OpenJMS User Guide

Authors:

Jim Alateras [jima@intalio.com]

Tim Anderson [tima@intalio.com]

Jim Mourikis [mourikis@intalio.com]

Revision: April 2, 2003

Introduction

What is OpenJMS?

OpenJMS is an **Open Source** implementation of Sun Microsystems's Java Message Service specification (http://java.sun.com/products/jms.html)

Features!

OpenJMS v0.7.5 supports the following features

- ✓ Topic and Queue messaging models
- ✓ Persistent and non-persistent message delivery modes
- ✓ Persistence using JDBM (http://jdbm.sourceforge.com/) or JDBC.
- ✓ QueueBrowser and Selectors
- ✓ Local Transactions
- ✓ Synchronous and Asynchronous delivery
- ✓ Administration GUI
- ✓ XML-based configuration files
- ✓ In-memory and database garbage collection
- ✓ Automatic client disconnection detection
- ✓ Support for Applet
- ✓ Integrates with Servlet containers such as Jakarta Tomcat
- ✓ Support for RMI, TCP, HTTP[1] and SSL protocol stacks
- ✓ Support for large number of destinations and subscribers.

The distribution also includes a range of examples showcasing the different messaging models and delivery modes.

About This Guide

This manual is a user's guide to OpenJMS, covering installation, building, configuration and deployment. Although the guide deals exclusively with the Microsoft Windows™ and Linux platforms, it is also pertinent to Mac OSX and other flavours of UNIX.

Support Services

1. The OpenJMS web site http://openjms.sourceforge.net/.

[1] No all features are enabled for HTTP. In particular, asynchronous message listener facility is not available.

2. The OpenJMS mailing lists are located here: http://sourceforge.net/mail/?group_id=54559.
3. The OpenJMS CVS repository is located here https://sourceforge.net/cvs/?group_id=54559

System Requirements

Operating System	Windows 98/NT/2000 or Linux or Mac OSX[2]
JVM	JDK1.2+, JDK1.3+, JDK1.4+

Getting OpenJMS

Binary Distribution

The OpenJMS binary distribution contains the latest openjms jar files, dependent libraries, examples and documentation. It does not include the source files or the facility to build OpenJMS.

The binary distribution can be downloaded from http://openjms.sourceforge.net/download.html. The name of the file is in the form of **openjms-<*version*>.tgz** (Unix) or **openjms-<*version*>.zip** (Windows), where <***version***> denotes the openjms version number (i.e. 0.7.5).

When you have downloaded the file, unpack it using an appropriate utility. The **tgz** format requires *gunzip* and *tar* whereas the **zip** format requires *winzip*.

Source Distribution

The OpenJMS source distribution is a snapshot of the CVS repository at the time of the release. It contains all the libraries, source, examples, tests and documentation required to build and run OpenJMS.

The source distribution can be downloaded from http://openjms.sourceforge.net/download.html. The name of the file is in the form of **openjms-<*version*>-src.tgz** (Unix) or **openjms-<*version*>-src.zip** (Windows), where **version** denotes the openjms version number (i.e. 0.7.5).

When you have downloaded the file, unpack it using an appropriate utility. The **tgz** format requires *gunzip* and *tar* whereas the **zip** format requires *winzip*.

CVS

The OpenJMS source can be obtained from CVS

```
cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/openjms co –r <revision> -P openjms
```

where <*revision*> is the release version, of the form openjms_<*version*>, eg openjms_0_7_5.

[2] The list only reflects the platforms that we have tested against.

CVS Snapshot

A CVS snapshot is a **work in progress** version of OpenJMS, which has not been quality assured. You can find information about the CVS repository at http://openjms.sourceforge.net/cvs.html.

To check out a clean copy of the repository use the following commands

```
cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/openjms login

cvs -z3 -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/openjms co openjms
```

NOTE: When prompted for a password for *anonymous*, simply press the Enter key.

OpenJMS also uses the *exolabcore* library, which is distributed as a *jar* in the *openjms* module. To retrieve the source code for this library you need to check out the **exolabcore** module from the repository

```
cvs -z3 -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/openjms co exolabcore[3]
```

Environment Variables

The OpenJMS server requires the following environment variables to be set in order to run:

JAVA_HOME	Path of the JDK installation directory
OPENJMS_HOME	Path of the OpenJMS installation directory

[3] assumes you have already logged in

Upgrading

From Binary Distribution

In most cases, the simplest way to download and install the latest version of the OpenJMS libraries is to download the binary distribution. This will also include a copy of all the runtime libraries, examples, and configuration files.

From CVS Repository

Released Version

If you already have a copy of the CVS repository then you can upgrade to a released version by executing the following command in the *openjms* base directory:

```
cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/openjms update -dP -r <revision>
```

where <revision> is the release version, of the form openjms_<version>, eg openjms_0_7_2.

CVS Snapshot

To upgrade to the latest version, execute the following:

```
cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/openjms update update -dP
```

WARNING the latest version in the CVS repository is not Quality Assured.

Building

OpenJMS uses Jakarta ANT v1.5 (http://jakarta.apache.org/ant) as its build tool. ANT defines all its targets in an XML-based file. To get a list of supported targets enter 'build –projecthelp'.

Building the Source

This section describes how to build the OpenJMS source distribution.

Directory Structure

The OpenJMS source distribution has the following directory structure.

Directory	Description
.	OpenJMS root directory. It contains script files to *build* and *test* OpenJMS.
bin[4]	Contains all scripts (.bat and .sh) to execute the OpenJMS server and run the examples. All scripts are executed relative to the *bin* directory.
build	This is the root directory for all generated class files and documentation
build/classes	The class files generated by compiling the openjms source files
build/doc/api	Holds the generated OpenJMS API javadoc
build/doc/javadoc	Holds the generated OpenJMS javadoc
build/doc	A local copy of the openjms web site located at http://openjms.sourceforge.net
build/examples	The class files generate by compiling the examples
build/testharness	The class files generate by compiling the test suite
config[5]	Contains all the xml configuration files, which are required by the script files.
dist	Includes the OpenJMS generated release files
lib	Holds all the libraries required to build and execute OpenJMS
src	Base directory of the OpenJMS source
src/etc	Includes the license, changelog, readme and other miscellaneous files
src/examples	Base directory for all examples and applications
src/main	Base directory of the OpenJMS source
src/testharness[6]	Base directory for all the testing code. Most of our test cases are implemented in the CTS (Compliance Test Suite), which is not part of

[4] The files in this directory are generated by the build process.

[5] The files in this directory are generated by the build process.

[6] The testharness directory is being restructured.

our open source distribution.

Preparatory work for the UNIX Environment

If you are running OpenJMS on a UNIX platform then you need to do some preparatory work before building the product. Firstly, you must ensure that the *build.sh* script is executable. To make the script executable enter the following command in the console.

```
chmod +x ./build.sh
```

Secondly, you need to execute the following shell command to prime all the other scripts for the UNIX environment.

```
 ./build.sh convert
```

Building

To build the distribution you must execute this command from the OpenJMS root directory. The root directory contains the script files *build.bat* and *build.sh.*

```
For Windows
build all

For Unix
./build.sh all
```

The output from the build should look something like this

```
Buildfile: src\build.xml

clean:

prepare:
    [mkdir] Created dir: C:\openjms\build
    [mkdir] Created dir: C:\openjms\build\classes
    [mkdir] Created dir: C:\openjms\dist

schemas:
   [castor] -- Suppressing non fatal warnings.
   [castor] -- Suppressing non fatal warnings.
   [castor] Warning : do not forget to generate the source for the schema with this
targetNamespace: http://openjms.exolab.org/connector
   [castor] Warning : do not forget to generate the source for the schema with this
targetNamespace: http://openjms.exolab.org/connector
   [castor] Warning : do not forget to generate the source for the schema with this
targetNamespace: http://openjms.exolab.org/connector
   [castor] Warning : do not forget to generate the source for the schema with this
targetNamespace: http://openjms.exolab.org/connector
   [castor] -- Suppressing non fatal warnings.

parser:
   [cantlr] ANTLR Parser Generator   Version 2.7.2a2 (20020112-1)   1989-2002 jGuru.com
   [cantlr] ANTLR Parser Generator   Version 2.7.2a2 (20020112-1)   1989-2002 jGuru.com

jdk.version:
     [echo]  Using version 1.3

jdk14work:

jdk13work:
     [echo] Compiling with JDK1.3 or below
     [copy] Copying 1 file to C:\openjms\src\main\org\exolab\jms\persistence

jdkspecific:

main:
```

```
    [javac] Compiling 465 source files to C:\openjms\build\classes

rmi:
     [rmic] RMI Compiling 6 classes to C:\openjms\build\classes

archive:
     [copy] Copying 5 files to C:\openjms\build\classes
     [copy] Copying 6 files to C:\openjms\build\classes

replacetags:
      [jar] Building jar: C:\openjms\lib\openjms-0.7.5.jar
      [jar] Building jar: C:\openjms\lib\openjms-rmi-0.7.5.jar
      [jar] Building jar: C:\openjms\lib\openjms-client-0.7.5.jar

debug-jar:

config:

prepare:

gen-scripts:

replacetags:

replacetags:

convert:

testharness:
    [mkdir] Created dir: C:\openjms\build\testharness
    [javac] Compiling 20 source files to C:\openjms\build\testharness

examples:
    [mkdir] Created dir: C:\openjms\build\examples
    [javac] Compiling 35 source files to C:\openjms\build\examples

replacetags:

prepare:

make-keystore:

prepare:

generate-keystore:

convert:

main-opt:

jar:

war:
      [war] Building war: C:\openjms\lib\openjms-0.7.5.war

all:

BUILD SUCCESSFUL
```

Building the Examples

When you download the binary distribution, you only get the examples source code. This section explains how to compile the examples.

Directory Structure

The OpenJMS binary distribution has the following directory structure

Directory	Description
.	OpenJMS root directory. It contains script files to *build* the examples. The name of the base directory is in the form of **openjms-version**, where version denotes the version number of that distribution
bin	Contains all scripts (.bat and .sh) to execute the OpenJMS server and run the examples. All scripts are executed relative to the *bin* directory.
build	This is the root directory for all generated class files
build/examples	The class files generate by compiling the examples
config	Contains all the xml configuration files, which are required by the script files.
lib	Holds all the libraries required to execute OpenJMS and associated examples
src	Base directory of the all source files
src/etc	Includes the license, changelog, readme and other miscellaneous files
src/examples	Base directory for all examples and applications

Preparatory work for the UNIX Environment

If you are running OpenJMS on a UNIX platform then you need to do some preparatory work before building the product. Firstly, you must ensure that the *build.sh* script is executable. To make the script executable enter the following command in the console.

```
chmod +x ./build.sh
```

Secondly, you need to execute the following shell command to prime all the other scripts for the UNIX environment.

```
 ./build.sh convert
```

Building

To build the examples you must execute the following command from the OpenJMS root directory. The root directory contains the script files *build.bat* and *build.sh.*

```
For Windows
build all

For Unix
./build.sh all
```

The output from the build should look something like this

```
C:\temp\openjms-0.7.2>build all
lib\xslp_1.1.jar;lib\xerces-J_1.3.1.jar;lib\oro-2.0.4.jar;lib\junit_3.7.jar;lib\exolabtools-
1.0.jar;lib\castor-0.9.3.jar;lib\antlral
l_2.7.1.jar;lib\ant_optional_1.5.jar;lib\ant_1.5.jar;c:\jdk1.3.1\lib\tools.jar
Buildfile: src\build.xml

clean:
```

```
prepare:
    [mkdir] Created dir: C:\temp\openjms-0.7.2\build

examples:
    [mkdir] Created dir: C:\temp\openjms-0.7.2\build\examples
    [javac] Compiling 35 source files to C:\temp\openjms-0.7.2\build\examples

all:

BUILD SUCCESSFUL
```

Server Scripts

Overview

A number of scripts are provided to start, stop, and administer the OpenJMS server. These are located in the $OPENJMS_HOME/bin directory. Versions exist for both Windows and UNIX – append a .bat or .sh suffix accordingly.

Script	Description
startup	Starts the OpenJMS server. On Windows, starts the server in a new window[7].
shutdown	Shuts down the OpenJMS server.
admin	Runs the OpenJMS administration tool..
openjms	The main server script, invoked by startup, shutdown, and admin.

Environment Scripts

As of version 0.7.4, the scripts ignore the global CLASSPATH. Instead the scripts customize their environment by calling **setenv.bat** (for Windows) or **setenv.sh** (for Unix) script in $OPENJMS_HOME/bin directory, if they exist.

Eg: to configure the CLASSPATH for Oracle on Windows, setenv.bat might look like:

```
rem set up the classpath to include the Oracle JDBC drivers

set CLASSPATH=c:/oracle/jdbc/lib/classes12.zip
```

Note

By default, each of the above scripts look for the OpenJMS server configuration in ***$OPENJMS_HOME/config/openjms.xml***.

A different configuration file can be specified using the **–config** argument eg:

```
> startup –config ../config/openjms-tcp.xml
```

[7] Use “openjms run” to run in the current window

Configuration

Configuration File Format

Overview

The configuration file is generated by compiling an XML Schema file with the Castor SourceGenerator (http://www.castor.org). The compiler generates Java objects for each element with corresponding setters and getters for each defined attribute or sub-element.

OpenJMS defines the following sections in its configuration file.

Configuration		
Element	Description	Required
AdminConfiguration	This section deals with configuration elements specific to the Administration GUI. From the GUI you can start, stop and configure the OpenJMS server.	1
AdministeredDestinations	This section allows administered Topic and Queue objects to be registered when the server is started. This avoids the need to create them programmatically, or via the Administration GUI.	0..1
Connectors	This section lists the connectors (eg, tcp, http, rmi) that may be used to connect to the server.	0..1
DatabaseConfiguration	All database related configuration options are specified by this section	1
GarbageCollectionConfiguration	Configures the in-memory garbage collection service	0..1
HttpConfiguration	This section is used to configure HTTP, when using an HTTP or HTTPS connector	0..1
JndiConfiguration	Information required to connect to an external JNDI provider for the purpose of registering connection factories and administered destinations.	0..1
LeaseManagerConfiguration	The options in this section deals with the Lease Manager, which is by the OpenJMS server to handle message expiration.	0..1
LoggerConfiguration	This section deals with the logger module, which uses the log4j package exclusively.	0..1

MessageManagerConfiguration	The message manager configuration section relates to the message manager core	0..1
RmiConfiguration	This section is used to configure RMI, when using an RMI connector	0..1
ServerConfiguration	The server configuration section relates to the server core	0..1
SchedulerConfiguration	Configures options for the OpenJMS scheduler service	0..1
TcpConfiguration	This section is used to configure TCP, when using a TCP or TCPS connector	0..1

Examples

The following is a minimal configuration for an OpenJMS server running on Windows, using the JDBM database for persistency.

```
<Configuration>
  <AdminConfiguration
    script="${openjms.home}\bin\startup.bat"
    config="${openjms.home}\config\openjms.xml" />

  <DatabaseConfiguration>
    <JdbmDatabaseConfiguration name="${openjms.home}\openjms.db" />
  </DatabaseConfiguration>
</Configuration>
```

The following is a minimal configuration for an OpenJMS server running on UNIX, using the JDBM database for persistency.

```
<Configuration>
  <AdminConfiguration
    script="${openjms.home}/bin/startup.sh"
    config="${openjms.home}/config/openjms.xml" />

  <DatabaseConfiguration>
    <JdbmDatabaseConfiguration name="${openjms.home}/openjms.db" />
  </DatabaseConfiguration>
</Configuration>
```

Administration Configuration

AdminConfiguration		
Attribute	Description	Required
script	The relative or absolute file name of script used to start the OpenJMS server.	1
config	The relative or absolute file name of the XML configuration file used by the OpenJMS server	1

Examples

The administration configuration for an OpenJMS server, running on Windows.

```
<AdminConfiguration
    script="${openjms.home}\bin\startup.bat"
    config="${openjms.home}\config\openjms.xml" />
```

The administration configuration for an OpenJMS server, running on UNIX.

```
<AdminConfiguration
    script="${openjms.home}/bin/startup.sh"
    config="${openjms.home}/config/openjms.xml" />
```

NOTE: the ${openjms.home} variable is replaced with the value of the OPENJMS_HOME environment variable, which points to the OpenJMS root directory.

Binding Administered Destinations

AdministeredDestinations		
Element	Description	Required
AdministeredTopic	Defines a list of administered topics.	0..*
AdministeredQueue	Defines a list of administered queues.	0..*

AdministeredTopic		
Attribute	Description	Required
name	The name of the administered topic that is registered at server startup time. Each topic has zero or more registered durable subscribers	1
Element		
Subscriber	This element is used to register a durable subscriber for the topic.	0..*

Subscriber		
Attribute	Description	Required
name	The name of the durable subscriber that is registered at server startup time.	1

AdministeredQueue		
Attribute	Description	Required
name	The name of the administered queue that is registered at server startup time.	1

Examples

The following creates an administered topic ‘topic1’ with two durable subscribers ‘sub1’ and ‘sub2’, and three administered queues, ‘queue1’, ‘queue2’, and ‘queue3’

```
<AdministeredDestinations>
   <AdministeredTopic topic="topic1">
        <Subscriber name="sub1" />
        <Subscriber name="sub2" />
    </AdministeredTopic>

    <AdministeredQueue name="queue1" />
    <AdministeredQueue name="queue2" />
    <AdministeredQueue name="queue3" />
</AdministeredDestinations>
```

Connectors

Connectors specify the transport protocols that may be used to connect to an OpenJMS server. OpenJMS supports the following connectors:

- tcp
- tcps
- http
- https
- rmi
- embedded (used when the JMS client and OpenJMS server run in the same JVM)

Each configured connector must have a set of connection factories registered for it. These are bound in JNDI and enable clients to create new javax.jms.Connection instances.

Connectors		
Element	Description	Required
Connector	Specifies the type of transport protocols that the OpenJMS server should use, and the connection factories to register for that protocol.	1..*

Connector		
Attribute	Description	Required
scheme	The type of protocol to use. Valid values for this attribute are “tcp”, “tcps”, “http”, “https”, “rmi”, or “embedded”	1
Element		
ConnectionFactories	Specifies the list of connection factories to bind in JNDI for the connector.	1

ConnectionFactories

Element	Description	Required
QueueConnectionFactory	Binds a javax.jms.QueueConnectionFactory in JNDI with the specified name	0..*
TopicConnectionFactory	Binds a javax.jms.TopicConnectionFactory in JNDI with the specified name	0..*
XAQueueConnectionFactory	Binds a javax.jms.XAQueueConnectionFactory in JNDI with the specified name	0..*
XATopicConnectionFactory	Binds a javax.jms.XATopicConnectionFactory in JNDI with the specified name	0..*

QueueConnectionFactory		
Attribute	Description	Required
name	The name of the QueueConnectionFactory bound in JNDI	1

TopicConnectionFactory		
Attribute	Description	Required
name	The name of the TopicConnectionFactory bound in JNDI	1

XAQueueConnectionFactory		
Attribute	Description	Required
name	The name of the XAQueueConnectionFactory bound in JNDI	1

XATopicConnectionFactory		
Attribute	Description	Required
name	The name of the XATopicConnectionFactory bound in JNDI	1

Examples

The following specifies to use both a ‘tcp’ and ‘rmi’ connector, and binds each of the connection factories in JNDI. This will enable one OpenJMS server to support both requests from a TCP client and an RMI client.

```
<Connectors>
  <Connector scheme="tcp">
    <ConnectionFactories>
      <QueueConnectionFactory name="QueueConnectionFactory" />
      <TopicConnectionFactory name="TopicConnectionFactory" />
```

```
      <XAQueueConnectionFactory name="XAQueueConnectionFactory" />
      <XATopicConnectionFactory name="XATopicConnectionFactory" />
    </ConnectionFactories>
  </Connector>
  <Connector scheme="rmi">
    <ConnectionFactories>
      <QueueConnectionFactory name="rmiQueueConnectionFactory" />
      <TopicConnectionFactory name="rmiTopicConnectionFactory" />
      <XAQueueConnectionFactory name="rmiXAQueueConnectionFactory" />
      <XATopicConnectionFactory name="rmiXATopicConnectionFactory" />
    </ConnectionFactories>
  </Connector>
</Connectors>
```

Database Configuration

DatabaseConfiguration		
Attribute	Description	Required
garbageCollectionInterval	To automatically remove processed persistent messages from the database, specify this attribute with an interval, in seconds. The interval indicates how often messages will be checked. Note that the current algorithm , which deletes messages is very expensive and a high interval should be set. In addition this facility can be disabled if you are only using the queue message model	0..1
garbageCollectionBlockSize	This attribute hints on the block size that the garbage collector should use when removing messages. It can impact the performance of the system	0..1
garbageCollectionThreadPriority	This is the priority assigned to the garbage collection thread. It ranges from 1-10 and if one is not allocated it defaults to 5.	0..1
Element		
RdbmsDatabaseConfiguration	Specifies the database configuration for a JDBC compliant database.	0..1
JdbmDatabaseConfiguration	Specifies the database configuration for a JDBM (or object) database.	0..1

RdbmsDatabaseConfiguration		
Attribute	Description	Required
driver	The JDBC driver class. This must be XA	1

	compliant.	
url	The database URL.	1
user	The user name that OpenJMS uses to access the database	1
password	The user's password	1
retries	The number of times to retry a failed transaction. If not specified it defaults to 5	0..1
timeout	The interval, in seconds, between transaction retries. If not specified it defaults to *2* seconds	0..1

JdbmDatabaseConfiguration		
Attribute	Description	Required
name	The JDBM database path name.	1
cacheSize	The size of the JDBM database cache held in memory. All recently accessed objects are cached in memory to improve performance.	0..1

Examples

The following specifies to use an RDBMS for persistency, in this case MySQL.

```
<DatabaseConfiguration
  garbageCollectionInterval="180"
  garbageCollectionBlockSize="500"
  garbageCollectionThreadPriority="5">
  <RdbmsDatabaseConfiguration
    driver="org.gjt.mm.mysql.Driver"
    url="jdbc:mysql://localhost/test"
    user="openjms"
    password="openjms"
    retries="5"
    timeout ="2" />
</DatabaseConfiguration>
```

The following specifies to use JDBM for persistency.

```
<DatabaseConfiguration
  garbageCollectionInterval="180"
  garbageCollectionBlockSize="500"
  garbageCollectionThreadPriority="5">
  <JdbmDatabaseConfiguration name="${openjms.home}/openjms.db" />
</DatabaseConfiguration>
```

HTTP Configuration

HttpConfiguration

Attribute	Description	Required
host	The web server host. Defaults to *localhost* if not set.	0..1
port	The web server port. Defaults to *8080* if not set.	0..1
proxyHost	The proxy host used to connect back to clients if required.	0..1
proxyPort	The proxy port used to connect back to clients if required.	0..1
clientPingInterval	The client ping interval, specified in seconds. Defaults to *15* if not set. If set to 0, the ping is disabled.	0..1

Examples

```
<HttpConfiguration host="localhost"
                   port="8080"
                   clientPingInterval="20" />
```

Garbage Collection Configuration

GarbageCollectionConfiguration		
Attribute	Description	Required
memoryCheckInterval	Indicates how often the server will check the memory utilization of the server. It is specified in seconds and defaults to 30 seconds. It will check to ensure that the ratio of free memory to total memory doesn't fall below the lowWaterThreshold.	0..1
lowWaterThreshold	The ratio of free memory to total memory, specified as a percentage, which will trigger GC. The default value of 20, indicates that when free memory falls below 20% of total memory (i.e. total VM memory) then garbage collection will be triggered. The range of valid values is between 10-50.	0..1
garbageCollectionInterval	Indicates how often, in seconds, the in memory garbage collector will run to remove processed messages from the cache. The value is specified in seconds. A value of zero will disable this capability. The default value is 300 seconds. THIS IS NOT LONGER USED	0..1

garbageCollectionThreadPriority	The priority assigned to the garbage collection thread. It ranges from 1-10 and defaults to 5.	0..1

Examples

```
<GarbageCollectionConfiguration
  memoryCheckInterval="60"
  lowWaterThreshold="20"
  garbageCollectionInterval="120"
  garbageCollectionThreadPriority="5" />
```

JNDI Configuration

JndiConfiguration		
Element	Description	Required
property	Properties used to instantiate an InitialContext	0..*

property		
Attribute	Description	Required
name	The JNDI property name	1
value	The JNDI property value	1

Examples

The following JNDI configuration specifies the properties to use rmiregistry as the JNDI provider.

```
<JndiConfiguration>
  <property name="java.naming.factory.initial"
            value="com.sun.jndi.rmi.registry.RegistryContextFactory" />
  <property name="java.naming.provider.url" value="rmi://localhost:3031" />
</JndiConfiguration>
```

Lease Manager Configuration

LeaseManagerConfiguration		
Attribute	Description	Required
sleepTime[8]	Time, in milliseconds reaper thread will sleep if there is no work to be done.	0..1

[8] This will be removed in the near future

Examples

```
<LeaseManagerConfiguration sleepTime="5000" />
```

Logger Configuration

LoggerConfiguration		
Attribute	Description	Required
file	The name of the log4j configuration file, which is an xml file conforming to log4j.dtd	1

Examples

Logger configuration using the default log4j file installed in <openjms home>/config.

```
  <LoggerConfiguration
      file="${openjms.home}/config/log4j.xml" />
```

Message Manager Configuration

MessageManagerConfiguration		
Attribute	Description	Required
destinationCacheSize	The maximum size of a destination cache before non-persistent messages are discarded. This is used to limit the memory consumption of the JMS server. If the cache exceeds this size then new non-persistent messages are dropped and persistent messages are evicted from memory	1

Examples

```
<MessageManagerConfiguration destinationCacheSize = "10000" />
```

RMI Configuration

RmiConfiguration		
Attribute	Description	Required
embeddedRegistry	Determines whether to run an embedded or external RMI registry. To run an external	0..1

	RMI registry then set this to *false.*	
registryHost	The host name or the IP address of the machine hosting the RMI Registry. It defaults to *localhost.*	0..1
registryPort	The port number that the RMI Registry is using. It defaults to *1099.*	0..1
clientPingInterval	The client ping interval, specified in seconds. If set to 0, the ping is disabled.	0..1
serverName	The name of the OpenJMS server. This must be unique within the RMI registry being used.	0..1
jndiName	The name of the JNDI server. This must be unique within the RMI registry being used.	0..1
adminName	The name of the Administration server. This must be unique within the RMI registry being used.	0..1

Examples

The following specifies an embedded RMI registry, running on port 1099.

```
    <RmiRegistryConfiguration
        embeddedRegistry = "true"
        registryPort = "1099" />
```

The following specifies an external RMI registry, running on host ‘myhost’ port 1099.

```
    <RmiRegistryConfiguration
        embeddedRegistry="false"
        registryHost=”myhost”
        registryPort = "1099" />
```

Server Configuration

ServerConfiguration		
Attribute	Description	Required
host	The address of the machine hosting the OpenJMS server. If this is not explicitly specified it will default to *localhost.* If you are running the server and clients across a number of machines then you must use either the hostname or the IP address of the machine. In addition, if the machine running the OpenJMS server is a multi-homed host, then you must specified one of the IP addresses.	0..1

embeddedJNDI	This specifies whether to use an embedded (or internal) JNDI provider, or an external one. If not specified, it will default to *true*. If it is set to false, then the JndiConfiguration element is should be configured.	0..1

Examples

OpenJMS server running on host ‘myhost’ using an embedded JNDI provider.

```
<ServerConfiguration host=”myhost” embeddedJNDI=”true” />
```

OpenJMS server running on host ‘myhost’ using an external JNDI provider running on ‘myotherhost’, port 1099.

```
<ServerConfiguration host=”myhost” embeddedJNDI=”false” />

<JndiConfiguration>
  <property name="java.naming.factory.initial"
            value="com.sun.jndi.rmi.registry.RegistryContextFactory" />
  <property name="java.naming.provider.url" value="rmi://myotherhost:1099" />
</JndiConfiguration>
```

Scheduler Configuration

SchedulerConfiguration		
Attribute	Description	Required
maxThreads	The maximum number of worker threads that the scheduler uses.	1

Examples

```
<SchedulerConfiguration maxThreads="10" />
```

TCP Configuration

TcpConfiguration		
Attribute	Description	Required
internalHost[9]	This is only applicable when the server is behind a NAT firewall. This becomes the internal address the server is known by and the host address in ServerConfiguration is the external address.	0..1

[9] this may change in the future

	Clients will attempt to connect to ServerConfiguration/host first. If that fails, they will try to connect to internalHost	
port	The port number that the server runs on. Defaults to *3030*.	0..1
jndiPort	The JNDI port, if an embedded JNDI provider is being used. Defaults to *3035*.	0..1

Examples

```
<TcpConfiguration port="3030" jndiPort=”3035” />
```

Other Configuration Files

Log4j Configuration File

OpenJMS uses the Jakarta Log4J logger component, which is configured through an XML file. The LoggerConfiguration element, in the OpenJMS configuration file points to this file

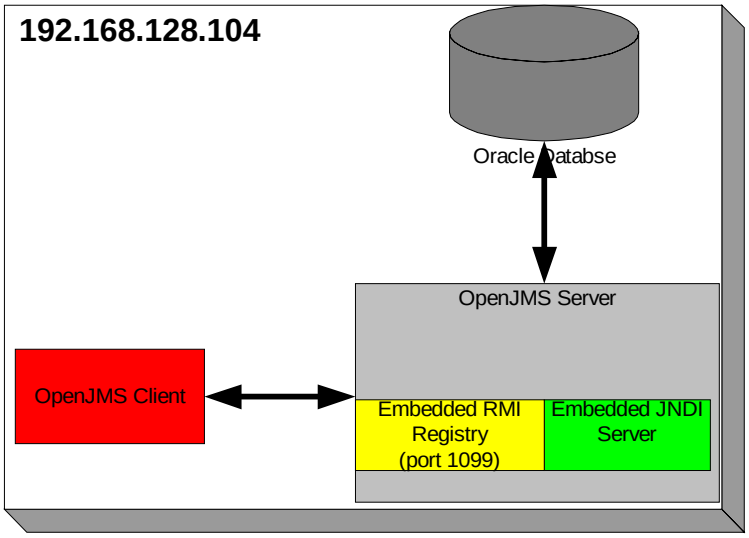
```
<LoggerConfiguration file="${openjms.home}/config/log4j.xml” />
```

The sample Log4J configuration file is shown below but you should consult the web site http://jakarta.apache.org/log4j/docs/index.html for more information

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j='http://jakarta.apache.org/log4j/'>

  <appender name="STDOUT" class="org.apache.log4j.ConsoleAppender">
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern"
             value="%d{HH:mm:ss.SSS} %-5p [%t] - %m\n"/>
    </layout>

  </appender>

  <appender name="openjms" class="org.apache.log4j.FileAppender">
    <param name="File" value="openjms.log" />
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern"
             value="%d{HH:mm:ss.SSS} %-5p [%t] - %m\n"/>
    </layout>

  </appender>

  <category name="exolab">
    <priority value="info" />
    <appender-ref ref="STDOUT" />
  </category>

</log4j:configuration>
```

Configuration of an RMI OpenJMS Server

This section describes how to configure an RMI OpenJMS Server using an Oracle database where the client and the server are running on the same machine. The RMI Registry, which is embedded in the OpenJMS server, is running on port 1099. The JNDI Server registers with the connection factory and subsequently the OpenJMS binds its connection factories in the root JNDI context. The client must perform a lookup on the RMI registry to get a reference to the root JNDI context.

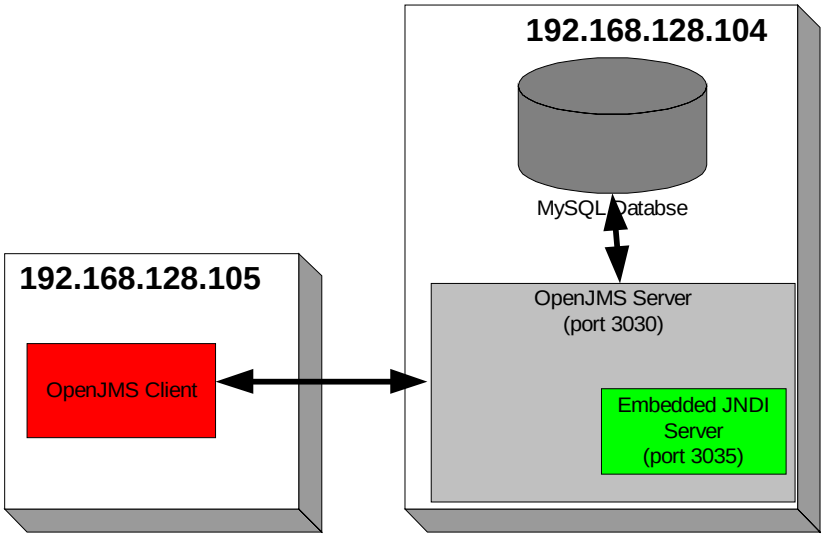



```
<?xml version="1.0"?>
<Configuration>
  <Connectors>
    <Connector scheme="rmi">
      <ConnectionFactories>
        <QueueConnectionFactory name="JmsQueueConnectionFactory" />
        <TopicConnectionFactory name="JmsTopicConnectionFactory" />
      </ConnectionFactories>
    </Connector>
  </Connectors>

  <DatabaseConfiguration>
    <RdbmsDatabaseConfiguration
      driver="oracle.jdbc.driver.OracleDriver"
      url="jdbc:oracle:oci8:@openjms"
      user="openjms"
      password="openjms"
      retries="5"
      timeout="5" />
  </DatabaseConfiguration>

  <AdminConfiguration
     script="${openjms.home}\bin\startup.bat"
     config="${openjms.home}\config\openjms.xml" />

  <AdministeredDestinations>
    <AdministeredTopic name="topic1">
      <Subscriber name="sub1" />
      <Subscriber name="sub2" />
    </AdministeredTopic>

    <AdministeredQueue name="queue1" />
    <AdministeredQueue name="queue2" />
    <AdministeredQueue name="queue3" />
  </AdministeredDestinations>
```

```
</Configuration>
```

Configuration of a TCP OpenJMS Server

This section describes how to configure an TCP OpenJMS Server using the MySQL database. In this configuration the client and server are running on different machines. The JNDI server, which the client initially uses to discover the connection factories is running on port 3035 and the OpenJMS sever, which the client connects to and exchanges messages with is running on port 3030.



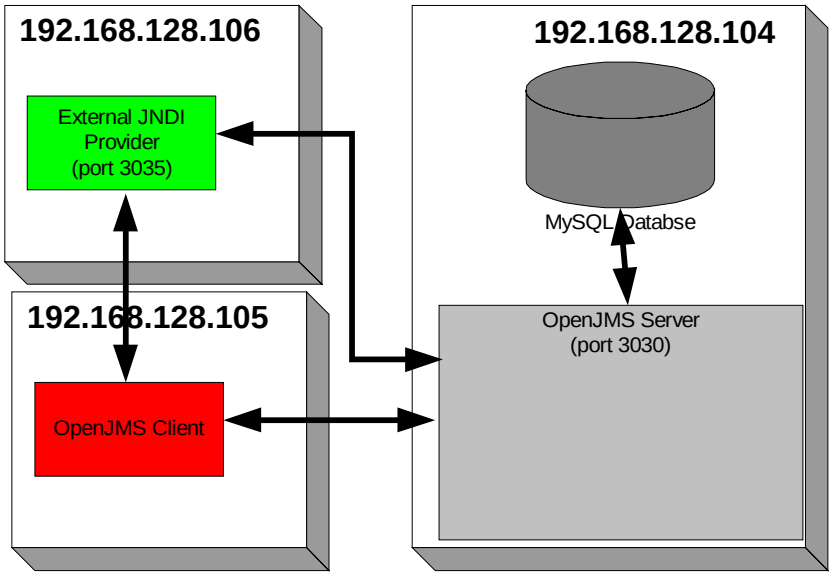

```
<?xml version="1.0"?>

<Configuration>

  <!-- Optional. The tcp connector is the default connector -->
  <Connectors>
    <Connector scheme="tcp">
      <ConnectionFactories>
        <QueueConnectionFactory name="JmsQueueConnectionFactory" />
        <TopicConnectionFactory name="JmsTopicConnectionFactory" />
      </ConnectionFactories>
    </Connector>
  </Connectors>

  <!-- Optional. This represents the default configuration  -->
  <TcpConfiguration port="3030" jndiPort="3035" />

  <DatabaseConfiguration>
    garbageCollectionInterval="180"
    garbageCollectionBlockSize="500"
    garbageCollectionThreadPriority="5">
    <RdbmsDatabaseConfiguration
      driver="org.gjt.mm.mysql.Driver"
      url="jdbc:mysql://localhost/test"
      user="openjms"
      password="openjms"
      retries="5"
      timeout ="5" />
  </DatabaseConfiguration>

  <AdminConfiguration
```

```
    script="${openjms.home}\bin\startup.bat"
    config="${openjms.home}\config\openjms.xml" />

  <AdministeredDestinations>
    <AdministeredTopic name="topic1">
      <Subscriber name="sub1" />
      <Subscriber name="sub2" />
    </AdministeredTopic>

    <AdministeredQueue name="queue1" />
    <AdministeredQueue name="queue2" />
    <AdministeredQueue name="queue3" />
  </AdministeredDestinations>

</Configuration>
```

Support for External JNDI Provider

OpenJMS can be run against an external JNDI compliant naming service. In the configuration illustrated below, the JNDI provider is running on 192.168.128.106, the OpenJMS server on 192.168.128.104 and the client on 192.168.128.105.

The OpenJMS Server and the OpenJMS client both utilise the JNDI provider to bind and lookup connection factories and administered destinations respectively.



The OpenJMS configuration files, supporting this configuration are show below.

```
<?xml version="1.0"?>

<Configuration>

  <!-- Optional. The tcp connector is the default connector -->
  <Connectors>
    <Connector scheme="tcp">
      <ConnectionFactories>
        <QueueConnectionFactory name="JmsQueueConnectionFactory" />
        <TopicConnectionFactory name="JmsTopicConnectionFactory" />
      </ConnectionFactories>
    </Connector>
  </Connectors>

  <ServerConfiguration host="192.168.128.104" embeddedJNDI="false" />
```

```
  <!-- Optional. This represents the default configuration  -->
  <TcpConfiguration port="3030" jndiPort="3035" />

  <JndiConfiguration>
    <property name="java.naming.factory.initial"
              value="com.sun.jndi.rmi.registry.RegistryContextFactory" />
    <property name="java.naming.provider.url"
              value="rmi:// 192.168.128.106:1099" />
  </JndiConfiguration>

  <DatabaseConfiguration>
    garbageCollectionInterval="180"
    garbageCollectionBlockSize="500"
    garbageCollectionThreadPriority="5">
    <RdbmsDatabaseConfiguration
      driver="org.gjt.mm.mysql.Driver"
      url="jdbc:mysql://localhost/test"
      user="openjms"
      password="openjms"
      retries="5"
      timeout ="5" />
  </DatabaseConfiguration>

  <AdminConfiguration
     script="${openjms.home}\bin\startup.bat"
     config="${openjms.home}\config\openjms.xml" />

  <AdministeredDestinations>
    <AdministeredTopic name="topic1">
      <Subscriber name="sub1" />
      <Subscriber name="sub2" />
    </AdministeredTopic>

    <AdministeredQueue name="queue1" />
    <AdministeredQueue name="queue2" />
    <AdministeredQueue name="queue3" />
  </AdministeredDestinations>

</Configuration>
```

Sample Database Configurations

This portion of the document provides sample configuration files for some of more popular databases.

Oracle

The following configuration requires **classes12.zip** (or equivalent) to be in the classpath.

```
    <DatabaseConfiguration>
        <RdbmsDatabaseConfiguration
            driver="oracle.jdbc.driver.OracleDriver"
            url="jdbc:oracle:oci8:@myhostname"
            user="openjms"
            password="openjms"
            retries="5"
            timeout ="2000" />
    </DatabaseConfiguration>
```

Sybase

The following configuration requires **jconn2.jar** (or equivalent) to be in the classpath.

```
<DatabaseConfiguration>
    <RdbmsDatabaseConfiguration
        driver="com.sybase.jdbc2.jdbc.SybDriver"
        url="jdbc:sybase:Tds:myhostname:2048/mydatabase"
        user="openjms"
        password="openjms"
        retries="5"
        timeout ="2000" />
</DatabaseConfiguration>
```

MySQL

The following requires **mm.mysql-2.0.4-bin.jar** (or equivalent) to be in the classpath. The driver can be downloaded from http://mmmysql.sourceforge.net/.

```
<DatabaseConfiguration>
    <RdbmsDatabaseConfiguration
        driver="org.gjt.mm.mysql.Driver"
        url="jdbc:mysql://localhost/openjms"
        user="openjms"
        password="openjms"
        retries="5"
        timeout ="2000" />
</DatabaseConfiguration>
```

HSQL

The following requires **hsqldb_1.61.jar** (or equivalent) to be in the classpath. The driver can be downloaded from http://hsql.sourceforge.net.

```
<DatabaseConfiguration>
  <RdbmsDatabaseConfiguration
          driver="org.hsqldb.jdbcDriver"
          url="jdbc:hsqldb:openjms.db"
          user="sa"
          password="" />
</DatabaseConfiguration>
```

Interbase

The following configuration requires **interclient.jar** (or equivalent) to be in the classpath.

NOTE: you need to run InterServer 2.02 or higher. Earlier versions will not work with dbtool.

```
<DatabaseConfiguration>
    <RdbmsDatabaseConfiguration
        driver="interbase.interclient.Driver"
        url="jdbc:interbase://myhostname/c:/openjms/openjms.gdb"
        user="openjms"
        password="openjms"
        retries="5"
        timeout ="2000"  />
</DatabaseConfiguration>
```

JDBM[10]

```
    <DatabaseConfiguration>
        garbageCollectionInterval="180"
        garbageCollectionBlockSize="500"
        garbageCollectionThreadPriority = "5" >
        <JdbmDatabaseConfiguration name=”openjms.db” />
    </DatabaseConfiguration>
```

[10] Support for JDBM will be dropped in the near future

Configuring a JDBC Database

OpenJMS may be configured to use an JDBC 2.0 compliant driver to support persistent messages. The following databases have been tested.

Database	Version	Web Site
Oracle8i	8.1.7	http://www.oracle.com/
Sybase ASE	12.0	http://www.sybase.com/
Borland InterBase	6.0.1	http://www.borland.com/interbase
MySQL	3.23.39	http://www.mysql.com/
HSQL	1.61	http://hsql.sourceforge.net

The following tasks must be completed to configure OpenJMS and the database

1. Add the JDBC driver to the classpath
2. Edit the OpenJMS configuration file
3. Execute the **dbtool** to create the database tables

Adding JDBC Driver to the classpath

Add the relevant JDBC driver to the classpath. A list of databases and their associated drivers is shown below.

Database	Version	JDBC Driver
Oracle8i	8.1.7	classes12.zip
Sybase ASE	12.0	jconn2.jar
Borland InterBase	6.0.1	interclient.jar
MySQL	3.23.39	mm.mysql-2.0.4-bin.jar
HSQL	1.61	hsqldb_1.61.jar

The openjms scripts ignore the CLASSPATH set in the global environment. To customize the CLASSPATH, you need to create a ***setenv.bat*** script (***setenv.sh*** script on UNIX) in the $OPENJMS_HOME/bin directory. Eg: to configure the CLASSPATH for Oracle, setenv.bat might look like:

```
set CLASSPATH=c:/oracle/jdbc/lib/classes12.zip
```

Edit the OpenJMS configuration file

Edit the OpenJMS configuration and modify the attributes of the *DatabaseConfiguration* element. Refer to Sample Database Configurations for examples.

Execute the dbtool application

The dbtool application may be used to create, drop, and recreate OpenJMS database tables and indexes. To create tables, run the following:

```
[windows]
cd <openjms root dir>\bin
dbtool.bat –create –config <config file>.xml

[unix]
cd <openjms root dir>/bin
dbtool.sh –create –config <config file>.xml
```

What If dbtool doesn’t work?

The dbtool application may not support all available JDBC drivers, due to buggy JDBC implementations. In this case, the tables and the indexes must be manually created. The OpenJMS distribution ships with SQL scripts for most popular databases. These scripts are located in *<openjms root dir>/config* directory and are in the form of create_*db*.sql (i.e. create_oracle.sql, create_mysql.sql).

For example to manually create the tables and indexes for an Oracle database use the following command line

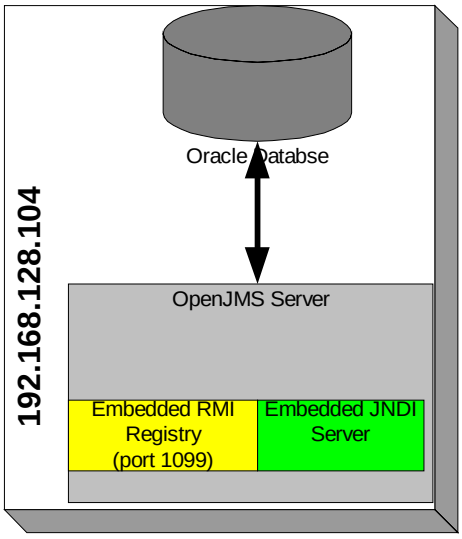
```
sqlplus user/password @create_oracle.sql
```

Starting the Server

Once the database has been correctly configured the OpenJMS server can be started. This part of the document describes how to start an RMI based and a TCP based server.

RMI Server

This particular example will start a server with an embedded JNDI provider, as illustrated below. The server uses the JNDI provider to bind all its connection factories and administered destinations.



The OpenJMS server uses the following configuration file

```
<?xml version="1.0"?>
<<Configuration>

  <Connectors>
    <Connector scheme="rmi">
      <ConnectionFactories>
        <QueueConnectionFactory name="JmsQueueConnectionFactory" />
        <TopicConnectionFactory name="JmsTopicConnectionFactory" />
      </ConnectionFactories>
    </Connector>
  </Connectors>

  <ServerConfiguration host="192.168.128.104" />

  <DatabaseConfiguration>
    <RdbmsDatabaseConfiguration
      driver="oracle.jdbc.driver.OracleDriver"
      url="jdbc:oracle:oci8:@myhostname"
      user="openjms"
      password="openjms"
      retries="5"
      timeout ="2000" />
  </DatabaseConfiguration>

  <AdminConfiguration
     script="${openjms.home}\bin\startup.bat"
     config="${openjms.home}\config\openjms.xml" />
```

```
  <!-- Optional. If not specified, no destinations will be created -->
  <AdministeredDestinations>
    <AdministeredTopic name="topic1">
      <Subscriber name="sub1" />
      <Subscriber name="sub2" />
    </AdministeredTopic>

    <AdministeredQueue name="queue1" />
    <AdministeredQueue name="queue2" />
    <AdministeredQueue name="queue3" />
  </AdministeredDestinations>

</Configuration>
```

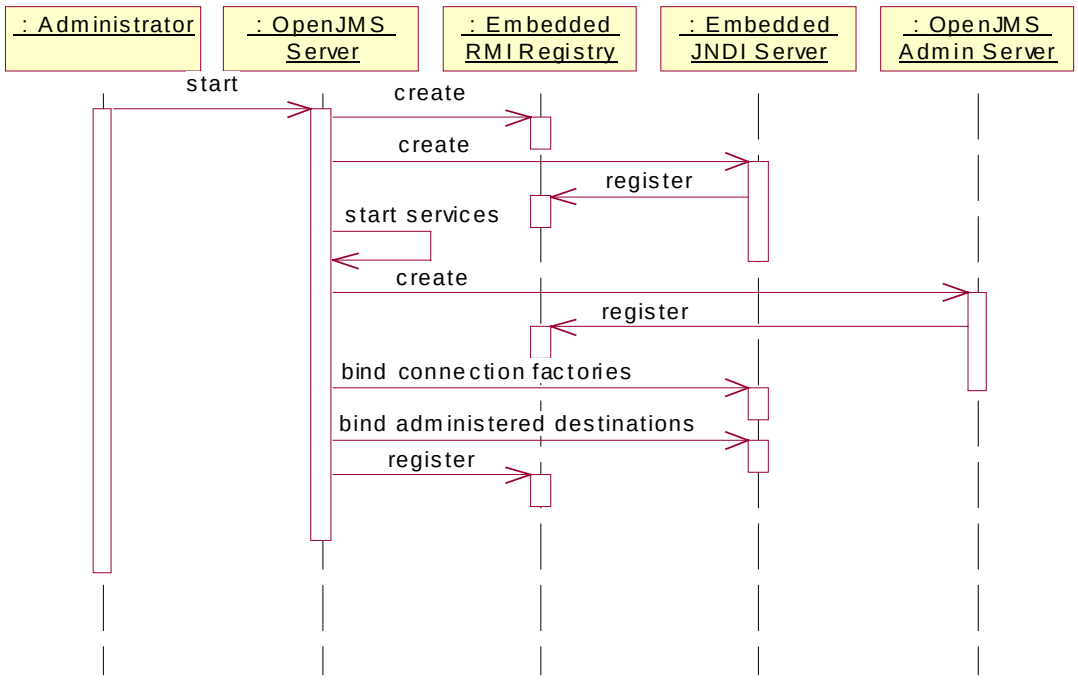
To start the server go to the *bin* directory and enter the following command

```
[windows]
startup

[unix]
./startup.sh
```

A message log, similar to the one shown below, will be displayed on your console. The OpenJMS server starts a number an embedded RMI Registry, which it uses to bind a reference to the JndiServer, the OpenJmsServer and the JmsAdminServer. Next it starts all the low level services and facilities. Finally it binds any configured administered destinations and connection factories to the root JNDI Context.

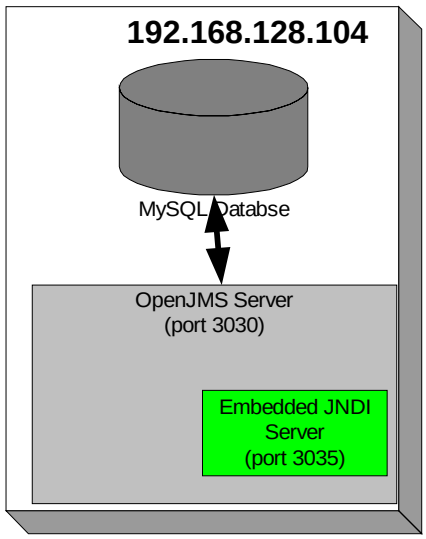
```
OpenJMS 0.7.5 (build 17)
Exolab Inc. (C) 1999-2003. All rights reserved. openjms.sourceforge.net
23:31:04.505 INFO  [main] - Instantiated the RmiRegistryService service on port 1099
23:31:04.515 INFO  [main] - Embedded RMI Registry running on port 1099
23:31:04.976 INFO  [main] - Registered the service JndiServer with the registry.
23:31:04.976 INFO  [main] - Started service [RmiRegistryService]
23:31:04.976 INFO  [main] - Started service [ThreadPoolManager]
23:31:04.976 INFO  [main] - Started service [EventManagerThread]
23:31:05.156 INFO  [main] - Removed expired messages.
23:31:05.156 INFO  [main] - Started service [DatabaseService]
23:31:05.156 INFO  [main] - Started service [Scheduler]
23:31:05.156 INFO  [main] - Started service [LeaseManagerReaper]
23:31:05.156 INFO  [main] - Registering Garbage Collection every 60000 for memory.
23:31:05.166 INFO  [main] - Registering Garbage Collection every 120000 for other resources.
23:31:05.166 INFO  [main] - Started service [GCCollectionService]
23:31:05.336 INFO  [main] - Started service [MessageManager]
23:31:05.356 INFO  [main] - JMS Server is bound to //localhost:1099/OpenJMSServer
23:31:05.376 INFO  [main] - JMS Admin Server is bound to //localhost:1099/JmsAdminServer
```

An interaction diagram of the startup sequence is presented below.



TCP Server

The TCP Server example, illustrated below, also utilises an embedded JNDI server.



The configuration file for the TCP server is shown below

```
<?xml version="1.0"?>
<Configuration>

  <!-- Optional. The tcp connector is the default connector -->
  <Connectors>
    <Connector scheme="tcp">
      <ConnectionFactories>
        <QueueConnectionFactory name="JmsQueueConnectionFactory" />
        <TopicConnectionFactory name="JmsTopicConnectionFactory" />
      </ConnectionFactories>
    </Connector>
  </Connectors>
```

```
  <ServerConfiguration host="192.168.128.104" />

  <DatabaseConfiguration>
    <RdbmsDatabaseConfiguration
      driver="oracle.jdbc.driver.OracleDriver"
      url="jdbc:oracle:oci8:@myhostname"
      user="openjms"
      password="openjms"
      retries="5"
      timeout ="2000" />
  </DatabaseConfiguration>

  <AdminConfiguration
     script="${openjms.home}\bin\startup.bat"
     config="${openjms.home}\config\openjms.xml" />

  <!-- Optional. If not specified, no destinations will be created -->
  <AdministeredDestinations>
    <AdministeredTopic name="topic1">
      <Subscriber name="sub1" />
      <Subscriber name="sub2" />
    </AdministeredTopic>

    <AdministeredQueue name="queue1" />
    <AdministeredQueue name="queue2" />
    <AdministeredQueue name="queue3" />
  </AdministeredDestinations>

</Configuration>
```

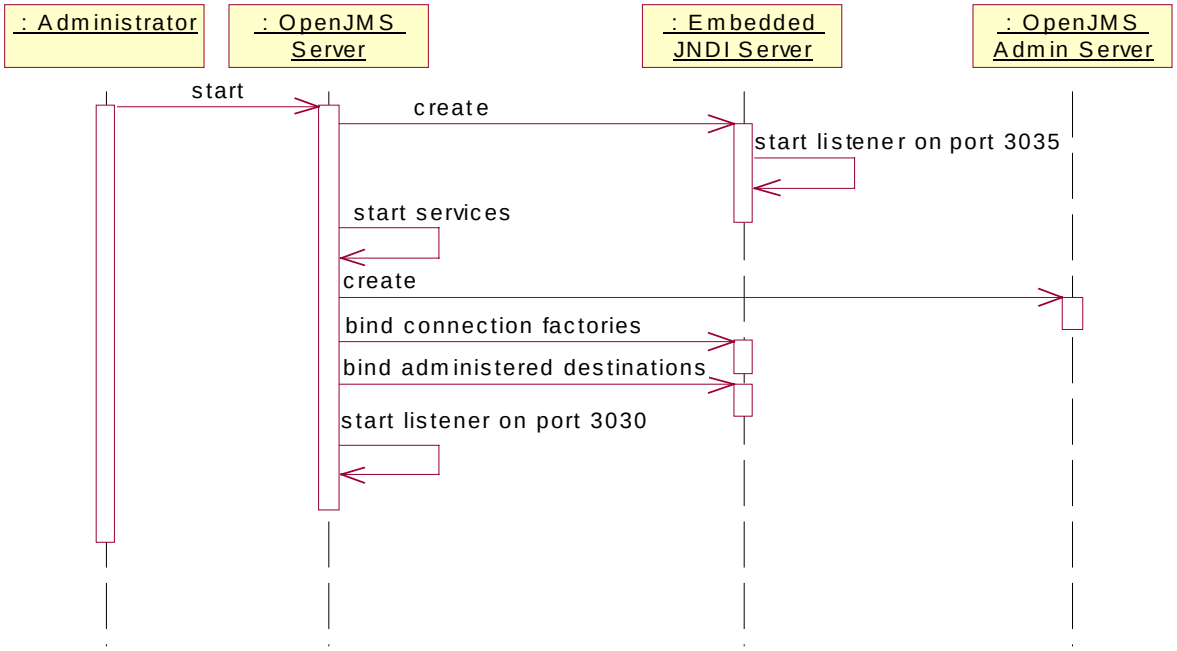
To start the server go to the *bin* directory and enter the following command.

```
[windows]
startup

[unix]
./startup.sh
```

A message log, similar to the one shown below, will be displayed on your console. The OpenJMS server starts the embedded JNDI server, which listens for connections on port 3035, starts all the low level services and facilities, binds the connection factories and administered destinations and finally listens for connections on port 3030.

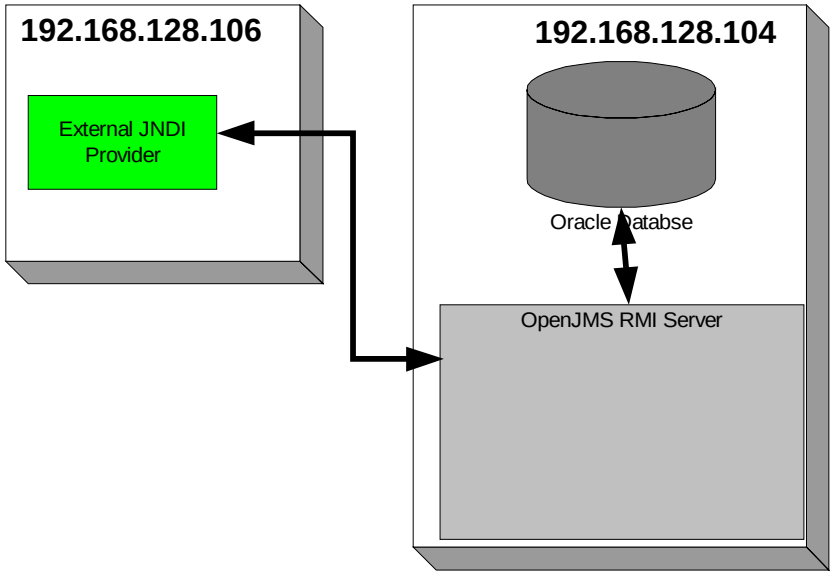
```
OpenJMS 0.7.2 (build 10)
Exolab Inc. (C) 1999-2002. All rights reserved. www.openjms.org
11:18:24.793 INFO  [main] - Started TCP JNDI Server on port 3035
11:18:25.454 INFO  [main] - Started the JNDI service
11:18:25.454 INFO  [main] - Started service [ThreadPoolManager]
11:18:25.464 INFO  [main] - Started service [EventManagerThread]
11:18:25.915 INFO  [main] - Removed expired messages.
11:18:25.925 INFO  [main] - Started service [DatabaseService]
11:18:25.925 INFO  [main] - Started service [Scheduler]
11:18:25.925 INFO  [main] - Started service [LeaseManagerReaper]
11:18:25.935 INFO  [main] - Started service [GCCollectionService]
11:18:26.155 INFO  [main] - Started service [MessageManager]
```

An interaction diagram of the startup sequence is presented below.



RMI Server with External JNDI Provider

This particular example will start a server using an external JNDI provider. In this example we illustrate this configuration using the rmiregistry JNDI provider



To start the external jndi provider change to the *bin* directory and enter the following command

```
[windows]
startreg 3031

[unix]
./startreg.sh 3031
```

This starts rmiregistry on port 3031.

The OpenJMS Server configuration is as follows.

```
<?xml version="1.0"?>
<Configuration>

  <Connectors>
    <Connector scheme="rmi">
      <ConnectionFactories>
        <QueueConnectionFactory name="JmsQueueConnectionFactory" />
        <TopicConnectionFactory name="JmsTopicConnectionFactory" />
      </ConnectionFactories>
    </Connector>
  </Connectors>

  <ServerConfiguration host="192.168.128.104" embeddedJNDI="false" />

  <JndiConfiguration>
    <property name="java.naming.factory.initial"
              value="com.sun.jndi.rmi.registry.RegistryContextFactory" />
    <property name="java.naming.provider.url"
              value="rmi:// 192.168.128.106:3031" />
  </JndiConfiguration>

  <DatabaseConfiguration>
    <RdbmsDatabaseConfiguration
      driver="oracle.jdbc.driver.OracleDriver"
      url="jdbc:oracle:oci8:@myhostname"
      user="openjms"
      password="openjms"
      retries="5"
      timeout ="2000" />
  </DatabaseConfiguration>

  <AdminConfiguration
     script="${openjms.home}\bin\startup.bat"
     config="${openjms.home}\config\openjms.xml" />

  <!-- Optional. If not specified, no destinations will be created -->
  <AdministeredDestinations>
    <AdministeredTopic name="topic1">
      <Subscriber name="sub1" />
      <Subscriber name="sub2" />
    </AdministeredTopic>

    <AdministeredQueue name="queue1" />
    <AdministeredQueue name="queue2" />
    <AdministeredQueue name="queue3" />
  </AdministeredDestinations>

</Configuration>
```

To start the server go to the *bin* directory and enter the following command.

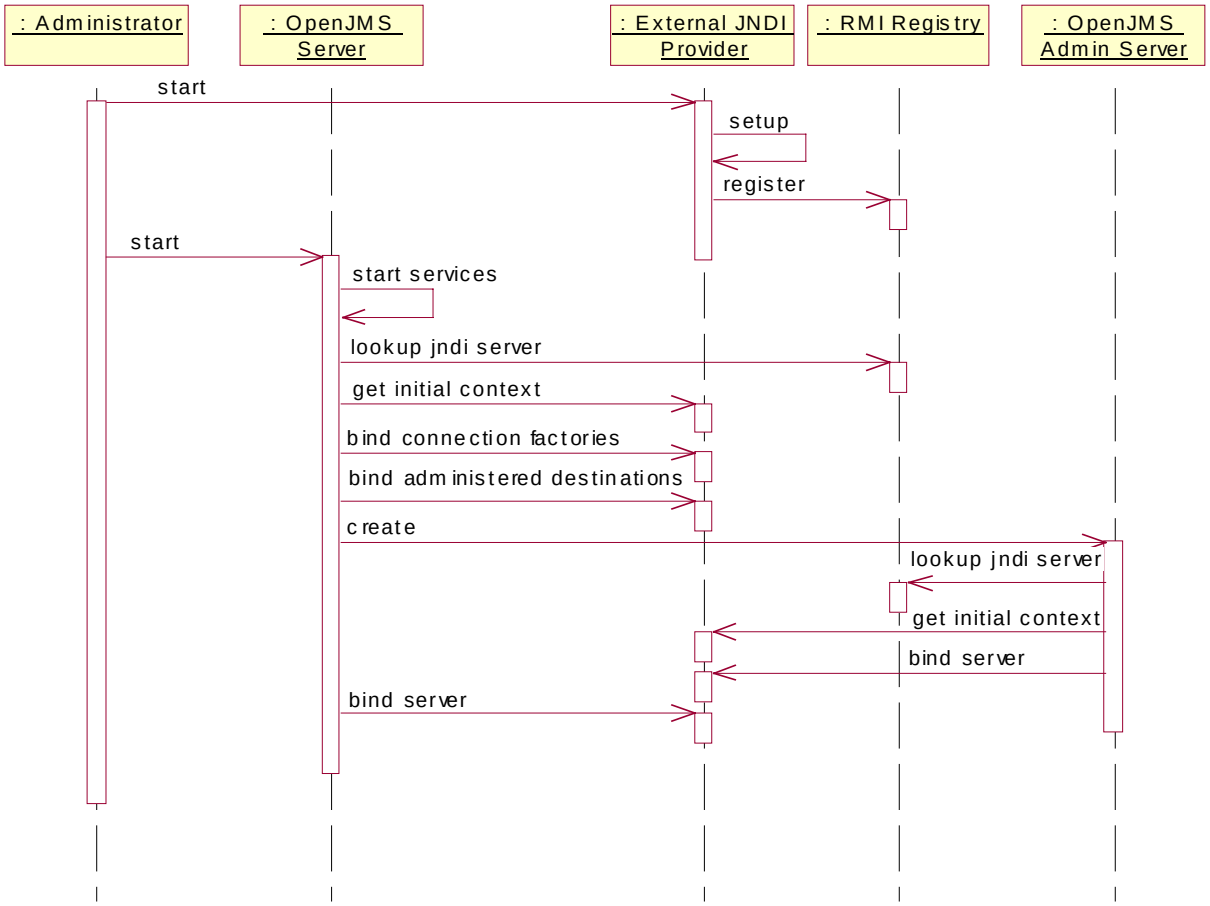
```
[windows]
startup

[unix]
./startup.sh
```

The OpenJMS server starts all the low level services and facilities. It then connects to the JNDI Server and binds any configured administered destinations and connection factories to the root context.

```
OpenJMS 0.7.2 (build 10)
Exolab Inc. (C) 1999-2002. All rights reserved. www.openjms.org
11:23:11.145 INFO  [main] - Creating the RMI-based JMS Server
```

```
11:23:11.916 INFO  [main] - Registered the service JndiServer with the registry.
11:23:11.916 INFO  [main] - Started service [ThreadPoolManager]
11:23:11.916 INFO  [main] - Started service [EventManagerThread]
11:23:12.367 INFO  [main] - Removed expired messages.
11:23:12.367 INFO  [main] - Started service [DatabaseService]
11:23:12.367 INFO  [main] - Started service [Scheduler]
11:23:12.367 INFO  [main] - Started service [LeaseManagerReaper]
11:23:12.377 INFO  [main] - Started service [GCCollectionService]
11:23:12.647 INFO  [main] - Started service [MessageManager]
11:23:12.747 INFO  [main] - JMS Server is bound to //localhost:1099/OpenJMSServer
11:23:12.788 INFO  [main] - JMS Admin Server is bound to //localhost:1099/JmsAdminServer
```

An interaction diagram of the startup sequence is presented below.



Common Problems

Database Lock Not Released

If you are using a JDBM persistent store and the OpenJMS server abnormally terminates the database lock will not be released. Subsequently trying to restart the server will yield the following error

```
18:04:22.010 INFO  [main] - Creating JNDI Server org.exolab.jms.jndi.ipc.IpcJndiServer
18:04:22.020 INFO  [main] - Started IPC JNDI Server on port 3035
18:04:22.070 INFO  [main] - Opening database: openjms.db
org.exolab.jms.server.FailedToCreateServerException: JmsServer constructor failure
javax.jms.JMSException: Cannot create persistency
 layer: Cannot acquire lock to the database openjms
        at org.exolab.jms.server.JmsServer.<init>(JmsServer.java:261)
```

```
        at org.exolab.jms.server.JmsServer.main(JmsServer.java:98)
```

The solution is to delete the database lock file, which has a *.lock* suffix. If the database name is openjms.db then the corresponding lock file will be openjms.lock.

Address in Use

This problem usually occurs when you start more than one OpenJMS server with the same configuration file. You should see the following error on the console

```
C:\projects\openjms\development\openjms\bin>startup
org.exolab.jms.server.FailedToCreateServerException: JmsServer constructor failure
java.net.BindException: Address in use: JVM_Bind
        at org.exolab.jms.server.JmsServer.<init>(JmsServer.java:261)
        at org.exolab.jms.server.JmsServer.main(JmsServer.java:98)
```

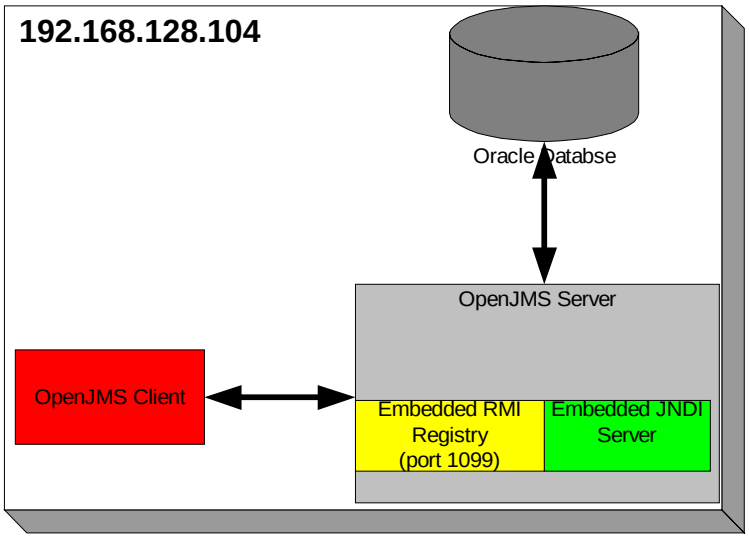
If you need to run multiple independent OpenJMS servers then you should use separate configuration files with different parameters.

Running Your First Programs

Overview

Now that you can start the openjms server, you are ready to run some of the example programs. These programs are located in the *src/examples* directory and are intended to showcase simple JMS features.

These examples assume you are using a RMI based server and the client and server are executing on the same machine.



If the client and server are executing on a different machine then you must append the following options to the command line

```
-jndiport <port number> -jndihost <host address>
```

Client Interaction Diagram

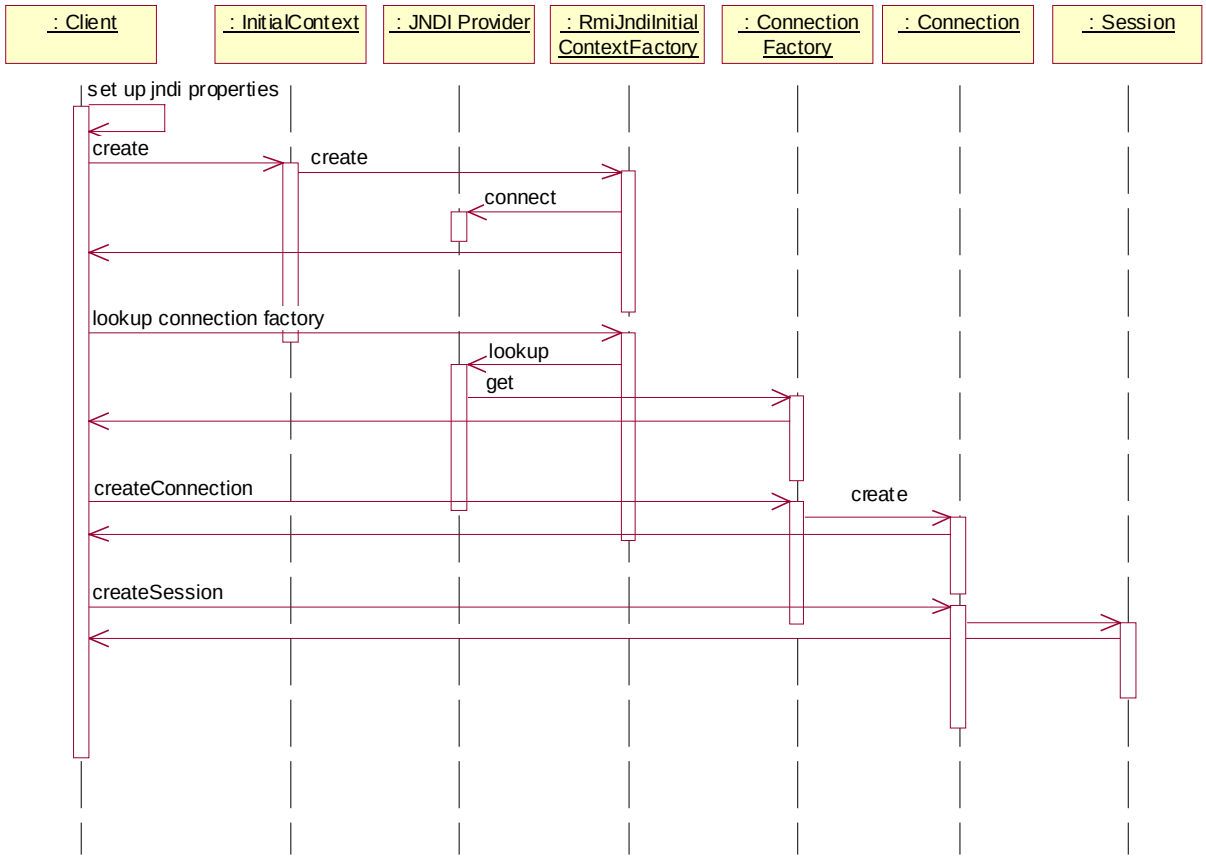
This interaction diagram describes the typical sequence of events when a client connects to the OpenJMS server and creates a session. This is true for all providers and in fact the OpenJMS examples can be run against other providers only by changing the code that connects that looks up the ConnectionFactory. By changing the bold lines you can execute the examples against other providers. Once the initial context has been retrieved, the client code is provider independent.

```
Properties props = new Properties();
props.put(Context.PROVIDER_URL, “rmi://myhost:1099/”);
props.put(Context.INITIAL_CONTEXT_FACTORY,
          org.exolab.jms.jndi.InitialContextFactory.class.getName());
Context context = new InitialContext(props);
```

The interaction diagram for all clients will typically look like this. The sequence of steps is

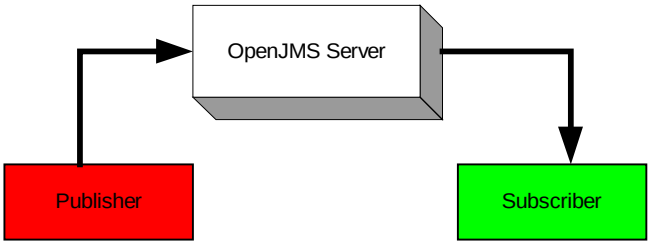
1. get the initial context
2. lookup a ConnectionFactory

3. create a Connection
4. create a Session



Single Publisher, Single Transient Subscriber

This diagram below illustrates a publisher and consumer interacting with the OpenJMS server.



The initial example uses non-persistent messages, asynchronous message delivery and transient subscribers. In this scenario, the subscriber must be started before the publisher to ensure that messages are not missed. Non-persistent messages are dropped if there are no active subscribers.

1 Start the OpenJMS server as per the *RMI Server* instructions described earlier.

2 Once the OpenJMS server is up and running change to the *bin* directory and execute *runconsumer* to start a transient subscriber, which will wait for 10 messages before existing.

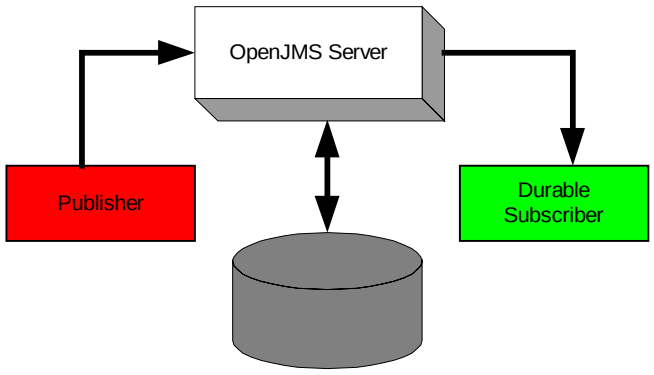
```
[windows]
runconsumer –topic topic1 –count 10

[unix]
./runconsumer.sh –topic topic1 –count 10
```

3 Now that the transient subscriber is up and running we can start the publisher using the following command line, which is executed from the *bin* directory. This will publish 10 messages and exist.

```
[windows]
runpublisher –topic topic1 –count 10.

[unix]
./runpublisher.sh –topic topic1 –count 10
```

Single Publisher, Single Durable Subscriber

This example uses a similar configuration except now it deals with persistent messages and durable subscribers. A durable subscriber is registered to receive persistent messages for a particular topic irrespective of whether the subscriber is active or inactive. Persistent messages are stored in a database.



1 Start the OpenJMS server as per the *RMI Server* instructions described earlier. Before starting the server, you must ensure that the AdministeredDestinations section of the configuration file includes the following AdministeredTopic entry.

```
<AdministeredTopic name=”topic1”>
    <Subscriber name=”sub1” />
</AdministeredTopic>
```

The above entry defines a durable subscriber *sub1* for the topic *topic1*.

2 Since we are using a durable subscriber you do not need to start the consumer before the publisher. To illustrate this we will start the publisher first.

```
[windows]
runpublisher –topic topic1 –persistent –count 10
```

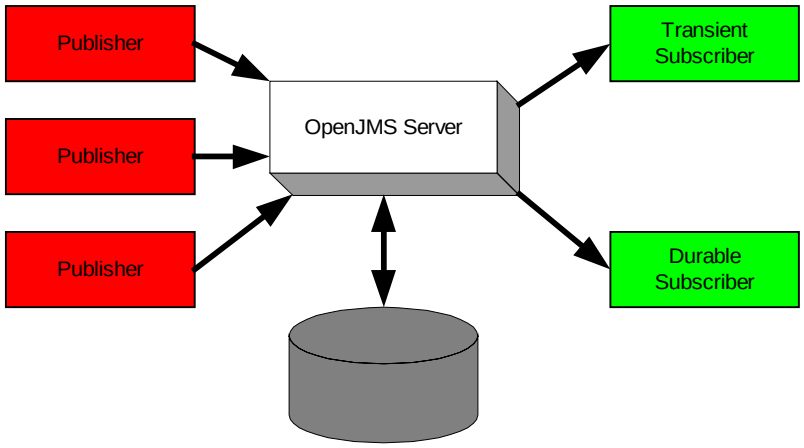
```
[unix]
./runpublisher.sh –topic topic1 –persistent –count 10
```

3 You can now start a consumer with the durable subscriber *sub1* consuming messages from *topic1*. This consumer will consume 10 messages and then exit.

```
[windows]
runconsumer –topic topic1 –persistent –name sub1 –count 10

[unix]
./runconsumer.sh –topic topic1 –persistent –name sub1 –count 10
```

Multiple Publishers, Multiple Subscribers

The diagram below depicts multiple publishers publishing messages to the same topic and multiple subscribers (transient and durable) consuming the messages



1 Start the OpenJMS server as per the *RMI Server* instructions described earlier. Before starting the server, you must ensure that the AdministeredDestinations section in the configuration file includes the following AdministeredTopic entry.

```
<AdministeredTopic name=”topic1”>
    <Subscriber name=”sub1” />
</AdministeredTopic>
```

The above entry defines a durable subscriber *sub1* for the topic *topic1*.

2 Since we are using both transient and durable subscribers, it is easier if we start them both before the publishers. In practice you can start durable subscribers until after running the publishers. Each of the three publisher will publish 10 messages under *topic1* so each consumer will be configured to receive 30 messages.

To start the transient subscriber

```
[windows]
runconsumer –topic topic1 –count 30

[unix]
./runconsumer.sh –topic topic1 –count 30
```

3 To start the durable subscriber *sub1*

```
[windows]
```

```
runconsumer –topic topic1 –persistent –name sub1 –count 30

[unix]
./runconsumer.sh –topic topic1 –persistent –name sub1 –count 30
```

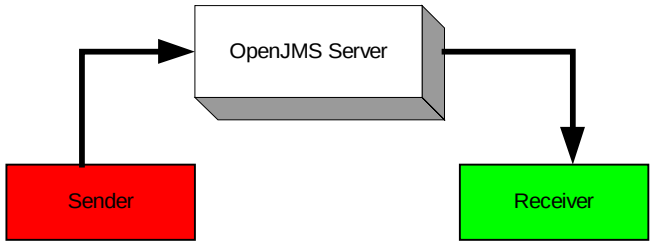
4 We can start each of the 3 publishers using by running the following command in each console. Every message will be published as persistent

```
[windows]
runpublisher –topic topic1 –persistent –count 10

[unix]
./runpublisher.sh –topic topic1 –persistent –count 10
```

5 Each of the transient and durable consumers should receive 30 messages and then exit.

Single Sender, Single Receiver

This diagram below depicts a sender and a receiver connected to the OpenJMS server and exchanging messages on the *queue1* destination



A Queue does not have a notion of transient or durable receivers.

1 Start the OpenJMS server as per the *RMI Server* instructions described earlier. Prior to starting the server you must ensure that the AdministeredDestinations section of the configuration file includes the following AdministeredQueue entry.

```
<AdministeredDestinations>
          :
          :
    <AdministeredQueue name=”queue1” />
</AdministeredDestinations>
```

The above entry defines the queue *queue1*

2 Messages send to a queue remain queues until a receiver is ready to retrieve them or the message expires. There is no need to start the receiver before the sender as in the publish-subscribe model.

To start a receiver enter the following command line in the console

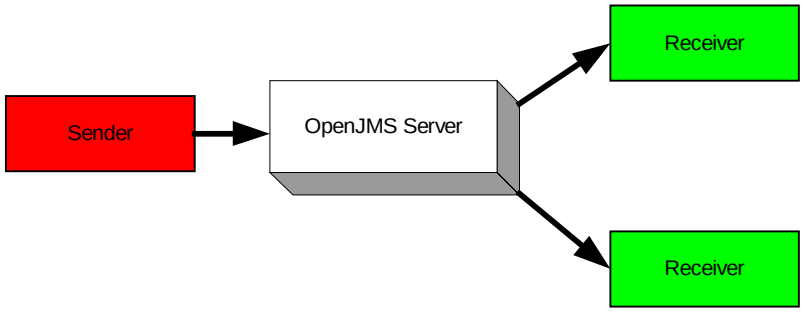
```
[windows]
runreceiver –queue queue1 –count 10

[unix]
./runreceiver.sh –queue queue1 –count 10
```

3 To execute the sender enter the following command in the sender console.

```
[windows]
runsender –queue queue1 –count 10

[unix]
./runsender.sh –queue queue1 –count 10
```

Single Senders, Multiple Receivers

This next example shows what happens when multiple receivers are attached to the same queue. Messages on a queue can only be consumed by **one** of the attached receivers unlike messages on topic destinations, which are consumed by **all** registered subscribers.



1 Start the OpenJMS server as per the *RMI Server* instructions described earlier. Prior to starting the server you must ensure that the AdministeredDestinations section in the configuration file includes the following AdministeredQueue entry.

```
<AdministeredDestinations>
          :
          :
    <AdministeredQueue name=”queue1” />
</AdministeredDestinations>
```

The above entry defines the queue *queue1*

2 We need to start two receivers we need to enter the following command line in each receiver’s console. The *count* argument denotes the number of messages each receiver will consume before existing

```
[windows]
runreceiver –queue queue1 –count 10

[unix]
./runreceiver.sh –queue queue1 –count 10
```

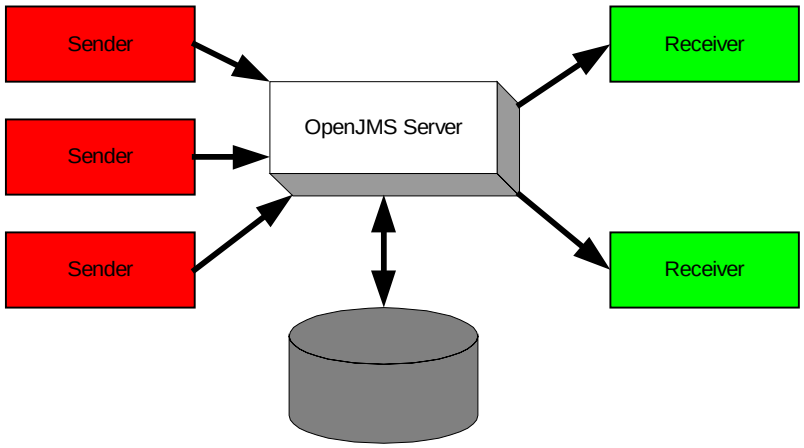
3 To execute the sender enter the following command in the sender console. The sender will publish 20 messages.

```
[windows]
runsender –queue queue1 –count 20

[unix]
./runsender.sh –queue queue1 –count 20
```

4 Each receiver should receiver 10 messages (none of which are duplicate) and exit.

Multiple Senders, Multiple Receivers

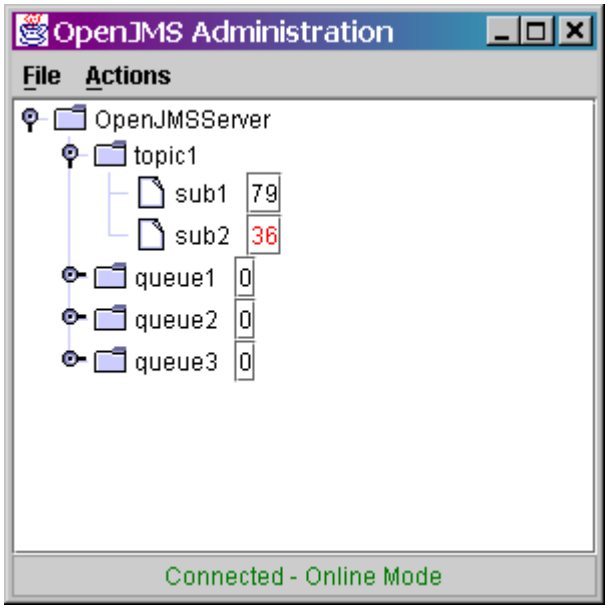
You can obviously have multiple senders publishing messages to the same queue as shown below. The previous example provides enough information to execute this scenario.



Using the Administration Tool

Introduction

OpenJMS distributes an Admin GUI tool, shown below, to facilitate the management of destinations and durable consumers.



The tools allows the administrator to

- ✓ Create and delete durable queues and topics
- ✓ Add and remove durable consumers for a topic
- ✓ Deregister active consumers
- ✓ Monitor number of messages for a queue
- ✓ Monitor number of messages per topic consumer
- ✓ Start and stop the OpenJMS server
- ✓ Purge persistent messages

Future releases of the tool will support

- ✓ Management of non-durable destinations
- ✓ Management of multiple servers from a single console

Running the Administration Tool

To start the Administration Tool go to the *bin* directory and enter the following command:

```
[windows]
admin
```

```
[unix]
./admin.sh
```

By default, the Administration Tool will use the OpenJMS configuration file:

${openjms.home}/config/openjms.xml

If the OpenJMS server was started with an alternative configuration file, using the ***–config*** command line parameter, then the Administration Tool should be passed this configuration as well. Eg, if the server was run with the command:

```
startup –config ../config/tcp_jms.xml
```

The corresponding command to start the Administration Tool would be:

```
admin –config ../config/tcp_jms.xml
```

Menu Options

The menu options, which are described below, are context sensitive.

Menu Option	Description
File->Exit	Exit the administration tool
Action->Refresh	Updates the display with the most recent information
Action->Connection->Online Connection	Connect to an active OpenJMS server.
Action->Connection->Offline Connection	This option is used to work offline and connects to the corresponding data store. This should be used for all offline maintenance
Action->Disconnect	Disconnect from a server
Action->Start OpenJMSServer	Start the OpenJMS server, using the configuration file specified during startup
Action->Stop OpenJMSServer	Stop the OpenJMS server.

Modes of Operation

The administration tool offers two modes of operation. In *online mode* the tool connects to an active OpenJMS server and in *offline mode* the tool connects directly to the OpenJMS persistent store.

Online Mode

Online mode is only possible if the OpenJMS Server is running. The administration tool makes a connection to the server, and requests all information such as topics and consumers directly from the server.

In this mode all information is available. The tool displays destinations, durable consumers and the number of outstanding messages. Additionally, the tool will highlight durable consumers that are currently active in the server.

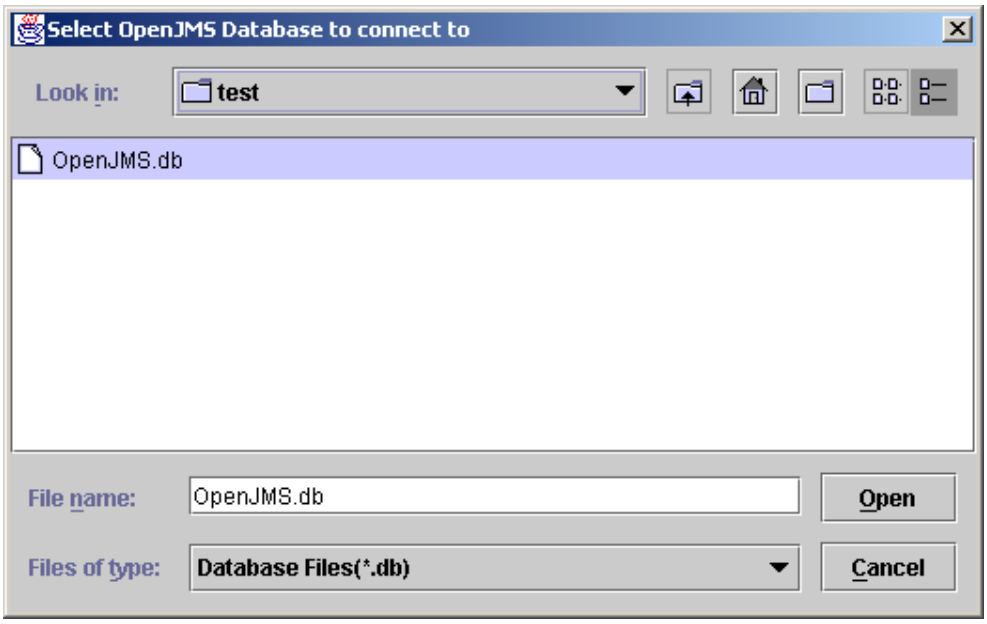
Information displayed on the console is not updated dynamically. Instead, the administrator must select the *refresh* menu option to retrieve the latest information

Offline Mode

Offline mode is a convenient way to access and update information on durable queues and topics only, when the server is not active. It also provides a means to purge processed or expired messages while the server is offline for maintenance. The *offline mode* behavior differs depending on the underlying persistent store being used.

If the system is using an RDBMS database, the appropriate configuration information, for making a connection to the database, is retrieved from the specified configuration file. **The administrator must ensure that the OpenJMS server is not active while using this mode since could compromise the integrity of the system.**

If the system is using the JDBM database a file selector is displayed to select the corresponding database file.



Selecting the database file and pressing *open* will initiate a connection to the database. The operation will fail if the selected file is not a valid OpenJMS database. Check the OpenJMS server configuration file to locate the name of the file.

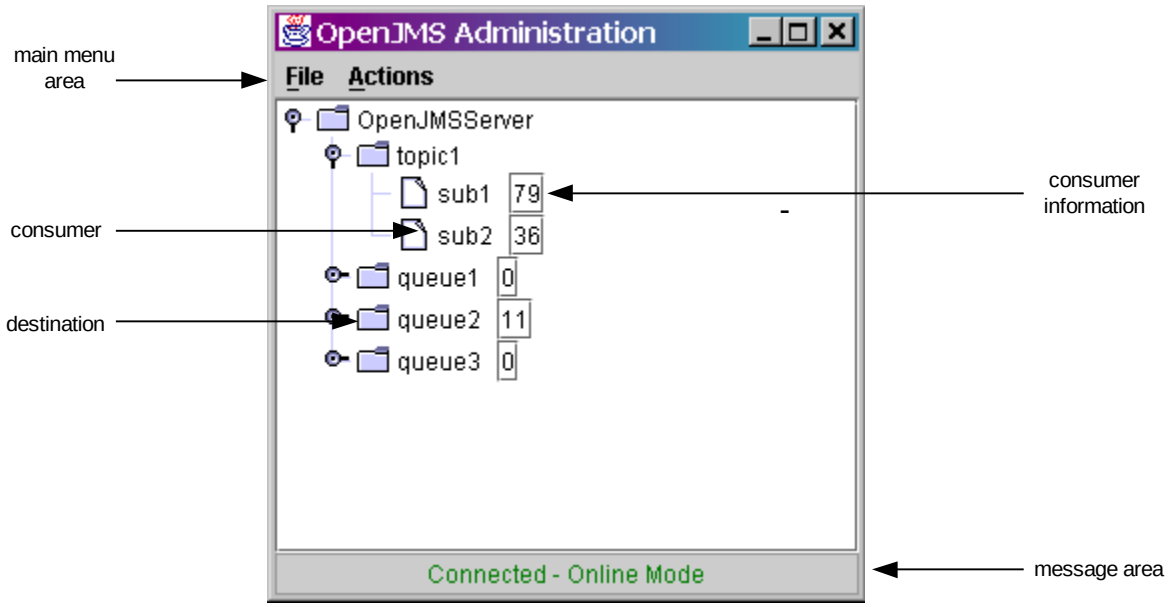
The administrator can also create a new database file simply by entering the name, without the *.db* suffix. To use the new database file simply modify the entry in the server configuration file

```
<DatabaseConfiguration
  garbageCollectionInterval="180"
  garbageCollectionBlockSize="500"
  garbageCollectionThreadPriority="5">
  <JdbmDatabaseConfiguration name="${openjms.home}/openjms.db" />
</DatabaseConfiguration>
```

A file locking mechanism is used to lock the database and prevent simultaneous access by the admin tool and the OpenJMS server.

Understanding the Display

The figure below outlines the various elements of the display. The top-level node in the hierarchy is essentially the OpenJMS server or persistent store. Inside the root node there is a node for each persistent destination. If the destination is a topic then a node will exist for each durable consumer. Nodes can be expanded or collapsed as necessary in the usual manner by selecting the node handle or double clicking on the node itself.



The text information displayed next to some nodes denotes the number of outstanding messages for a particular queue or a durable subscriber. If the text is displayed in red it indicates this consumer is currently active

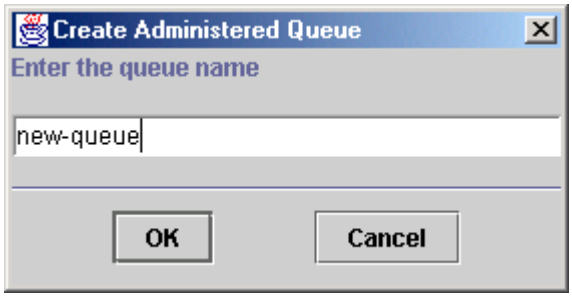
Context Sensitive Commands

Context sensitive commands are available for the different nodes in the tree. Pressing the right mouse button on any node will display the available commands for that object.

OpenJMS Server Node

Add Queue

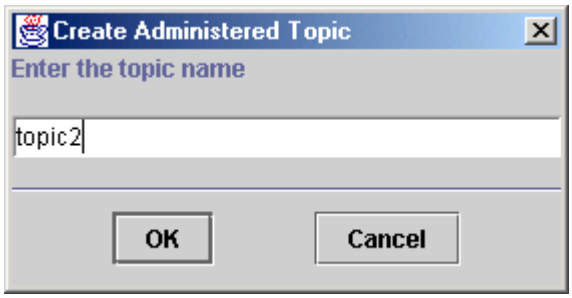
Select the *OpenJMSServer* node, right-click the mouse button and choose *Add Queue*. When the following dialog box appears, enter the name of the queue and press *OK*.



The name of the queue must be unique and any attempt to create an existing queue will display an error message.

Add Topic

Select the *OpenJMSServer* node, right-click the mouse button and choose *Add Topic*. When the following dialog box appears, enter the name of the topic and press *OK*.



The name of the topic must be unique and any attempt to create an existing topic will display an error message.

Purge Messages

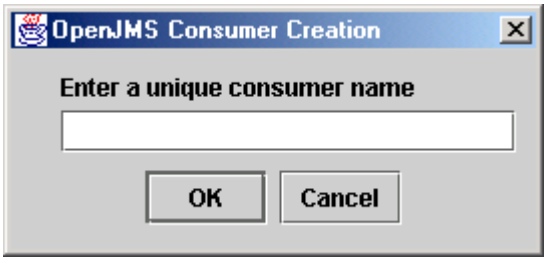
Select the *OpenJMSServer* node, right-click the mouse button and choose *Purge Messages*. A dialog box will be displayed asking the user to confirm the deletion of all processed messages. Pressing *Yes* will initiate the process.

This can be a very CPU intensive operation, especially if there are many messages to process.

Topic Node

Add Consumer

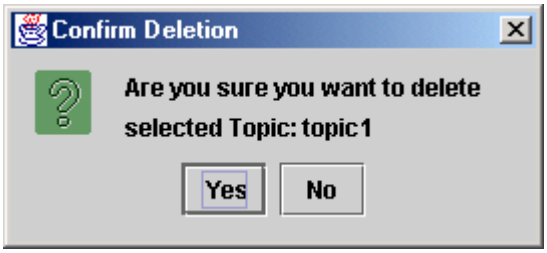
Select the corresponding topic node, right-click the mouse button and choose *Add Consumer*. When the following dialog box appears enter the unique durable consumer name and press *OK*.



The operation will fail if the consumer name is not unique **across all topics**.

Delete Topic

Select the corresponding topic node, right-click the mouse button and choose *Delete Topic.* A confirmation dialog box will be display asking the user to confirm the action. Pressing *Yes* will initiate the request.

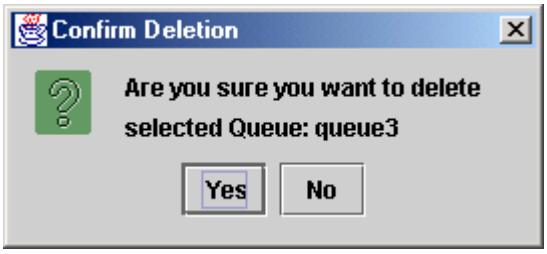


Deleting a topic will automatically delete all registered durable consumers and all associated persistent messages.

Queue Node

Delete Queue

Select the corresponding queue node, right-click the mouse button and choose *Delete Queue.* A confirmation dialog box will be display asking the user to confirm the action. Pressing *Yes* will initiate the request.



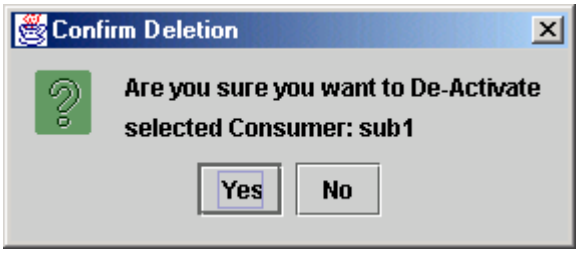
Deleting a queue will automatically delete all associated persistent messages.

Consumer Node

Deactivate Consumer

When a durable consumer is active with the server the corresponding text, on the display, will be highlighted in red. Once a durable consumer is active it prevents other clients from connecting using that consumer name. The administrator can use this option to deactivate the durable consumer within the server.

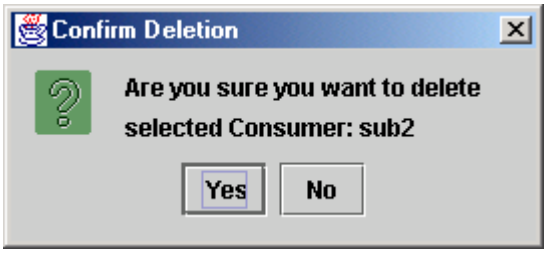
To deactivate a consumer select the consumer node, right click the mouse button and choose *Deactivate Consumer*. A confirmation dialog box will be displayed asking the user to confirm the action. Pressing *Yes* will deactivate the consumer.



The corresponding text, on the display, should now be in black indicating that the consumer has been deactivated.

Delete Consumer

To deactivate a consumer select the consumer node, right click the mouse button and choose *Delete Consumer*. A confirmation dialog box will be displayed asking the user to confirm the action. Pressing *Yes* will delete the consumer and remove all associated persistent messages from the database.



Using the OpenJMS Admin API

Since the JMS specification does not cover administration features of a messaging system most providers support such features through a proprietary interface. OpenJMS has a proprietary administration API, which can be used to manage the lifecycle of destinations, subscribers and messages.

Features

The table below provides a brief description of the features available through the administration API[11].

Feature	Description
getDurableConsumerMessageCount	Return the number of unprocessed messages for a particular consumer
getQueueMessageCount	Return the number of unprocessed messages on a particular queue
addDurableConsumer	Add a durable consumer for a particular topic destination.
removeDurableConsumer	Remove a particular durable consumer
durableConsumerExists	Check if the specified durable consumer exists
getDurableConsumers	Return a list of durable consumer names for a particular topic destination
unregisterConsumer	Deactivate an active durable consumer
isConnected	Check to see if the given consumer is currently active in the server
getAllDestinations	Return a list of all persistent destinations
addDestination	Add a destination with a particular name
removeDestination	Destroy a destination and all associated messages and consumers.
stopServer	Terminate the JMS Server. If it is running as a standalone application then exit the application. If is running as an embedded application then just terminate the thread
purgeMessages	Purge all processed persistent messages from the database.

[11] Consult the org.exolab.jms.admin.JmsAdmin JavaDoc for the latest set of features

Using the API

Accessing the OpenJMS Admin API

OpenJMS uses the specified JNDI provider to register the JMSAdminServerIfc interface. The URL that is used to bind to the JNDI provider is displayed on the console when the server is started.

```
10:50:35.283 INFO  [main] - Started service [MessageManager]
10:50:35.293 INFO  [main] - JMS Server is bound to //localhost:1099/OpenJMSServer
10:50:35.333 INFO  [main] - JMS Admin Server is bound to //localhost:1099/JmsAdminServer
```

The example, shown below, illustrates how to get access to the JMSAdminServerIfc to an RMI OpenJMS server.

```
import java.util.Iterator;
import java.util.Vector;

// jms
import javax.jms.Destination;
import javax.jms.Queue;
import javax.jms.Topic;

// openjms
import org.exolab.jms.administration.AdminConnectionFactory;
import org.exolab.jms.administration.JmsAdminServerIfc;
import org.exolab.jms.util.CommandLine;


/**
 * Example demonstrating the administration interface
 */
public class Admin {
    public static void main(String[] args) throws Exception {
        String url = “rmi://localhost:1099/”;
        JmsAdminServerIfc admin = AdminConnectionFactory.create(url);
        if (admin != null) {
            // we can now administer the server
        } else {
            System.err.println(“Failed to get access to the admin interface”);
        }
    }
}
```

Using the OpenJMS Admin API

Once you have access to JmsAdminServerIfc interface you can call any method as illustrated below

```
/**
 * Example demonstrating the administration interface
 */
public class Admin {
    public static void main(String[] args) throws Exception {
        String url = “rmi: //localhost:1099/”;
        JmsAdminServerIfc admin = AdminConnectionFactory.create(url);
        if (admin != null) {
            Vector destinations = admin.getAllDestinations();
            Iterator iter = destinations.iterator();
            while (iter.hasNext()) {
                Destination destination = (Destination) iter.next();
                if (destination instanceof Queue) {
```

```
                    Queue queue = (Queue) destination;
                    System.out.println(queue.getQueueName() + ": Queue");
                } else {
                    Topic topic = (Topic) destination;
                    System.out.println(topic.getTopicName() + ": Topic");
                }
            }
            admin.close();
        }
    }
}
```

The e*xolabcore* library

The OpenJMS distribution uses the *exolabcore-version.jar* library. This source code for this library is part of the openjms CVS repository and can be checked out using the following CVS command

```
cvs –d:pserver:anoncvs@virtuals.intalio.com:/cvs/openjms login
password=anoncvs

cvs –d:pserver:anoncvs@virtuals.intalio.com:/cvs/openjms exolabcore
```

OpenJMS over SSL

OpenJMS SSL is only available for OpenJMS IPC and has been implemented using Sun Microsystems' JSSE (http://java.sun.com/products/jsse) for the initial release. JSSE 1.0.2 is required for JDK 1.2 and 1.3. As of JDK1.4 JSSE is part of the standard distribution.

Follow the JSSE installation instructions to install the 1.0.2 product. Additional information is also available JSSE 1.0.2 Guide.

Once you have downloaded and setup JSSE, a trustore (security certification) must be generated. A sample one is provided in the OpenJMS bin directory (*cacerts*) but you can create your own use the ***keytool*** utility that ships with JSSE.

```
keytool -genkey -keystore filename -keyalg rsa
where filename is the name of the trustore
```

Next you must modify the OpenJMS Server configuration file. In particular you must change the value of the *scheme* attribute in the *Connector* element.

```
  <Connectors>
    <Connector scheme="tcps">
      <ConnectionFactories>
        <QueueConnectionFactory name="JmsQueueConnectionFactory" />
        <TopicConnectionFactory name="JmsTopicConnectionFactory" />
      </ConnectionFactories>
    </Connector>
  </Connectors>
```

If the JSSE JAR files are stored in the *lib/ext* directory, then they will be automatically picked up by the JVM. If this is not the case then the jar files should either be accessible by setting the CLASSPATH in command line or the environment classpath variable. The required libraries are

Library	Version	Description
jcert.jar	1.0.2	
jnet.jar	1.0.2	
jsse.jar	1.0.2	

Finally you must create (or modify) the *setenv.[bat|sh]* (see Environment Scripts) to pass some properties across to JSSE via the JAVA_OPTS environment variable.. The following configures SSL using the sample trust store:

```
set JAVA_OPTS=-Djavax.net.ssl.trustStore=cacerts -Djavax.net.ssl.keyStore=cacerts \
-Djavax.net.ssl.keyStoreType=jks -Djavax.net.ssl.keyStorePassword=ab1234
```

If the system property: *javax.net.ssl.trustStore* is defined, then the *TrustManagerFactory* attempts to find a file using the filename specified by that system property, and uses that file for the KeyStore. If that system property is not specified, and the file *<java-home>/lib/security/jssecacerts* exists, it will be used. Otherwise, it will use the file *<java-home>/lib/security/cacerts* if it exists.

You can debug the JSSE component by setting the system property ***-Djavax.net.debug=ssl*** to turn.

Clients connecting to the OpenJMS SSL server must specify the trustore property ***-Djavax.net.ssl.trustStore=filename*** otherwise the server will reject the connection.

Typical Configurations

RMI, Oracle, Embedded JNDI Server

```
<?xml version="1.0"?>
<Configuration>

  <Connectors>
    <Connector scheme="rmi">
      <ConnectionFactories>
        <QueueConnectionFactory name="JmsQueueConnectionFactory" />
        <TopicConnectionFactory name="JmsTopicConnectionFactory" />
      </ConnectionFactories>
    </Connector>
  </Connectors>

  <DatabaseConfiguration>
    <RdbmsDatabaseConfiguration
      driver="oracle.jdbc.driver.OracleDriver"
      url="jdbc:oracle:oci8:@openjms"
      user="openjms"
      password="openjms" />
  </DatabaseConfiguration>

  <AdminConfiguration
     script="${openjms.home}\bin\startup.bat"
     config="${openjms.home}\config\rmi_jms_oracle.xml" />

  <!-- Optional. If not specified, no destinations will be created -->
  <AdministeredDestinations>
    <AdministeredTopic name="topic1">
      <Subscriber name="sub1" />
      <Subscriber name="sub2" />
    </AdministeredTopic>

    <AdministeredQueue name="queue1" />
    <AdministeredQueue name="queue2" />
    <AdministeredQueue name="queue3" />
  </AdministeredDestinations>

</Configuration>
```

TCP, MySQL, External JNDI Server

```
<?xml version="1.0"?>
<Configuration>

  <DatabaseConfiguration>
    <RdbmsDatabaseConfiguration
      driver="org.gjt.mm.mysql.Driver"
      url="jdbc:mysql://localhost/test"
      user="openjms"
      password="openjms" />

  <ServerConfiguration host="localhost" embeddedJNDI="false" />

  <JndiConfiguration>
    <property name="java.naming.factory.initial"
              value="com.sun.jndi.rmi.registry.RegistryContextFactory" />
    <property name="java.naming.provider.url"
              value="rmi://localhost:1099" />
  </JndiConfiguration>
  <AdminConfiguration
     script="${openjms.home}\bin\startup.bat"
```

```
    config="${openjms.home}\config\openjms.xml" />

  <!-- Optional. If not specified, no destinations will be created -->
  <AdministeredDestinations>
    <AdministeredTopic name="topic1">
      <Subscriber name="sub1" />
      <Subscriber name="sub2" />
    </AdministeredTopic>

    <AdministeredQueue name="queue1" />
    <AdministeredQueue name="queue2" />
    <AdministeredQueue name="queue3" />
  </AdministeredDestinations>

</Configuration>
```

RMI, JDBM, External JNDI Server

```
<?xml version="1.0"?>
<Configuration>

  <Connectors>
    <Connector scheme="rmi">
      <ConnectionFactories>
        <QueueConnectionFactory name="JmsQueueConnectionFactory" />
        <TopicConnectionFactory name="JmsTopicConnectionFactory" />
      </ConnectionFactories>
    </Connector>
  </Connectors>

  <ServerConfiguration host="localhost" embeddedJNDI="false" />

  <JndiConfiguration>
    <property name="java.naming.factory.initial"
              value="com.sun.jndi.rmi.registry.RegistryContextFactory" />
    <property name="java.naming.provider.url"
              value="rmi://localhost:3031" />
  </JndiConfiguration>

  <DatabaseConfiguration>
    <JdbmDatabaseConfiguration name="openjms.db" />
  </DatabaseConfiguration>

  <AdminConfiguration
     script="${openjms.home}\bin\startup.bat"
     config="${openjms.home}\config\openjms.xml" />

  <!-- Optional. If not specified, no destinations will be created -->
  <AdministeredDestinations>
    <AdministeredTopic name="topic1">
      <Subscriber name="sub1" />
      <Subscriber name="sub2" />
    </AdministeredTopic>

    <AdministeredQueue name="queue1" />
    <AdministeredQueue name="queue2" />
    <AdministeredQueue name="queue3" />
  </AdministeredDestinations>

</Configuration>
```

OpenJMS Performance

This section is an attempt to crudely quantify the performance of OpenJMS for both persistent and non-persistent messages against different databases. The test has a single publisher connecting to the server and publishing 10,000 messages

The tests were conducted on a PIII 800MHz with 256MB memory.

	publishing 10000 non-persistent messages	publishing 10000 persistent messages
Oracle 8i	500 msgs/second	64 msgs/second
MySQL	500 msgs/second	152 msgs/second
JDBM	526 msgs/second	1 msg/second[12]

[12] only published 1000 messages

Appendix A: OpenJMS with Jakarta Tomcat

Introduction

This application note illustrates how to integrate OpenJMS (http://openjms.sourceforge.net/) with the Jakarta Tomcat (http://jakarta.apache.org/tomcat/index.html). It deals exclusively with Tomcat v4 (or Catalina), which is the next generation servlet/jsp container and OpenJMS v.0.7.

The article assumes that the reader is familiar with both OpenJMS and Tomcat and does not describe how to install or use either product. Therefore, to allow you to work through this note we recommend that you install and familiarize yourself with the products by visiting their respective web sites.

Dependencies

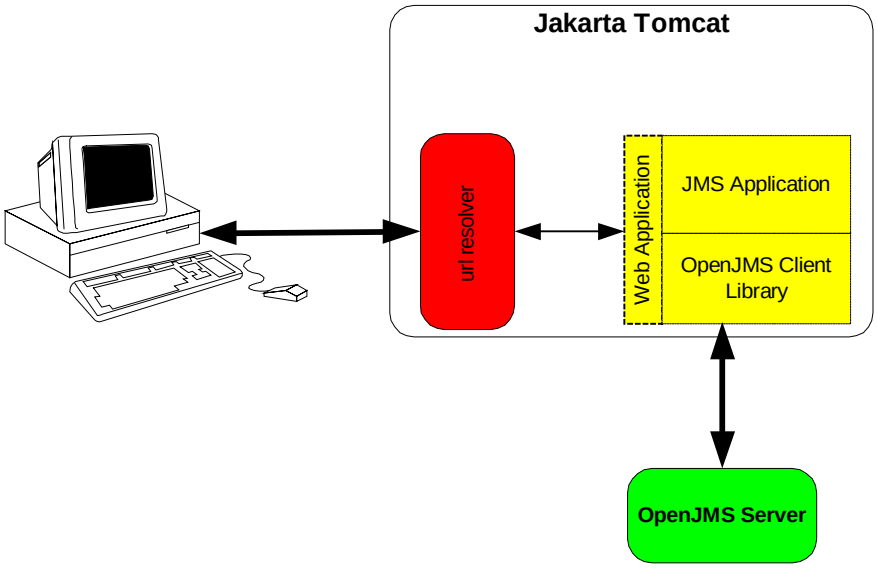
The examples in this application note has been tested with

- ✓ OpenJMS v0.7.5, which conforms to v1.0.2 of the specification (http://java.sun.com/products/jms) and is available at http://sourceforge.net/project/showfiles.php?group_id=54559
- ✓ Jakarta Tomcat v4.1.18, which conforms to v2.3 of the Servlet specification (http://java.sun.com/products/servlets) and v1.2 of the Java Server Pages (JSP) specification (http://java.sun.com/products/jsp) and is available at http://jakarta.apache.org/builds/jakarta-tomcat-4.0/release/v4.1.18/

Context Diagram

The diagram below illustrates the relationship between the various Tomcat and OpenJMS components. The OpenJMS Web Application consists of JMS client code talking to the OpenJMS Client Library. The application is deployed within the Tomcat container.

In this configuration the OpenJMS server is executing outside the Tomcat container and may even be executing on another machine on a different network.

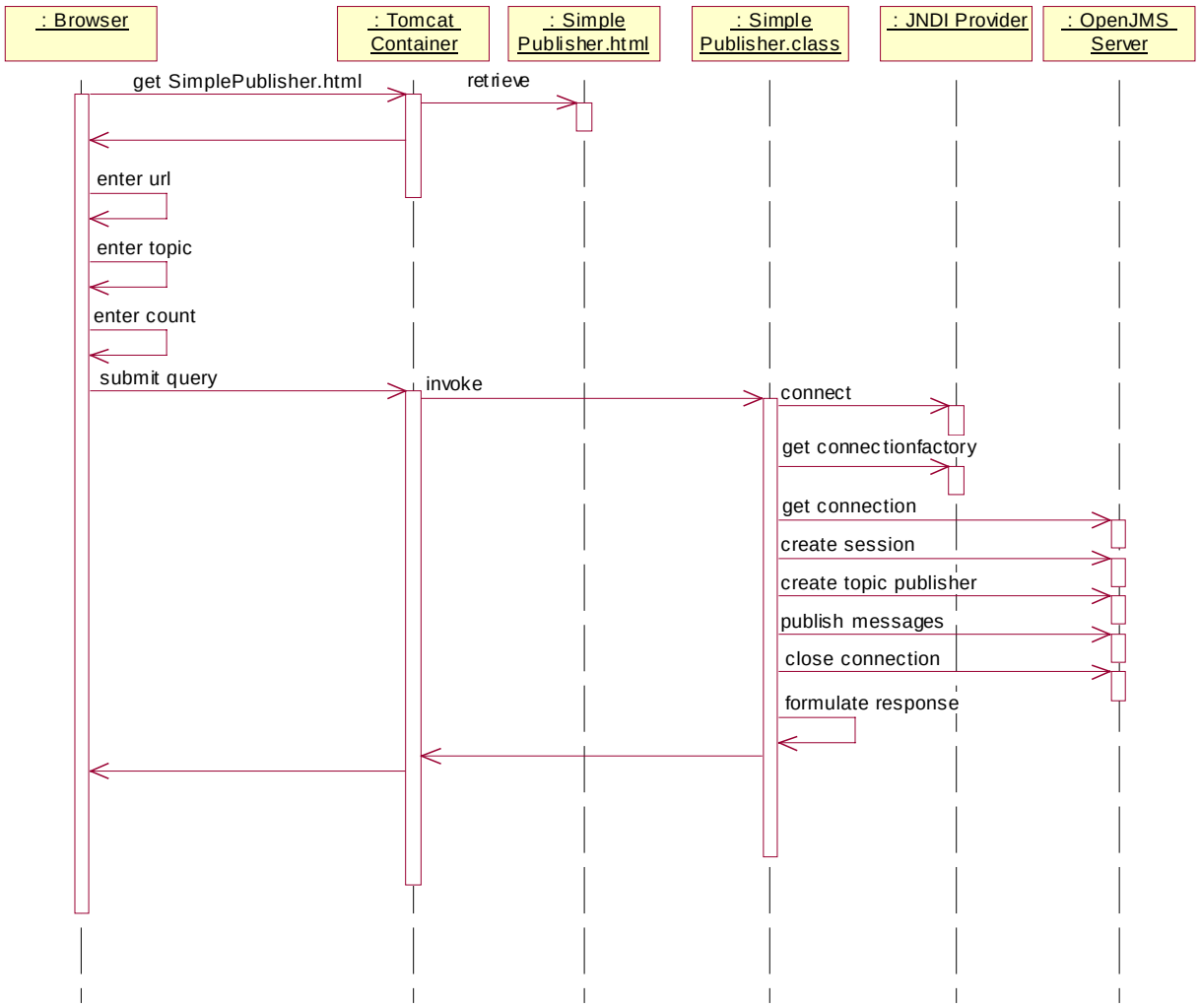


The user can access the OpenJMS application by entering the corresponding URL in the browser or application. Tomcat will resolve the URL and redirect it to the corresponding servlet, which will execute and then return a response back to the user.

In the example that follows, Tomcat is used to serve both dynamic and static web pages therefore functioning as a typical web server in addition to a servlet and JSP container.

Writing an OpenJMS Web Application

This section describes how to write and package a trivial OpenJMS Web Application, which simply connects to the OpenJMS Server and publishes messages. The interaction diagram, illustrated below, depicts the sequence of events between the various components.



The table, below, provides a brief description of each of the components in the interaction diagram. Further details are provided in subsequent sections.

Component	Description
Browser	Represents the client browser making the requests to Tomcat.
Tomcat Container	The Servlet/JSP container
SimplePublisher.html	A static HTML page containing a form, which gathers information to publish messages to the OpenJMS Server
SimplePublisher.class	A servlet class that interacts with the OpenJMS server
JNDI Provider	Used to bind and lookup OpenJMS connection factories
OpenJMS Server	A JMS v1.0.2 compliant server

The sequence of events is as follows. The user will enter a URL to access the *SimplePublisher.html* page

Once the page is displayed in the browser, the user will complete the *url, topic* and *count* fields.

Field	Description
url	This denotes the URL of the JNDI Server, which holds the OpenJMS ConnectionFactories. The ConnectionFactories are used to bootstrap a connection to the OpenJMS Server. The form of the URL depends on the configuration of the OpenJMS Server.
topic	This is the name of the destination that messages will be published under. The messages are in the form “publish message X”, where X is the message number.
count	The number of messages to publish.

When the user presses the *Submit Query* button the information in the form will be send across to Tomcat and used to invoke the *SimplePublisher* servlet.

The servlet will connect to the OpenJMS server, create a *TopicPublisher* using the *topic* form parameter, publish *count* messages and then close the connection. On successful completion, the user should see the following response in their browser

Published 20 messages on topic jima to url rmi://localhost:1099/JndiServer.

The application itself is not useful but it does provide all the information necessary to integrate the two components.

SimplePublisher.html

The HTML code includes the form, which gathers the user information and submits a request to the SimplePublisher servlet.

```
<html>
  <title>
    Simple Publisher
  </title>

  <body>
    <form method="get" action="../servlet/SimplePublisher">
      <table cols="2" cellspacing="10">
        <tr>
          <td>url</td>
```

```
          <td><input type="text" name="url" size="80"></td>
        </tr>

        <tr>
          <td>topic</td>
          <td><input type="test" name="topic" size="30"></td>
        </tr>

        <tr>
          <td>count</td>
          <td><input type="text" name="count" size="5"></td>
        </tr>
      </table>
      <input type="submit">
    </form>
  </body>
</html>
```

SimplePublisher.java

Below is the servlet code for this example. It uses the information passed across by the user to retrieve a ConnectionFactory from the JNDI provider, connect to the OpenJMS server, create a publisher and publish messages to it. Finally, it closes the connection and returns a response to the user.

If a problem is encountered, while this request is being processed, a ServletException is returned to the user.

```
import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.jms.*;
import javax.naming.*;

public class SimplePublisher extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException {

        String url = request.getParameter("url");
        String topic_name = request.getParameter("topic");
        String count_str = request.getParameter("count");
        int count = 0;
        try {
            count = Integer.parseInt(count_str);
        } catch (Exception exception) {
            throw new ServletException("Error in converting count " + exception);
        }

        try {
            String factory_name = "JmsTopicConnectionFactory";
            Properties props = new Properties();

            props.put(Context.INITIAL_CONTEXT_FACTORY,
                      org.exolab.jms.jndi. InitialContextFactory.class.getName());
            props.put(Context.PROVIDER_URL, url);

            // lookup the connection factory from the context
            Context context = new InitialContext(props);
            TopicConnectionFactory factory = (TopicConnectionFactory)
                context.lookup(factory_name);

            // create the connection and session
            TopicConnection connection = factory.createTopicConnection();
            TopicSession session =  connection.createTopicSession(
```

```
                false, Session.AUTO_ACKNOWLEDGE);
            Topic topic = session.createTopic(topic_name);

            // publish messages
            TopicPublisher publisher = session.createPublisher(topic);
            for (int index = 0; index < count; index++) {
                publisher.publish(session.createTextMessage("publish message " + count));
            }

            // close the connction
            connection.close();
        } catch (Exception exception) {
            exception.printStackTrace();
            throw new ServletException("Exception in processign request " + exception);
        }

        // formulate response
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>OpenJMS Response</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("Published " + count + " messages on topic " + topic_name +
                    " to url " + url + ".");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Compiling and Packaging

To compile the servlet you need the following libraries

Library	Description
openjms-client-0.7.5.jar	The core openjms client library
openjms-rmi-0.7.5.jar	Additional classes for rmi based OpenJMS server
exolabcore-0.3.4.jar	Core Exolab library
jms_1.0.2a.jar	Java Message Service (JMS) API

In addition, before we package the web application we must create the *web.xml* descriptor file, which identifies the servlet and its corresponding URL mapping.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>A Simple Publisher Application</display-name>
  <context-param>
    <param-name>Webmaster</param-name>
    <param-value>jima@comware.com.au</param-value>
  </context-param>
  <servlet>
    <servlet-name>SimplePublisher</servlet-name>
    <servlet-class>SimplePublisher</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>SimplePublisher</servlet-name>
```

```
    <url-pattern>/openjms/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Finally, we must package all the above components in a **W**eb **A**pplication A**r**chive (war), which is identical, in format, to JAR files. The content of *openjms.war*, which is shown below, complies with the structure identified in the Servlet specifications.

```
     0 Wed Oct 03 16:18:36 GMT+11:00 2001 META-INF/
     0 Wed Oct 03 16:18:36 GMT+11:00 2001 WEB-INF/
     0 Wed Oct 03 16:18:36 GMT+11:00 2001 WEB-INF/classes/
     0 Wed Oct 03 16:18:38 GMT+11:00 2001 WEB-INF/lib/
     0 Wed Oct 03 16:18:36 GMT+11:00 2001 servlets/
     0 Wed Oct 03 16:18:36 GMT+11:00 2001 build/
     0 Wed Oct 03 16:18:36 GMT+11:00 2001 dist/
   667 Wed Oct 03 16:18:36 GMT+11:00 2001 WEB-INF/web.xml
  3752 Wed Oct 03 16:18:36 GMT+11:00 2001 WEB-INF/classes/SimplePublisher.java
  4279 Wed Oct 03 16:18:38 GMT+11:00 2001 WEB-INF/classes/SimplePublisher.class
268061 Wed Oct 03 16:18:38 GMT+11:00 2001 WEB-INF/lib/openjms-client-0.7.jar
 44025 Wed Oct 03 16:18:38 GMT+11:00 2001 WEB-INF/lib/openjms-rmi-0.7.jar
124524 Wed Oct 03 16:18:38 GMT+11:00 2001 WEB-INF/lib/exolabcore-0.3.1.jar
 27724 Wed Oct 03 16:18:38 GMT+11:00 2001 WEB-INF/lib/jms_1.0.2a.jar
   757 Wed Oct 03 16:18:36 GMT+11:00 2001 servlets/SimplePublisher.html
    51 Wed Oct 03 16:18:36 GMT+11:00 2001 META-INF/MANIFEST.MF
```

The archive contains all the libraries, the servlet and associated HTML files.

Deploying the OpenJMS Web Application

We must follow the following steps to deploy the archive, generated above, in Tomcat.

- ✓ copy and unpack the archive in *Tomcat's webapps* directory.
- ✓ modify Tomcat's *server.xml* file and add a new Context.
- ✓ restart Tomcat
- ✓ start OpenJMS server

Unpack the Web Application

Locate the *webapps* directory in your Tomcat installation, create the *openjms* subdirectory and unpack the archive.

The directory structure should look be similar to this.

```
webapps/openjms
webapps/openjms/META-INF
webapps/openjms/META-INF/MANIFEST.MF
webapps/openjms/WEB-INF
webapps/openjms/WEB-INF/classes
webapps/openjms/WEB-INF/classes/SimplePublisher.java
webapps/openjms/WEB-INF/classes/SimplePublisher.class
webapps/openjms/WEB-INF/lib
webapps/openjms/WEB-INF/lib/openjms-client-0.7.jar
webapps/openjms/WEB-INF/lib/openjms-rmi-0.7.jar
webapps/openjms/WEB-INF/lib/exolabcore-0.3.1.jar
webapps/openjms/WEB-INF/lib/jms_1.0.2a.jar
webapps/openjms/WEB-INF/web.xml
webapps/openjms/servlets
webapps/openjms/servlets/SimplePublisher.html
webapps/openjms/build
```

```
webapps/openjms/dist
```

Modify the *server.xml* configuration file

In Tomcat's server configuration file, add a new context that defines the openjms web application. The configuration redirects URLs in the form of *http://localhost:2222/openjms/…*to the web application we have just deployed.

```
:
:
:
<Server port="2220" shutdown="SHUTDOWN" debug="0">
  <Service name="Tomcat-Standalone">
    <!-- Define a non-SSL HTTP/1.1 Connector on port 8080 -->
    <Connector className="org.apache.catalina.connector.http.HttpConnector"
               port="2222" minProcessors="5" maxProcessors="75"
               enableLookups="true" redirectPort="2223"
               acceptCount="10" debug="0" connectionTimeout="60000"/>
    <Engine name="Standalone" defaultHost="localhost" debug="0">
      <Logger className="org.apache.catalina.logger.FileLogger"
              prefix="catalina_log." suffix=".txt" timestamp="true"/>
      <Realm className="org.apache.catalina.realm.MemoryRealm" />
      <Host name="localhost" debug="0" appBase="webapps" unpackWARs="true">
        <!-- Tomcat Root Context -->
        <Context path="" docBase="ROOT" debug="0"/>
              :
              :
        <!-- SimplePublisher examples -->
        <Context path="openjms" docBase="openjms" debug="0"
                 reloadable="true">
          <Logger className="org.apache.catalina.logger.FileLogger"
                     prefix="localhost_examples_log." suffix=".txt"
                timestamp="true"/>
        </Context>
      </Host>
    </Engine>
  </Service>
</Server>
```

Restarting Tomcat

Restart Tomcat, using either c*atalina* or *tomcat* scripts files located in the *bin* directory. If server comes up without any problems, the following will be displayed on the console.

```
Starting service Tomcat-Standalone
Apache Tomcat/4.0-b8-dev
```

Starting the OpenJMS Server

For simplicity use the out-of-the-box RMI/JDBM configuration, openjms.xml for the server. Change to the *bin* directory of the OpenJMS installation and enter the following command line.

```
startup.bat
```

If the server has successfully started then the following output should appear in the console.

```
OpenJMS 0.7.5 (build 17)
Exolab Inc. (C) 1999-2003. All rights reserved. www.openjms.org
23:22:14.303 INFO  [main] - Instantiated the RmiRegistryService service on port 1099
23:22:14.303 INFO  [main] - Embedded RMI Registry running on port 1099
23:22:14.673 INFO  [main] - Registered the service JndiServer with the registry.

23:22:14.673 INFO  [main] - Started service [RmiRegistryService]
```

```
23:22:14.673 INFO  [main] - Started service [ThreadPoolManager]
23:22:14.673 INFO  [main] - Started service [EventManagerThread]
23:22:14.874 INFO  [main] - Removed expired messages.
23:22:14.874 INFO  [main] - Started service [DatabaseService]
23:22:14.874 INFO  [main] - Started service [Scheduler]
23:22:14.874 INFO  [main] - Started service [LeaseManagerReaper]
23:22:14.884 INFO  [main] - Registering Garbage Collection every 60000 for memory.
23:22:14.884 INFO  [main] - Registering Garbage Collection every 120000 for other resources.
23:22:14.884 INFO  [main] - Started service [GCCollectionService]
23:22:15.074 INFO  [main] - Started service [MessageManager]
23:22:15.084 INFO  [main] - JMS Server is bound to //localhost:1099/OpenJMSServer
23:22:15.104 INFO  [main] - JMS Admin Server is bound to //localhost:1099/JmsAdminServer
```

The important thing to note is that the server uses RMI with an embedded JNDI provider, called *JndiServer*, running on the *localhost* at port *1099*. This information will be required, by the user, to formulate the url.

Using the OpenJMS Web Application

Once the application has been deployed and both Tomcat and OpenJMS have been started then use the following URL to interact with the application.

```
http://localhost:2222/openjms/servlets/SimplePublisher.html
```

The following page, excluding the information, will be rendered in the browser.

Fill in the details, as specified above, and press *Submit Query*. This request will publish *20* messages under the topic *jima*.

Resources

OpenJMS

web site	http://openjms.sourceforge.net
mailing lists	http://sourceforge.net/mail/?group_id=54559
download	http://openjms.sourceforge.net/download.html
specifications	http://java.sun.com/products/jms

Jakarta Tomcat

web site	http://jakarta.apache.org/tomcat/index.html
subscribe to mailing list	http://jakarta.apache.org/site/mail.html
mail archive	http://jakarta.apache.org/site/mail.html
download	http://jakarta.apache.org/site/binindex.html
specifications	http://java.sun.com/products/servlets
	http://java.sun.com/products/jsp

Appendix B: OpenJMS Library Dependencies

Runtime Library Dependencies

OpenJMS Server

The OpenJMS server requires the following libraries at runtime.

Library	Version	Source
openjms-0.7.5.jar	0.7.5	http://openjms.sourceforge.net
jdbm.jar	0.7	http://sourceforge.net/projects/jdbm [13]
log4J_1.1.3.jar	1.1.3	http://jakarta.apache.org/log4j/docs/index.html
xerces-2.3.0.jar	2.3.0	http://xml.apache.org/xerces-j/index.html
xml-apis-1.0.b2.jar	1.0.b2	http://xml.apache.org/commons
xalan-2.4.1.jar	2.4.1	http://xml.apache.org/xalan
antlr_2.7.2a2.jar	2.7.2a2	http://www.antlr.org
jdbc2_0-stdext.jar	2.0	http://java.sun.com/products/jdbc [14]
jndi_1.2.1.jar	1.2.1	http://java.sun.com/products/jndi
jms_1.0.2a.jar	1.0.2a	http://java.sun.com/products/jms/
jta_1.0.1.jar	1.0.1	http://java.sun.com/products/jta
oro-2.0.4.jar	2.0.4	http://jakarta.apache.org/oro/index.html
tyrex-0.9.8.7.jar	0.9.8.7	http://www.tyrex.org
castor-0.9.3.jar	0.9.3	http://www.castor.org
exolabcore-0.3.5.jar	0.3.5	http://openjms.sourceforge.net

OpenJMS Client

The OpenJMS requires the following libraries at runtime.

Library	Version	Library
jms_1.0.2a.jar	1.0.2a	http://java.sun.com/products/jms/
exolabcore-0.3.4.jar	0.3.3	http://openjms.sourceforge.net

[13] If using JDBM for persistency

[14] If using JDBC for persistency

jndi_1.2.1.jar	1.2.1	http://java.sun.com/products/jndi
openjms-client-0.7.5.jar	0.7.5	http://openjms.sourceforge.net
openjms-rmi-0.7.5.jar[15]	0.7.5	http://openjms.sourceforge.net

The following libraries are required to build OpenJMS. Note that all of these libraries are provided in the source distribution, and are also available from CVS.

Library	Version	Library
jdbm.jar	0.7	http://sourceforge.net/projects/jdbm
log4J_1.1.3.jar	1.1.3	http://jakarta.apache.org/log4j/docs/index.html
xerces-2.3.0.jar	2.3.0	http://xml.apache.org/xerces-j/index.html
xml-apis-1.0.b2.jar	1.0.b2	http://xml.apache.org/commons
xalan-2.4.1.jar	2.4.1	http://xml.apache.org/xalan
antlr_2.7.2a2.jar	2.7.2a2	http://www.antlr.org
jdbc2_0-stdext.jar	2.0	http://java.sun.com/products/jdbc
jndi_1.2.1.jar	1.2.1	http://java.sun.com/products/jndi
jms_1.0.2a.jar	1.0.2a	http://java.sun.com/products/jms/
jta_1.0.1.jar	1.0.1	http://java.sun.com/products/jta
oro-2.0.4.jar	2.0.4	http://jakarta.apache.org/oro/index.html
tyrex-0.9.8.7.jar	0.9.8.7	http://www.tyrex.org
castor-0.9.3.jar	0.9.3	http://www.castor.org
exolabcore-0.3.5.jar	0.3.5	http://openjms.sourceforge.net
junit_3.7.jar	3.7	http://www.junit.org
ant_1.5.1.jar	1.5.1	http://ant.apache.org/
ant_optional_1.5.1.jar	1.5.1	http://ant.apache.org/

[15] Only require this library if you are connecting to an RMI server