



Red Hat AMQ 7.7

Using the AMQ OpenWire JMS Client

For Use with AMQ Clients 2.7

Red Hat AMQ 7.7 Using the AMQ OpenWire JMS Client

For Use with AMQ Clients 2.7

Legal Notice

Copyright © 2020 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This guide describes how to install and configure the client, run hands-on examples, and use your client with other AMQ components.

Table of Contents

CHAPTER 1. OVERVIEW	4
1.1. KEY FEATURES	4
1.2. SUPPORTED STANDARDS AND PROTOCOLS	4
1.3. SUPPORTED CONFIGURATIONS	4
1.4. TERMS AND CONCEPTS	5
1.5. DOCUMENT CONVENTIONS	6
The sudo command	6
File paths	6
Variable text	6
CHAPTER 2. INSTALLATION	7
2.1. PREREQUISITES	7
2.2. USING THE RED HAT MAVEN REPOSITORY	7
2.3. INSTALLING A LOCAL MAVEN REPOSITORY	7
2.4. INSTALLING THE EXAMPLES	8
CHAPTER 3. GETTING STARTED	9
3.1. PREREQUISITES	9
3.2. RUNNING YOUR FIRST EXAMPLE	9
CHAPTER 4. CONFIGURATION	10
4.1. CONFIGURING THE INITIAL CONTEXT FACTORY	10
Using a jndi.properties file	10
Using a system property	10
4.2. CONFIGURING THE CONNECTION FACTORY	10
4.3. CONNECTION URIS	11
4.4. JMS OPTIONS	11
Prefetch policy options	12
Redelivery policy options	12
4.5. TCP OPTIONS	13
4.6. FAILOVER OPTIONS	13
4.7. SSL/TLS OPTIONS	13
4.8. LARGE MESSAGE OPTIONS	14
4.9. CONFIGURING JNDI RESOURCES	14
4.9.1. Configuring queue and topic names	14
4.9.2. Setting JNDI properties programatically	15
CHAPTER 5. MESSAGE DELIVERY	16
5.1. WRITING TO A STREAMED LARGE MESSAGE	16
5.2. READING FROM A STREAMED LARGE MESSAGE	16
APPENDIX A. USING YOUR SUBSCRIPTION	17
A.1. ACCESSING YOUR ACCOUNT	17
A.2. ACTIVATING A SUBSCRIPTION	17
A.3. DOWNLOADING RELEASE FILES	17
A.4. REGISTERING YOUR SYSTEM FOR PACKAGES	17
APPENDIX B. USING RED HAT MAVEN REPOSITORIES	19
B.1. USING THE ONLINE REPOSITORY	19
Adding the repository to your Maven settings	19
Adding the repository to your POM file	20
B.2. USING A LOCAL REPOSITORY	20

APPENDIX C. USING AMQ BROKER WITH THE EXAMPLES	22
C.1. INSTALLING THE BROKER	22
C.2. STARTING THE BROKER	22
C.3. CREATING A QUEUE	22
C.4. STOPPING THE BROKER	22

CHAPTER 1. OVERVIEW

AMQ OpenWire JMS is a Java Message Service (JMS) 1.1 client for use in messaging applications that send and receive OpenWire messages.

AMQ OpenWire JMS is part of AMQ Clients, a suite of messaging libraries supporting multiple languages and platforms. For an overview of the clients, see [AMQ Clients Overview](#). For information about this release, see [AMQ Clients 2.7 Release Notes](#).

AMQ OpenWire JMS is based on the JMS implementation from [Apache ActiveMQ](#). For more information about the JMS API, see the [JMS API reference](#) and the [JMS tutorial](#).

1.1. KEY FEATURES

- JMS 1.1 compatible
- SSL/TLS for secure communication
- Automatic reconnect and failover
- Distributed transactions (XA)
- Pure-Java implementation

1.2. SUPPORTED STANDARDS AND PROTOCOLS

AMQ OpenWire JMS supports the following industry-recognized standards and network protocols:

- Version 1.1 of the [Java Message Service](#) API.
- Modern [TCP](#) with [IPv6](#)

1.3. SUPPORTED CONFIGURATIONS

AMQ OpenWire JMS supports the OS and language versions listed below. For more information, see [Red Hat AMQ 7 Supported Configurations](#).

- Red Hat Enterprise Linux 7 and 8 with the following JDKs:
 - OpenJDK 8 and 11
 - Oracle JDK 8
 - IBM JDK 8
- Red Hat Enterprise Linux 6 with the following JDKs:
 - OpenJDK 8
 - Oracle JDK 8
- IBM AIX 7.1 with IBM JDK 8
- Microsoft Windows 10 Pro with Oracle JDK 8
- Microsoft Windows Server 2012 R2 and 2016 with Oracle JDK 8

- Oracle Solaris 10 and 11 with Oracle JDK 8

AMQ OpenWire JMS is supported in combination with the following AMQ components and versions:

- The latest version of AMQ Broker
- A-MQ 6 versions 6.2.1 and newer

1.4. TERMS AND CONCEPTS

This section introduces the core API entities and describes how they operate together.

Table 1.1. API terms

Entity	Description
ConnectionFactory	An entry point for creating connections.
Connection	A channel for communication between two peers on a network. It contains sessions.
Session	A context for producing and consuming messages. It contains message producers and consumers.
MessageProducer	A channel for sending messages to a destination. It has a target destination.
MessageConsumer	A channel for receiving messages from a destination. It has a source destination.
Destination	A named location for messages, either a queue or a topic.
Queue	A stored sequence of messages.
Topic	A stored sequence of messages for multicast distribution.
Message	An application-specific piece of information.

AMQ OpenWire JMS sends and receives *messages*. Messages are transferred between connected peers using *message producers* and *consumers*. Producers and consumers are established over *sessions*. Sessions are established over *connections*. Connections are created by *connection factories*.

A sending peer creates a producer to send messages. The producer has a *destination* that identifies a target queue or topic at the remote peer. A receiving peer creates a consumer to receive messages. Like the producer, the consumer has a destination that identifies a source queue or topic at the remote peer.

A destination is either a *queue* or a *topic*. In JMS, queues and topics are client-side representations of named broker entities that hold messages.

A queue implements point-to-point semantics. Each message is seen by only one consumer, and the message is removed from the queue after it is read. A topic implements publish-subscribe semantics. Each message is seen by multiple consumers, and the message remains available to other consumers after it is read.

See the [JMS tutorial](#) for more information.

1.5. DOCUMENT CONVENTIONS

The `sudo` command

In this document, **sudo** is used for any command that requires root privileges. Exercise caution when using **sudo** because any changes can affect the entire system. For more information about **sudo**, see [Using the sudo command](#).

File paths

In this document, all file paths are valid for Linux, UNIX, and similar operating systems (for example, **/home/andrea**). On Microsoft Windows, you must use the equivalent Windows paths (for example, **C:\Users\andrea**).

Variable text

This document contains code blocks with variables that you must replace with values specific to your environment. Variable text is enclosed in arrow braces and styled as italic monospace. For example, in the following command, replace **<project-dir>** with the value for your environment:

```
$ cd <project-dir>
```

CHAPTER 2. INSTALLATION

This chapter guides you through the steps to install AMQ OpenWire JMS in your environment.

2.1. PREREQUISITES

- You must have a [subscription](#) to access AMQ release files and repositories.
- To build programs with AMQ OpenWire JMS, you must install [Apache Maven](#).
- To use AMQ OpenWire JMS, you must install Java.

2.2. USING THE RED HAT MAVEN REPOSITORY

Configure your Maven environment to download the client library from the Red Hat Maven repository.

Procedure

1. Add the Red Hat repository to your Maven settings or POM file. For example configuration files, see [Section B.1, "Using the online repository"](#).

```
<repository>
  <id>red-hat-ga</id>
  <url>https://maven.repository.redhat.com/ga</url>
</repository>
```

2. Add the library dependency to your POM file.

```
<dependency>
  <groupId>org.apache.activemq</groupId>
  <artifactId>activemq-client</artifactId>
  <version>5.11.0.redhat-630416</version>
</dependency>
```

The client is now available in your Maven project.

2.3. INSTALLING A LOCAL MAVEN REPOSITORY

As an alternative to the online repository, AMQ OpenWire JMS can be installed to your local filesystem as a file-based Maven repository.

Procedure

1. [Use your subscription](#) to download the **AMQ Broker 7.7.0 Maven repository.zip** file.
2. Extract the file contents into a directory of your choosing.
On Linux or UNIX, use the **unzip** command to extract the file contents.

```
$ unzip amq-broker-7.7.0-maven-repository.zip
```

On Windows, right-click the .zip file and select **Extract All**.

3. Configure Maven to use the repository in the **maven-repository** directory inside the extracted install directory. For more information, see [Section B.2, "Using a local repository"](#).

2.4. INSTALLING THE EXAMPLES

Procedure

1. [Use your subscription](#) to download the **AMQ Broker 7.7.0** .zip file.
2. Extract the file contents into a directory of your choosing.
On Linux or UNIX, use the **unzip** command to extract the file contents.

```
$ unzip amq-broker-7.7.0.zip
```

On Windows, right-click the .zip file and select **Extract All**.

When you extract the contents of the .zip file, a directory named **amq-broker-7.7.0** is created. This is the top-level directory of the installation and is referred to as **<install-dir>** throughout this document.

CHAPTER 3. GETTING STARTED

This chapter guides you through the steps to set up your environment and run a simple messaging program.

3.1. PREREQUISITES

- To build the example, Maven must be configured to use the [Red Hat repository](#) or a [local repository](#).
- You must [install the examples](#).
- You must have a message broker listening for connections on **localhost**. It must have anonymous access enabled. For more information, see [Starting the broker](#).
- You must have a queue named **exampleQueue**. For more information, see [Creating a queue](#).

3.2. RUNNING YOUR FIRST EXAMPLE

The example creates a consumer and producer for a queue named **exampleQueue**. It sends a text message and then receives it back, printing the received message to the console.

Procedure

1. Use Maven to build the examples by running the following command in the **<install-dir>/examples/protocols/openwire/queue** directory.

```
$ mvn clean package dependency:copy-dependencies -DincludeScope=runtime -DskipTests
```

The addition of **dependency:copy-dependencies** results in the dependencies being copied into the **target/dependency** directory.

2. Use the **java** command to run the example.
On Linux or UNIX:

```
$ java -cp "target/classes:target/dependency/*"  
org.apache.activemq.artemis.jms.example.QueueExample
```

On Windows:

```
> java -cp "target\classes;target\dependency\*"  
org.apache.activemq.artemis.jms.example.QueueExample
```

Running it on Linux results in the following output:

```
$ java -cp "target/classes:target/dependency/*"  
org.apache.activemq.artemis.jms.example.QueueExample  
Sent message: This is a text message  
Received message: This is a text message
```

The source code for the example is in the **<install-dir>/examples/protocols/openwire/queue/src** directory. Additional examples are available in the **<install-dir>/examples/protocols/openwire** directory.

CHAPTER 4. CONFIGURATION

This chapter describes the process for binding the AMQ OpenWire JMS implementation to your JMS application and setting configuration options.

JMS uses the Java Naming Directory Interface (JNDI) to register and look up API implementations and other resources. This enables you to write code to the JMS API without tying it to a particular implementation.

Configuration options are exposed as query parameters on the connection URI. Some of the options are also exposed as corresponding **set** and **get** methods on the **ConnectionFactory** implementation object.

For more information about configuring AMQ OpenWire JMS, see the [ActiveMQ user guide](#).

4.1. CONFIGURING THE INITIAL CONTEXT FACTORY

JMS applications use a JNDI **InitialContext** object obtained from an **InitialContextFactory** to look up JMS objects such as the connection factory. AMQ OpenWire JMS provides an implementation of the **InitialContextFactory** in the **org.apache.activemq.jndi.ActiveMQInitialContextFactory** class.

The **InitialContextFactory** implementation is discovered when the **InitialContext** object is instantiated:

```
javax.naming.Context context = new javax.naming.InitialContext();
```

To find an implementation, JNDI must be configured in your environment. There are two main ways of achieving this, using a **jndi.properties** file or using a system property.

Using a jndi.properties file

Create a file named **jndi.properties** and place it on the Java classpath. Add a property with the key **java.naming.factory.initial**.

Example: Setting the JNDI initial context factory using a jndi.properties file

```
java.naming.factory.initial = org.apache.activemq.jndi.ActiveMQInitialContextFactory
```

In Maven-based projects, the **jndi.properties** file is placed in the **<project-dir>/src/main/resources** directory.

Using a system property

Set the **java.naming.factory.initial** system property.

Example: Setting the JNDI initial context factory using a system property

```
$ java -Djava.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory ...
```

4.2. CONFIGURING THE CONNECTION FACTORY

The JMS connection factory is the entry point for creating connections. It uses a connection URI that encodes your application-specific configuration settings.

To set the factory name and connection URI, create a property in the format below. You can store this configuration in a **jndi.properties** file or set the corresponding system property.

The JNDI property format for connection factories

```
connectionFactory.<factory-name> = <connection-uri>
```

For example, this is how you might configure a factory named **app1**:

Example: Setting the connection factory in a `jndi.properties` file

```
connectionFactory.app1 = tcp://example.net:61616?jms.clientID=backend
```

You can then use the JNDI context to look up your configured connection factory using the name **app1**:

```
ConnectionFactory factory = (ConnectionFactory) context.lookup("app1");
```

4.3. CONNECTION URIS

A connection factory is configured using a connection URI in the following format:

The Connection URI format

```
<scheme>://<host>:<port>[?<option>=<value>[&<option>=<value>...]]
```

The scheme is **tcp** for unencrypted connections and **ssl** for SSL/TLS connections.

For example, the following is a connection URI that connects to host **example.net** at port **61616** and sets the client ID to **backend**:

Example: A connection URI

```
tcp://example.net:61616?jms.clientID=backend
```

Failover URIs take the following form:

The failover URI format

```
failover:(<connection-uri>[,<connection-uri>])[?<option>=<value>[&<option>=<value>...]]
```

Options prefixed with **nested.** are applied to each connection URI in the list.

The available connection options are described in the following sections.

4.4. JMS OPTIONS

jms.username

The user name used to authenticate the connection.

jms.password

The password used to authenticate the connection.

jms.clientID

The client ID that is applied to the connection.

jms.closeTimeout

The close timeout in milliseconds. The default is 15000 (15 seconds).

jms.checkForDuplicates

If enabled, ignore duplicate messages. It is enabled by default.

jms.disableTimeStampsByDefault

If enabled, do not timestamp messages. It is disabled by default.

jms.useAsyncSend

If enabled, send messages without waiting for acknowledgment. It is disabled by default.

jms.alwaysSyncSend

If enabled, send waits for acknowledgment in all delivery modes. It is disabled by default.

jms.useCompression

If enabled, compress message bodies. It is disabled by default.

jms.useRetroactiveConsumer

If enabled, non-durable subscribers can receive messages that were published before the subscription started. It is disabled by default.

Prefetch policy options

Prefetch policy determines how many messages each **MessageConsumer** will fetch from the remote peer and hold in a local "prefetch" buffer.

jms.prefetchPolicy.queuePrefetch

The number of messages to prefetch for queues. The default is 1000.

jms.prefetchPolicy.queueBrowserPrefetch

The number of messages to prefetch for queue browsers. The default is 500.

jms.prefetchPolicy.topicPrefetch

The number of messages to prefetch for non-durable topics. The default is 32766.

jms.prefetchPolicy.durableTopicPrefetch

The number of messages to prefetch for durable topics. The default is 100.

jms.prefetchPolicy.all

This can be used to set all prefetch values at once.

The value of prefetch can affect the distribution of messages to multiple consumers on a queue or shared subscription. A higher value can result in larger batches sent at once to each consumer. To achieve more even round-robin distribution, use a lower value.

Redelivery policy options

Redelivery policy controls how redelivered messages are handled on the client.

jms.redeliveryPolicy.maximumRedeliveries

The number of times redelivery is attempted before the message is sent to the dead letter queue. The default is 6. -1 means no limit.

jms.redeliveryPolicy.redeliveryDelay

The time in milliseconds between redelivery attempts. This is used if **initialRedeliveryDelay** is 0. The default is 1000 (1 second).

jms.redeliveryPolicy.initialRedeliveryDelay

The time in milliseconds before the first redelivery attempt. The default is 1000 (1 second).

jms.redeliveryPolicy.maximumRedeliveryDelay

The maximum time in milliseconds between redelivery attempts. This is used if **useExponentialBackOff** is enabled. The default is 1000 (1 second). -1 means no limit.

jms.redeliveryPolicy.useExponentialBackOff

If enabled, increase redelivery delay with each subsequent attempt. It is disabled by default.

jms.redeliveryPolicy.backOffMultiplier

The multiplier for increasing the redelivery delay. The default is 5.

jms.redeliveryPolicy.useCollisionAvoidance

If enabled, adjust the redelivery delay slightly up or down to avoid collisions. It is disabled by default.

jms.redeliveryPolicy.collisionAvoidanceFactor

The multiplier for adjusting the redelivery delay. The default is 0.15.

4.5. TCP OPTIONS

connectionTimeout

The connection timeout in milliseconds. The default is 30000 (30 seconds). 0 means no timeout.

ioBufferSize

The I/O buffer size in bytes. The default is 8192 (8 KiB).

useKeepAlive

If enabled, periodically send data to keep the connection alive. It is enabled by default.

soTimeout

The socket read timeout. The default is 0, meaning no timeout.

soWriteTimeout

The socket write timeout. The default is 0, meaning no timeout.

tcpNoDelay

If enabled, do not delay and buffer TCP sends. It is disabled by default.

4.6. FAILOVER OPTIONS

maxReconnectAttempts

The number of reconnect attempts allowed before reporting the connection as failed. The default is -1, meaning no limit. 0 disables reconnect.

maxReconnectDelay

The maximum time in milliseconds between the second and subsequent reconnect attempts. The default is 30000 (30 seconds).

randomize

If enabled, randomly select one of the failover endpoints. It is enabled by default.

reconnectDelayExponent

The multiplier for increasing the reconnect delay backoff. The default is 2.0.

useExponentialBackOff

If enabled, increase the reconnect delay with each subsequent attempt. It is enabled by default.

4.7. SSL/TLS OPTIONS

socket.keyStore

The path to the SSL/TLS key store. A key store is required for mutual SSL/TLS authentication. If unset, the value of the **javax.net.ssl.keyStore** system property is used.

socket.keyStorePassword

The password for the SSL/TLS key store. If unset, the value of the **javax.net.ssl.keyStorePassword** system property is used.

socket.trustStore

The path to the SSL/TLS trust store. If unset, the value of the **javax.net.ssl.trustStore** system property is used.

socket.trustStorePassword

The password for the SSL/TLS trust store. If unset, the value of the **javax.net.ssl.trustStorePassword** system property is used.

socket.enabledCipherSuites

A comma-separated list of cipher suites to enable. If unset, the JVM default ciphers are used.

socket.enabledProtocols

A comma-separated list of SSL/TLS protocols to enable. If unset, the JVM default protocols are used.

4.8. LARGE MESSAGE OPTIONS

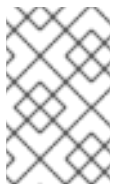
The client can enable large message support by setting a value for the property **wireFormat.minLargeMessageSize**. Any message larger than **wireFormat.minLargeMessageSize** is considered a large message.

wireFormat.minLargeMessageSize

The minimum size in bytes at which a message is treated as a large message. The default is 102400 (100 KiB).

wireFormat.compressLargeMessages

If enabled, compress large messages, as defined by **wireFormat.minLargeMessageSize**. It is disabled by default.

**NOTE**

If the compressed size of a large message is less than the value of **wireFormat.minLargeMessageSize**, the message is sent as a regular message. Therefore, it is not written to the broker's large-message data directory.

4.9. CONFIGURING JNDI RESOURCES

4.9.1. Configuring queue and topic names

JMS provides the option of using JNDI to look up deployment-specific queue and topic resources.

To set queue and topic names in JNDI, create properties in the following format. Either place this configuration in a **jndi.properties** file or define corresponding system properties.

The JNDI property format for queues and topics

■

```
queue.<queue-lookup-name> = <queue-name>  
topic.<topic-lookup-name> = <topic-name>
```

For example, the following properties define the names **jobs** and **notifications** for two deployment-specific resources:

Example: Setting queue and topic names in a `jndi.properties` file

```
queue.jobs = app1/work-items  
topic.notifications = app1/updates
```

You can then look up the resources by their JNDI names:

```
Queue queue = (Queue) context.lookup("jobs");  
Topic topic = (Topic) context.lookup("notifications");
```

4.9.2. Setting JNDI properties programmatically

As an alternative to using a **`jndi.properties`** file or system properties to configure JNDI, you can define properties programmatically using the [JNDI initial context API](#).

Example: Setting JNDI properties programmatically

```
Hashtable<Object, Object> env = new Hashtable<>();  
  
env.put("java.naming.factory.initial", "org.apache.activemq.jndi.ActiveMQInitialContextFactory");  
env.put("connectionFactory.app1", "tcp://example.net:61616?jms.clientID=backends");  
env.put("queue.jobs", "app1/work-items");  
env.put("topic.notifications", "app1/updates");  
  
InitialContext context = new InitialContext(env);
```

CHAPTER 5. MESSAGE DELIVERY

5.1. WRITING TO A STREAMED LARGE MESSAGE

To write to a large message, use the **BytesMessage.writeBytes()** method. The following example reads bytes from a file and writes them to a message:

Example: Writing to a streamed large message

```
BytesMessage message = session.createBytesMessage();
File inputFile = new File(inputFilePath);
InputStream inputStream = new FileInputStream(inputFile);

int numRead;
byte[] buffer = new byte[1024];

while ((numRead = inputStream.read(buffer, 0, buffer.length)) != -1) {
    message.writeBytes(buffer, 0, numRead);
}
```

5.2. READING FROM A STREAMED LARGE MESSAGE

To read from a large message, use the **BytesMessage.readBytes()** method. The following example reads bytes from a message and writes them to a file:

Example: Reading from a streamed large message

```
BytesMessage message = (BytesMessage) consumer.receive();
File outputFile = new File(outputFilePath);
OutputStream outputStream = new FileOutputStream(outputFile);

int numRead;
byte buffer[] = new byte[1024];

for (int pos = 0; pos < message.getBodyLength(); pos += buffer.length) {
    numRead = message.readBytes(buffer);
    outputStream.write(buffer, 0, numRead);
}
```

APPENDIX A. USING YOUR SUBSCRIPTION

AMQ is provided through a software subscription. To manage your subscriptions, access your account at the Red Hat Customer Portal.

A.1. ACCESSING YOUR ACCOUNT

Procedure

1. Go to access.redhat.com.
2. If you do not already have an account, create one.
3. Log in to your account.

A.2. ACTIVATING A SUBSCRIPTION

Procedure

1. Go to access.redhat.com.
2. Navigate to **My Subscriptions**.
3. Navigate to **Activate a subscription** and enter your 16-digit activation number.

A.3. DOWNLOADING RELEASE FILES

To access .zip, .tar.gz, and other release files, use the customer portal to find the relevant files for download. If you are using RPM packages or the Red Hat Maven repository, this step is not required.

Procedure

1. Open a browser and log in to the Red Hat Customer Portal **Product Downloads** page at access.redhat.com/downloads.
2. Locate the **Red Hat AMQ** entries in the **INTEGRATION AND AUTOMATION** category.
3. Select the desired AMQ product. The **Software Downloads** page opens.
4. Click the **Download** link for your component.

A.4. REGISTERING YOUR SYSTEM FOR PACKAGES

To install RPM packages on Red Hat Enterprise Linux, your system must be registered. If you are using downloaded release files, this step is not required.

Procedure

1. Go to access.redhat.com.
2. Navigate to **Registration Assistant**.
3. Select your OS version and continue to the next page.

4. Use the listed command in your system terminal to complete the registration.

For more information, see [How to Register and Subscribe a System to the Red Hat Customer Portal](#) .

APPENDIX B. USING RED HAT MAVEN REPOSITORIES

This section describes how to use Red Hat–provided Maven repositories in your software.

B.1. USING THE ONLINE REPOSITORY

Red Hat maintains a central Maven repository for use with your Maven–based projects. For more information, see the [repository welcome page](#).

There are two ways to configure Maven to use the Red Hat repository:

- [Add the repository to your Maven settings](#)
- [Add the repository to your POM file](#)

Adding the repository to your Maven settings

This method of configuration applies to all Maven projects owned by your user, as long as your POM file does not override the repository configuration and the included profile is enabled.

Procedure

1. Locate the Maven **settings.xml** file. It is usually inside the **.m2** directory in the user home directory. If the file does not exist, use a text editor to create it.
On Linux or UNIX:

```
/home/<username>/.m2/settings.xml
```

On Windows:

```
C:\Users\<username>\.m2\settings.xml
```

2. Add a new profile containing the Red Hat repository to the **profiles** element of the **settings.xml** file, as in the following example:

Example: A Maven settings.xml file containing the Red Hat repository

```
<settings>
  <profiles>
    <profile>
      <id>red-hat</id>
      <repositories>
        <repository>
          <id>red-hat-ga</id>
          <url>https://maven.repository.redhat.com/ga</url>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>red-hat-ga</id>
          <url>https://maven.repository.redhat.com/ga</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
```

```

        <enabled>>false</enabled>
      </snapshots>
    </pluginRepository>
  </pluginRepositories>
</profile>
</profiles>
<activeProfiles>
  <activeProfile>red-hat</activeProfile>
</activeProfiles>
</settings>

```

For more information about Maven configuration, see the [Maven settings reference](#).

Adding the repository to your POM file

To configure a repository directly in your project, add a new entry to the **repositories** element of your POM file, as in the following example:

Example: A Maven pom.xml file containing the Red Hat repository

```

<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>example-app</artifactId>
  <version>1.0.0</version>

  <repositories>
    <repository>
      <id>red-hat-ga</id>
      <url>https://maven.repository.redhat.com/ga</url>
    </repository>
  </repositories>
</project>

```

For more information about POM file configuration, see the [Maven POM reference](#).

B.2. USING A LOCAL REPOSITORY

Red Hat provides file-based Maven repositories for some of its components. These are delivered as downloadable archives that you can extract to your local filesystem.

To configure Maven to use a locally extracted repository, apply the following XML in your Maven settings or POM file:

```

<repository>
  <id>red-hat-local</id>
  <url>${repository-url}</url>
</repository>

```

\${repository-url} must be a file URL containing the local filesystem path of the extracted repository.

Table B.1. Example URLs for local Maven repositories

Operating system	Filesystem path	URL
Linux or UNIX	/home/alice/maven-repository	file:/home/alice/maven-repository
Windows	C:\repos\red-hat	file:C:\repos\red-hat

When you are done running the examples, use the **artemis stop** command to stop the broker.

```
┆ $ <broker-instance-dir>/bin/artemis stop
```

Revised on 2020-06-23 07:17:00 UTC