will use the plain JDBC persistence adapter and that the JDBC driver's configuration is specified in a `bean` element with the id `derby-ds`.
- ❸ The `bean` element specified the configuration for the Derby JDBC driver.

Message Cursors

FUSE Message Broker uses message cursors to improve the scalability of the persistent message store. By default, a hybrid approach that uses an in memory dispatch queue for fast consumers and message cursors for slower consumers is used. FUSE Message Broker also supports two alternative cursor implementations. The type of cursor can be configured on a per-destination basis.

Types of Cursors	38
Configuring the Type of Cursor Used by a Destination	42

Message cursors provide a means for optimizing a persistent message store. They allow the persistent store to maintain a pointer to the next batch of messages to pull from the persistent message store. FUSE Message Broker has three types of cursors that can be used depending on the needs of your application:

- [Store-based](#) cursors are the default cursor implementation. They offer the best all around performance.
- [VM](#) cursors are very fast, but cannot handle slow message consumers.
- [File-based](#) cursors are useful when the message store is slow and message consumers are relatively fast.

Types of Cursors

Store-based cursors

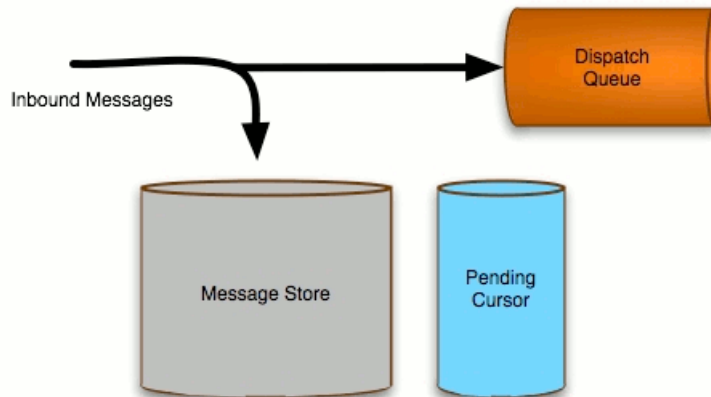
Store-based cursors are the default cursor implementation used by FUSE Message Broker. Store-based cursors are a hybrid implementation that offers the robustness of typical cursor implementations and the speed of in-memory message reference implementations.

Typically messaging systems will pull persistent messages from long-term storage in a batch when a client is ready to consume them. A cursor will be used to maintain the position for the next batch of messages. While this approach scales well and provides excellent robustness, it does not perform well when message consumers keep pace with message producers.

As shown in [Figure 3 on page 38](#), store-based cursors address the fast consumer case by skipping the message cursor. When a message consumer is keeping pace with the message producers, persistent messages are written to the message store and moved directly into a dispatch queue for the consumer.

Figure 3. Store-based Cursors for a Fast Consumer

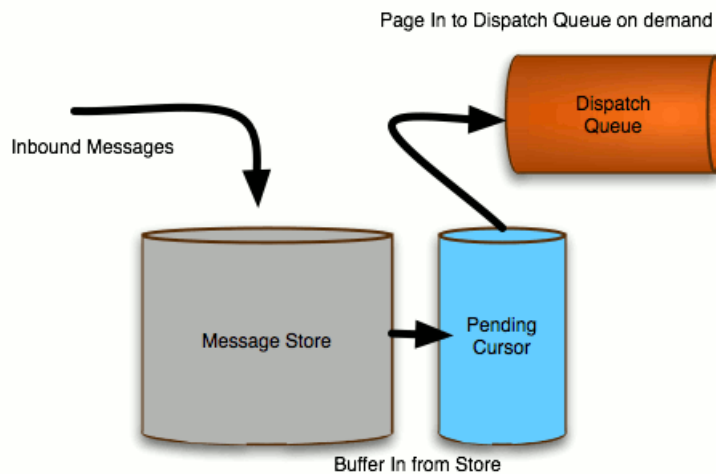
Dispatching Messages for Fast Consumers



When a consumer starts with a back log of messages or falls behind its message producers, FUSE Message Broker changes the strategy used to dispatch messages. As shown in [Figure 4 on page 39](#), messages are held in the message store and fed into the consumer's dispatch queue using the pending cursor.

Figure 4. Store-based Cursors for a Slow Consumer

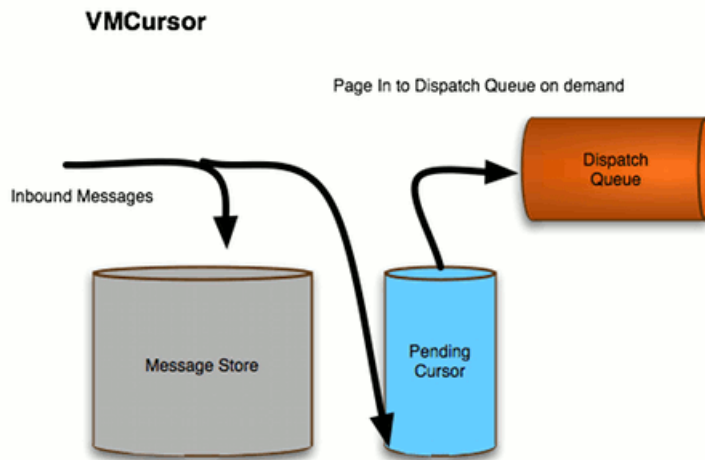
Dispatching Messages if Dispatch Queue is Full



VM cursors

When speed is the top priority and the consumers can definitely keep pace with the message producers, VM cursors could be the best approach. In this approach, shown in [Figure 5 on page 40](#), messages are written to the persistent store and then also stored in the pending cursor which is held completely in memory. The messages are fed into the dispatch queue from the pending cursor.

Figure 5. VM Cursors

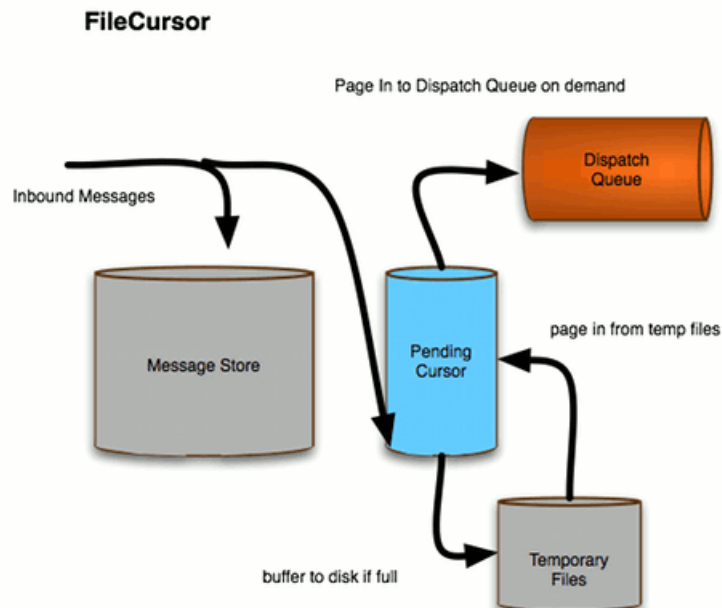


Because the messages are dispatched from active memory when using VM cursors, this method is exceptionally fast. However, if the number of unconsumed messages gets large the producers will be throttled to avoid exceeding the available memory.

File-based cursors

File-based cursors are a variation of VM cursors that provides a buffer against running out of memory when a consumer falls behind. As shown in [Figure 6 on page 41](#), the broker pages messages out to a temporary file when the broker's memory limit is reached.

Figure 6. File-based Cursors



Using a temporary file cushions the broker against situations where a consumer occasionally falls behind or messages are produced in a burst. The broker uses the temporary file instead of resorting to using slower persistent storage.

File-based cursors do not scale well when consumers are frequently behind by a large margin. It is also not ideal when a fast long term message store is available.

Configuring the Type of Cursor Used by a Destination

Overview

By default, FUSE Message Broker uses store-based cursors. You can, however, configure your destinations to use one of the alternative cursor implementations by adding the appropriate policy entries into the destination's policy map.

You configure a destination's policy set using a `destinationPolicy` element. The `destinationPolicy` element is a wrapper for a `policyMap` element. The `policyMap` element is a wrapper for a `policyEntries` element. The `policyEntries` element is a wrapper for one or more `policyEntry` elements.

The cursor policies are entered as children to a `policyEntry` element. The configuration elements used to specify the type of destination you are configuring. Topics use cursors for both durable subscribers and transient subscribers, so it uses two sets of configuration elements. Queues only a single cursor and only require a single set of configuration elements.

Configuring topics

Topics maintain a dispatch queue and a pending cursor for every consumer subscribed to the topic regardless of whether the subscription is durable or transient. You can configure the cursor implementation used by durable subscribers separately from the cursor implementation used by transient subscribers.



Important

If you want to use the store-based cursor implementation, you do not add any extra elements to the configuration. FUSE Message Broker uses store-based cursors by default.

You configure the cursor implementation used by durable subscribers by adding `PendingDurableSubscriberMessageStoragePolicy` child element to the topic's `policyEntry` element. [Table 6 on page 42](#) describes the possible children of `PendingDurableSubscriberMessageStoragePolicy`.

Table 6. Elements for Configuring the Type of Cursor to Use for Durable Subscribers

Element	Description
<code>vmDurableCursor</code>	Specifies the VM cursors will be used. See VM cursors on page 39 for more information.

Element	Description
<code>fileDurableSubscriberCursor</code>	Specifies that file-based cursors will be used. See File-based cursors on page 40 for more information.

You configure the cursor implementation used by transient subscribers by adding `pendingSubscriberPolicy` child element to the topic's `policyEntry` element. [Table 7 on page 43](#) describes the possible children of `pendingSubscriberPolicy`.

Table 7. Elements for Configuring the Type of Cursor to Use for Transient Subscribers

Element	Description
<code>vmCursor</code>	Specifies the VM cursors will be used. See VM cursors on page 39 for more information.
<code>fileCursor</code>	Specifies that file-based cursors will be used. See File-based cursors on page 40 for more information.

[Example 10 on page 43](#) shows a configuration snippet that configures a topic to use VM cursors for its transient subscribers and file-based cursors for its durable subscribers.

Example 10. Configuring a Topic's Cursor Usage

```
<beans ... >
  <broker ...>
    ...
    <destinationPolicy>
      <policyMap>
        <policyEntries>
          <policyEntry topic="com.ionas.">
            ...
            <pendingSubscriberPolicy>
              <vmCursor />
            </pendingSubscriberPolicy>
            <PendingDurableSubscriberMessageStoragePolicy>
              <fileDurableSubscriberPolicy />
            </PendingDurableSubscriberMessageStoragePolicy>
            ...
          </policyEntry>
          ...
        </policyEntries>
      </policyMap>
    </destinationPolicy>
    ...
  </broker>
```

```
...
</beans>
```

Configuring queues

Queues use a single pending cursor and dispatch queue. You configure the type of cursor to use by adding a `pendingQueuePolicy` element to the queue's `policyEntry` element. [Table 8 on page 44](#) describes the possible children elements of the `pendingQueuePolicy` element.

Table 8. Elements for Configuring the Type of Cursor to Use for a Queue

Element	Description
<code>vmQueueCursor</code>	Specifies the VM cursors will be used. See VM cursors on page 39 for more information.
<code>fileQueueCursor</code>	Specifies that file-based cursors will be used. See File-based cursors on page 40 for more information.

[Example 11 on page 44](#) shows a configuration snip-it that configures a queue to use VM cursors.

Example 11. Configuring a Queue's Cursor Usage

```
<beans ... >
  <broker ...>
    ...
    <destinationPolicy>
      <policyMap>
        <policyEntries>
          <policyEntry queue="com.iona.">
            ...
            <pendingQueuePolicy>
              <vmQueueCursor />
            </pendingQueuePolicy>
            ...
          </policyEntry>
          ...
        </policyEntries>
      </policyMap>
    </destinationPolicy>
    ...
  </broker>
  ...
</beans>
```

Index

A

- AMQ message store
 - archive folder, 21
 - journal folder, 21
 - kr-store folder, 21
 - tmp-storage folder, 22
- amqpPersistenceAdapter, 22
 - archiveDataLogs attribute, 23
 - checkpointInterval attribute, 23
 - cleanupInterval attribute, 23
 - directory attribute, 22
 - directoryArchive attribute, 24
 - indexBinSize attribute, 23
 - indexKeySize attribute, 23
 - indexPageSize attribute, 23
 - maxCheckpointMessageAddSize attribute, 23
 - maxFileLength attribute, 22
 - syncOnWrite attribute, 22
 - useNIO attribute, 22
- amqpPersistenceAdapter
 - persistentIndex attribute, 23

B

- broker element, 17
 - persistent attribute, 17

C

- configuration
 - turning persistence on/off, 17
- cursors
 - file-based, 40
 - store-based, 38
 - VM, 39

D

- destinationPolicy, 42
- durable subscribers
 - configuring cursors, 42

- using file-based cursors, 43
- using VM cursors, 42

F

- fileCursor, 43
- fileDurableSubscriberCursor, 43
- fileQueueCursor, 44

J

- JDBC
 - using generic providers, 31
- jdbcPersistenceAdapter, 35
 - adapter attribute, 31, 35
 - cleanupPeriod attribute, 35
 - createTablesOnStartup attribute, 35
 - dataDirectory attribute, 35
 - dataSource attribute, 35
 - useDatabaseLock attribute, 35
- journalJDBC, 32
 - adapter attribute, 31, 32
 - createTablesOnStartup attribute, 32
 - dataDirectory attribute, 33
 - dataSource attribute, 33
 - journalArchiveDirectory attribute, 33
 - journalLogFiles attribute, 33
 - journalLogFileSize attribute, 33
 - journalThreadPriority attribute, 33
 - useDatabaseLock attribute, 33
 - useJournal attribute, 33

P

- PendingDurableSubscriberMessageStoragePolicy, 42
- pendingQueuePolicy, 44
- pendingSubscriberPolicy, 43
- persistenceAdapter, 18, 22
- policyEntries, 42
- policyEntry, 42
- policyMap, 42

R

- reference store, 21

S

SQL data types, 28

statements, 28

- binaryDataType, 29

- containerNameDataType attribute, 29

- durableSubAcksTableName attribute, 29

- lockTableName attribute, 29

- longDataType attribute, 29

- messageTableName attribute, 29

- msgIdDataType attribute, 29

- sequenceDataType attribute, 29

- stringIdDataType attribute, 29

- tablePrefix attribute, 28

T

transient subscribers

- configuring cursors, 43

- using file-based cursors, 43

- using VM cursors, 43

V

vmCursor, 43

vmDurableCursor, 42

vmQueueCursor, 44