User Guide

http://mom4j.sourceforge.net

The mom4j development team

Table of Contents

1. INTRODUCTION 3

2. INSTALLING AND RUNNING MOM4J 3

3. JNDI (JAVA NAMING AND DIRECTORY INTERFACE) 3

4. CONFIGURATION 3

5. ADMINISTRATION 4

5.1. NAMING SERVER 6
5.2. MESSAGING SERVER 7
5.3. XCP SERVER 8

6. EMBEDDING MOM4J 8

7. WRITING A MESSAGING CLIENT 9

7.1. JAVA: SENDING A MESSAGE TO A QUEUE 9
7.2. PYTHON: SENDING A MESSAGE TO A QUEUE 10
7.3. JAVA: RECEIVING A MESSAGE FROM A QUEUE 11
7.4. PYTHON: RECEIVING A MESSAGE FROM A QUEUE 12

1. Introduction

Mom4j is a pure Java implementation of the JMS specification issued by Sun Microsystems. See [mom4j] for the key features of mom4j. Before you use mom4j, it makes a lot of sense to make yourself familiar with the JMS specification. You can download the JMS specification from the Sun Homepage (see [jms]).

2. Installing and Running mom4j

After you downloaded mom4j (see [mom4j]), unpack the archive. It unpacks into a directory named `mom4j`. Read the instructions in the `README.txt` file to compile and run mom4j.

3. JNDI (Java Naming and Directory Interface)

Mom4j comes with it's own JNDI server. JNDI is an important component in the JMS specification (refer to the JMS Spec for further details). As per version 1.1 of mom4j, there is no way of running mom4j with an external JNDI server. The possibility to integrate with an external JNDI server will probably come with a future release of mom4j.

4. Configuration

Mom4j is configured via a configuration file named `mom4j-config.xml`. It is located in the `config` subdirectory.

```
<?xml version="1.0"?>
<mom4j-config>
    <server>
        <port>4444</port>
        <admin-port>8888</admin-port>
        <jndi-port>8001</jndi-port>
        <thread-count>3</thread-count>
    </server>
    <users>
        <user id="system" admin="true" password="secret"/>
    </users>
    <store>
        <messages><![CDATA[ store  ]]></messages>
        <durables><![CDATA[ config/durable.dat  ]]></durables>
    </store>
    <client>
        <poll-interval sync="500" async="1000"/>
    </client>
    <destinations>
        <queue name="testQueue"/>
        <topic name="testTopic"/>
    </destinations>
</mom4j-config>
```

In the **server** section, the server specific properties are specified:

- **port**: the port the mom4j server listens on
- **admin-port**: the port of the administration server
- **jndi-port**: the port of the JNDI server

The **users** section defines the users that are allowed to connect to the mom4j server. As per version 1.1, any client can connect to the mom4j server. Any username or password given when opening a connection is ignored. There is no user-administration for mom4j so far. This is a feature that will come with one of the future releases.
For now, one user entry is necessary for the mom4j administration. A user has the following attributes:

- **id**: the id of the user
- **admin**: a flag indicating whether the user has administration rights
- **password**: the users password

The password is entered in clear text. Password encryption will be a feature in one of the future mom4j releases.

The **store** section holds parameters related to persistence issues:

- **messages**: the absolute or relative path to the message store directory of mom4j
- **durables**: the absolute or relative path to a file that mom4j uses to store durable subscriptions

The **client** section holds configuration parameters relevant for the messaging clients.

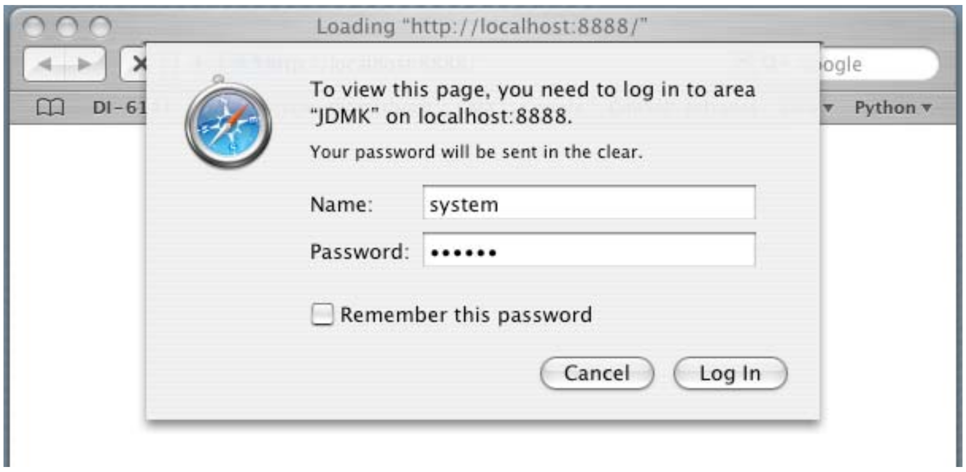
- **Poll-interval**: As mom4j uses XML/HTTP as a wire protocol, there is no communication from the server to client. If the client e.g. uses a `MessageListener` (refer to the JMS specification for further details), the client has to poll the server for new messages
  - **sync**: the interval for synchronous polls, e.g. when calling the receive (<timeout>) method
  - **async**: the interval for actions, that appear to be asynchronous to the client (e.g. When using a `MessageListener`)

The **destinations** section can be used to configure permanent messaging destinations (see the JMS specification for details).

- **queue**: Marks the destination as a queue
  - **name**: the name of the queue
- **topic**: Marks the destination as topic
  - **name**: the name of the topic

5. Administration

One part of the mom4j administration is adjusting the mom4j-config.xml file (see chapter 4) to the certain needs. Every change in the configuration file requires a restart of the mom4j server.
Remote administration can be done by pointing the browser to the administration port of the mom4j server. If you started mom4j on your local machine and didn't change the settings in the config-file, the URL is http://localhost:8888.



Loading "http://localhost:8888/"

To view this page, you need to log in to area "JDMK" on localhost:8888.

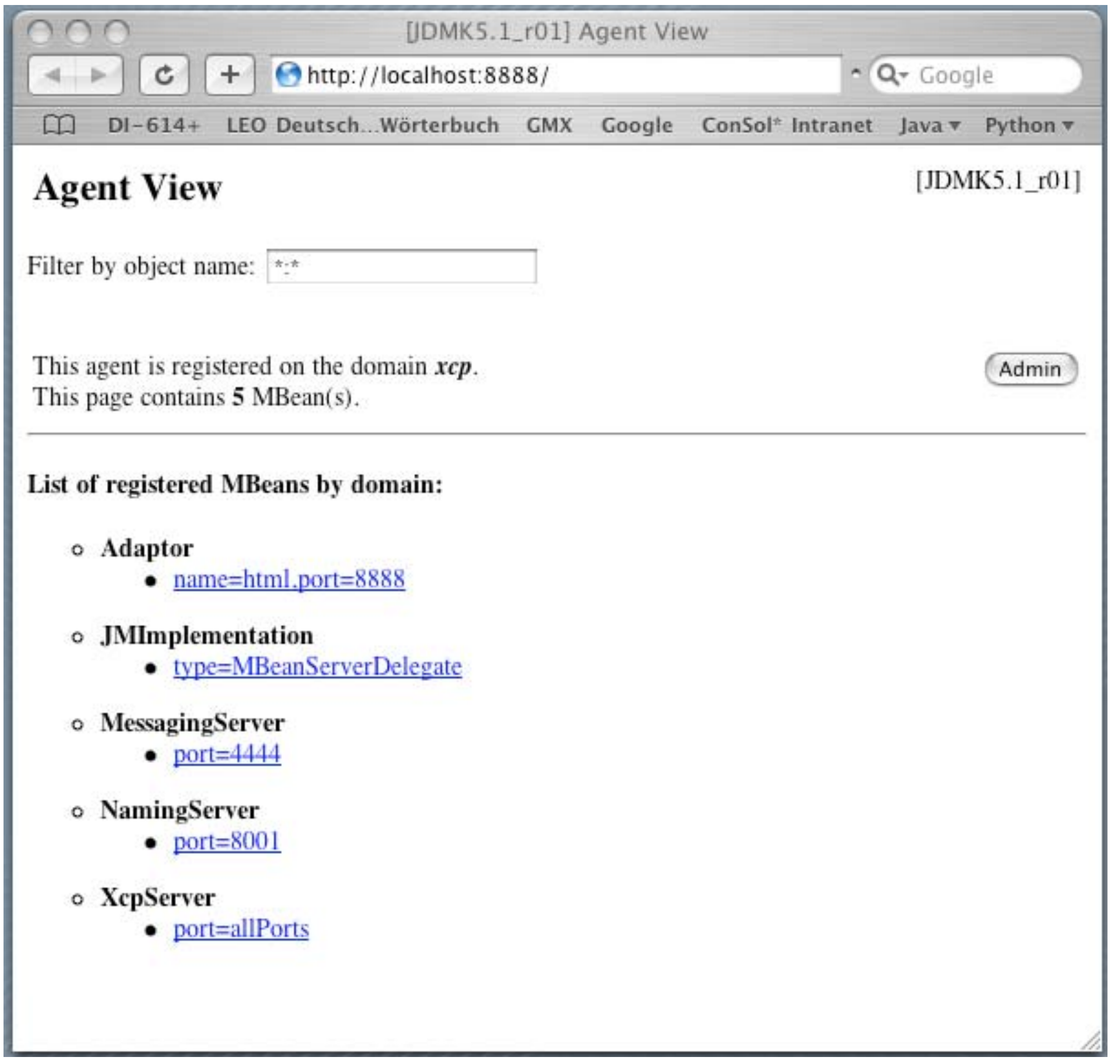
Your password will be sent in the clear.

Name: system

Password: ••••••

Remember this password

After logging in with username and password (default is system – secret) the following screen appears:



[JDMK5.1_r01] Agent View

http://localhost:8888/

Agent View [JDMK5.1_r01]

Filter by object name: *:*

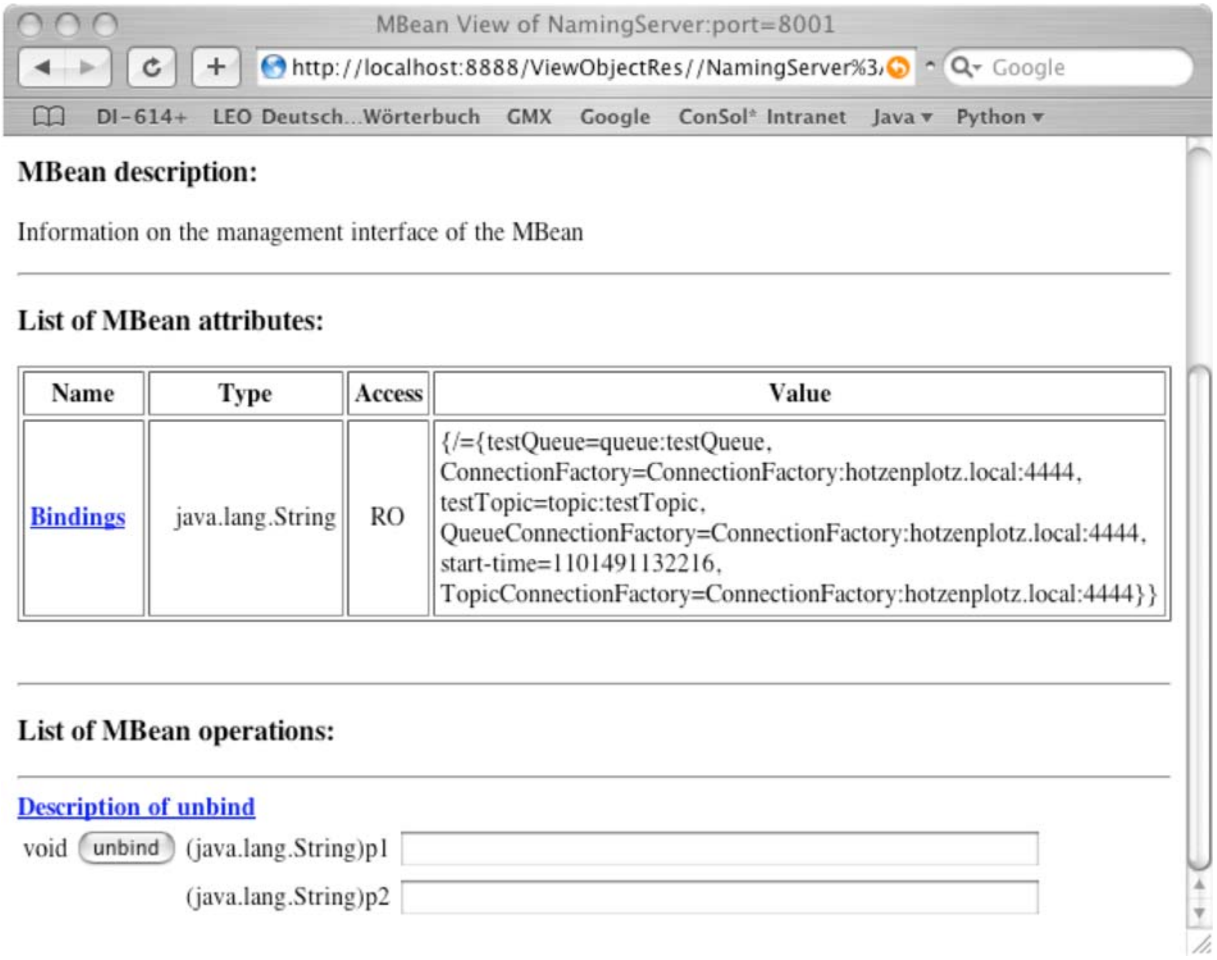
This agent is registered on the domain *xcp*.
This page contains **5** MBean(s).

List of registered MBeans by domain:

- **Adaptor**
  - name=html,port=8888
- **JMImplementation**
  - type=MBeanServerDelegate
- **MessagingServer**
  - port=4444
- **NamingServer**
  - port=8001
- **XcpServer**
  - port=allPorts

It shows the different sections where admin tasks can be performed.
Mom4j uses JMX® and the JMX reference implementation by Sun Microsystems® for runtime administration and management. The Adaptor and the JMImplementation shown in the picture above are part of the reference implementation.

5.1. *Naming Server*


MBean View of NamingServer:port=8001
http://localhost:8888/ViewObjectRes//NamingServer%3,
Google
DI-614+ LEO Deutsch...Wörterbuch GMX Google ConSol* Intranet Java Python

MBean description:

Information on the management interface of the MBean

List of MBean attributes:

Name	Type	Access	Value
Bindings	java.lang.String	RO	{/={testQueue=queue:testQueue, ConnectionFactory=ConnectionFactory:hotzenplotz.local:4444, testTopic=topic:testTopic, QueueConnectionFactory=ConnectionFactory:hotzenplotz.local:4444, start-time=1101491132216, TopicConnectionFactory=ConnectionFactory:hotzenplotz.local:4444}}

List of MBean operations:

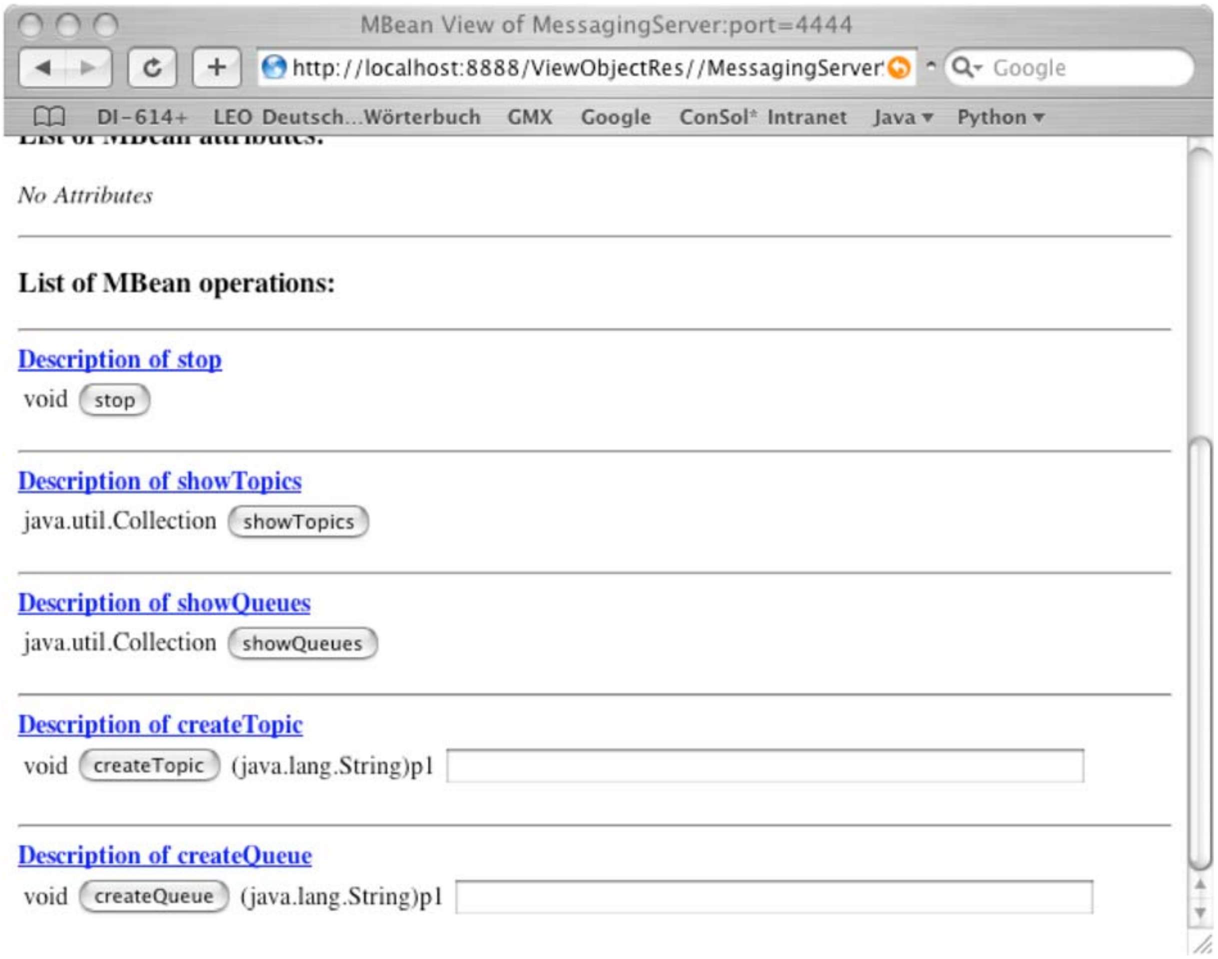
Description of unbind

void unbind (java.lang.String)p1

(java.lang.String)p2

The *NamingServer* MBean (see [jmx] for details about the term *MBean*) allows the administrator to view the bindings of the different naming contexts. By default, testQueue, testTopic, QueueConnectionFactory and TopicConnectionFactory are bound to the root context ('/'). It is also possible for the administrator to unbind a name. The first argument is the naming context. The second argument is the name of the binding the administrator wants to remove.

5.2. Messaging Server


MBean View of MessagingServer:port=4444
http://localhost:8888/ViewObjectRes//MessagingServer
Google
DI-614+ LEO Deutsch...Wörterbuch GMX Google ConSol* Intranet Java Python

No Attributes

List of MBean operations:

Description of stop
void stop

Description of showTopics
java.util.Collection showTopics

Description of showQueues
java.util.Collection showQueues

Description of createTopic
void createTopic (java.lang.String)p1

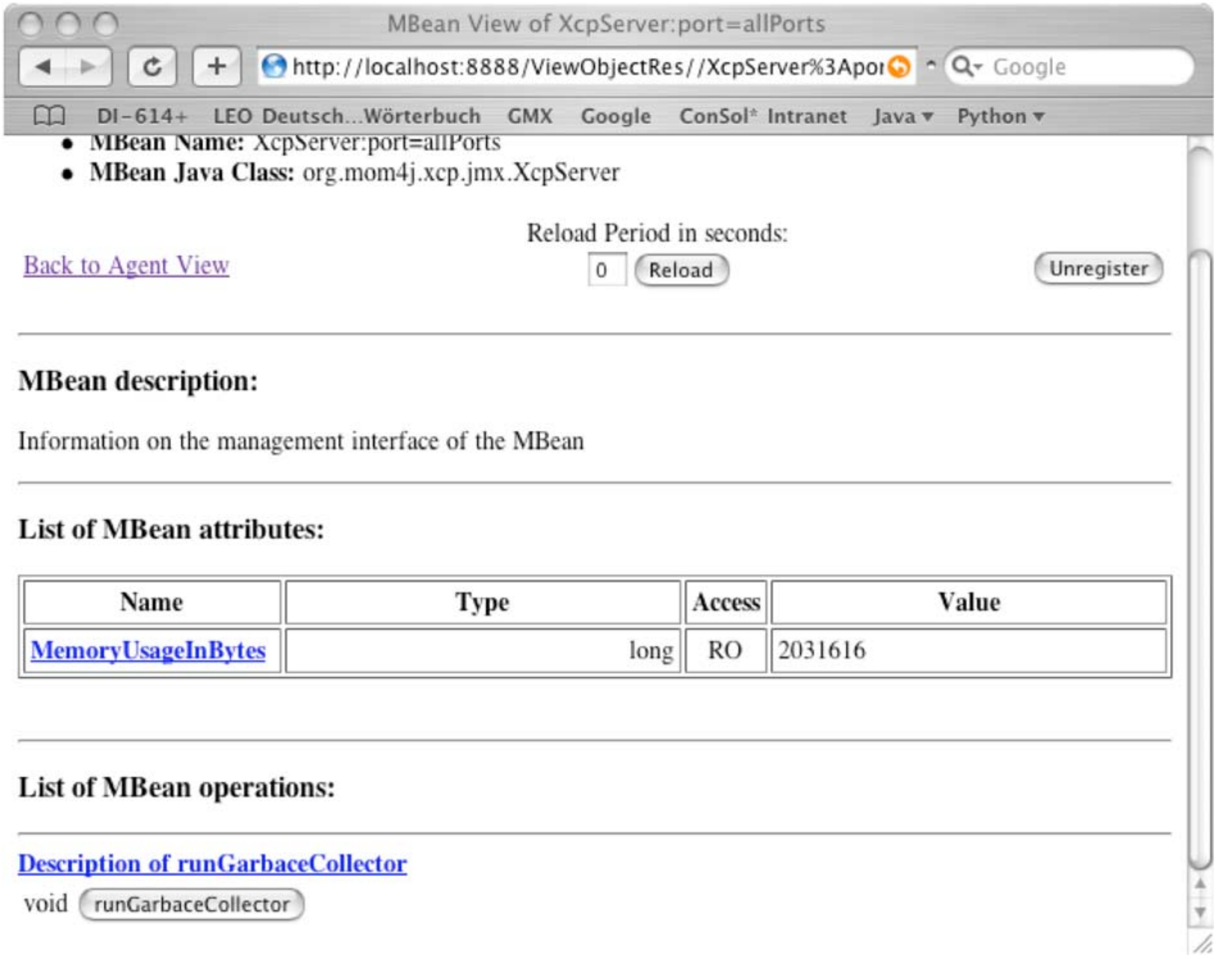
Description of createQueue
void createQueue (java.lang.String)p1

The *MessagingServer* MBean allows the administrator to

- stop the mom4j server (**stop** operation)
- view all the topics currently available (**showTopics** operation)
- view all the queues currently available (**showQueues** operation)
- create a new topic at runtime (**createTopic** operation)
- create a new queue at runtime (**createQueue** operation)

Queues and Topics created at runtime are gone when the server is restarted. Messages do not get lost. They remain in the message store. To restore messages after a server restart simply recreate the queue or the topic. For permanent availability of queues and topics, add them to the mom4j-config.xml file (see chapter 4).

5.3. *Xcp Server*


MBean View of XcpServer:port=allPorts
http://localhost:8888/ViewObjectRes//XcpServer%3Apor
Google
DI-614+ LEO Deutsch...Wörterbuch GMX Google ConSol* Intranet Java Python

- **MBean Name:** XcpServer:port=allPorts
- **MBean Java Class:** org.mom4j.xcp.jmx.XcpServer

Back to Agent View

Reload Period in seconds:
0 Reload

Unregister

MBean description:

Information on the management interface of the MBean

List of MBean attributes:

Name	Type	Access	Value
MemoryUsageInBytes	long	RO	2031616

List of MBean operations:

Description of runGarbaceCollector

void runGarbaceCollector

Xcp is the basic server and communication infrastructure of mom4j. The *XcpServer* MBean shows the current memory usage of the server process and allows to force running the garbage collector.

6. Embedding mom4j

Mom4j can be embedded into other peaces of software by using the classes and interfaces of the `org.mom4j.api` package (see the javadoc for details).

```
Mom4jConfig serverConfig = MyMom4jConfig(...);
Mom4jConsole server = Mom4jFactory.start(serverConfig);
// ...
// Use the console to send or receive messages
// ...
// Clients can connect an send/receive messages
// ... server.stop();
```

You have to implement the `Mom4jConfig` interface. If you want to use the `mom4jconfig.xml` file for configuring mom4j, you can use the class `org.mom4j.config.ConfigImpl` to read the configuration from the standard configuration file. If you have your own store for configuration parameters (e.g. your own custom configuration file or a database), the configuration information can be given to the mom4j server via the `Mom4jConfig` interface.
The static method `Mom4jFactory.start` launches the mom4j server and returns an instance of `Mom4jConsole`. This interface can be used to stop the server or to send and receive messages.

7. Writing a Messaging Client

This chapter shows simple examples of messaging clients written in Java and Python (see [jms-spec] for more examples and a detailed explanation of the JMS API). The Java clients need to have the jms.jar (downloaded from the source mentioned in the README.txt file coming with the mom4j distribution) and the mom4j-client.jar (created with the ant target `makelib` as mentioned in README.txt) in the `classpath`. Python clients need to have the `org.mom4j` package installed. You can download the python client from [mom4j]. The distribution contains a README file with further instructions.
It is worth trying to receive a message from mom4j `testQueue` with the python script in chapter 7.4 that has been sent to the queue with the java client from chapter 7.1 (or the other way round).

7.1. Java: Sending a message to a queue

```
import java.util.Properties;

import javax.naming.*;
import javax.jms.*;

public class Sender {

    public static void main(String[] args) {
        QueueConnection qc = null;
        try {
            Properties p = new Properties();
            p.put(Context.INITIAL_CONTEXT_FACTORY,
                  "org.mom4j.jndi.InitialCtxFactory");
            p.put(Context.PROVIDER_URL, "xcp://localhost:8001");
            Context ctx = new InitialContext(p);
            QueueConnectionFactory qcf =
                (QueueConnectionFactory)
                    ctx.lookup("QueueConnectionFactory");
            Queue queue = (Queue)ctx.lookup("testQueue");
            qc = qcf.createQueueConnection("system", "system");
            QueueSession qs =
                qc.createQueueSession(false,
                                      Session.AUTO_ACKNOWLEDGE);
            QueueSender qsend = qs.createSender(queue);
            TextMessage tm = qs.createTextMessage();
            tm.setText("How's it bud?");
            tm.setStringProperty("string", "yet another string");
            tm.setIntProperty("int", 456);
            tm.setFloatProperty("float", 9.82f);
            tm.setLongProperty("long", 987654321);
            qsend.send(tm);
            qsend.close();
            qs.close();
        } catch(Exception ex) {
            ex.printStackTrace();
        } finally {
           try {
               qc.close();
           } catch(Exception ex) { }
        }
    }

}
```

The above example creates an initial jndi naming context (providing the url to the mom4j jndi server and the mom4j implementation class of the context factory) and performs a lookup on the *administered objects* `QueueConnectonFactory` and `testQueue`. The queue connection factory is always available (as per jms spec), the `testQueue` is configured in the original `mom4j-config.xml` file. Then, a queue connection is created via the queue connection factory (as per mom4j 1.1, authentication is not supported on the server, so any username and password can be entered here). The next step is creating a session. In the example a non-transacted session with message acknowledge mode **auto** is created (see [jms-spec] for details). On top of the session, a `QueueSender` is created. As mom4j is 1.1 compliant, one can also create a `MessageProducer` (using `ConnectionFactory` instead of `QueueConnectionFactory` and `Session` instead of `QueueSession`). A `QueueSender` can send messages to a queue only, whereas a `MessageProducer` can send messages to queues and topics (see [jms-spec] for further details). Finally, a text message is created (via the session object), filled with text and properties and is then sent to the test queue. When using a transacted session, `session.commit()` has to be called to actually send the message.

7.2. Python: Sending a message to a queue

```
from org.mom4j import jms
from org.mom4j import jms_msg

con = None
sess = None
try:
    con = jms.Connection("http://localhost:4444/XCP")
    sess = con.createSession(jms.TRANSACTED, jms.AUTO_ACKNOWLEDGE)
    msg = sess.createTextMessage()
    msg.setStringProperty("string", "this is a string")
    msg.setIntProperty("int", 123)
    msg.setFloatProperty("float", 3.24)
    msg.setLongProperty("long", 1234567890)
    msg.setText("How's it buddy?")
    dest = jms.Destination("testQueue", 1)
    sender = sess.createProducer(dest)
    sender.send(msg)
    sess.commit()
    con.close()
finally:
    if sess:
        sess.close()
    if con:
        con.close()
```

The python client delivered with mom4j has a jms-like interface. It supports text messages only (other message types may be supported later; `ByteMessage` and `ObjectMessage` for example don't make much sense in python …). The steps required are very similar to writing a java client. There is no jndi lookup in python. The python client creates a connection object directly (not via a factory) and needs to know the name of the destination he wants to send to. He creates a destination object directly (instead of looking it up in the jndi). The flag – the second parameter – indicates if it is a queue or a topic.
The example uses a transacted session, so a commit is required to actually send the message to the queue.

7.3. Java: Receiving a message from a queue

```
import java.util.Properties;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.jms.*;


public class Receiver {


    public static void main(String[] args)
    {
        QueueConnection qc = null;
        try {
            Properties p = new Properties();
            p.put(Context.INITIAL_CONTEXT_FACTORY,
                 "org.mom4j.jndi.InitialCtxFactory");
            p.put(Context.PROVIDER_URL, "xcp://localhost:8001");
            Context ctx = new InitialContext(p);
            QueueConnectionFactory qcf =
                (QueueConnectionFactory)
                    ctx.lookup("QueueConnectionFactory");
            Queue queue = (Queue)ctx.lookup("testQueue");
            qc = qcf.createQueueConnection("system", "system");
            QueueSession qs =
                qc.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
            QueueReceiver qr = qs.createReceiver(queue);
            TextMessage tm = (TextMessage)qr.receiveNoWait();
            if(tm != null) {
                System.out.println("got text message: " + tm.getText());
                System.out.println("string property: " +
                    tm.getStringProperty("string"));
                System.out.println("int property: " +
                    tm.getIntProperty("int"));
                System.out.println("float property: " +
                    tm.getFloatProperty("float"));
                System.out.println("long property: " +
                    tm.getLongProperty("long"));
            } else {
                System.out.println("got nothing");
            }
            qr.close();
            qs.close();
        } catch(Exception ex) {
            ex.printStackTrace();
        } finally {
            if(qc != null) {
                try {
                    qc.close();
                } catch(Exception ex) { }
            }
        }
    }

}
```

If you compare the code above with the one from chapter 7.1, you will notice a lot of things the two clients have in common. Instead of creating a sender, a queue receiver is created. Messages can be received via the various receive methods (refer to [jms-spec] for further details). This example uses a non transactional session with message acknowledge mode **auto**. No acknowledge method on the message object and no commit method on the session objects needs to be called to remove the message from the queue.

7.4. Python: Receiving a message from a queue

```
from org.mom4j import jms

con = None
sess = None
try:
    con = jms.Connection("http://localhost:4444/XCP")
    sess = con.createSession(jms.NON_TRANSACTED, jms.CLIENT_ACKNOWLEDGE)
    dest = jms.Destination("testQueue", 1)
    consumer = sess.createConsumer(dest)
    msg = consumer.recieveNoWait()
    if msg:
        print "got message: '%s'" % msg.getText()
        print "string property value: '%s'" \
              % msg.getStringProperty("string")
        print "int property value: '%d'" % msg.getIntProperty("int")
        print "float property value: '%f'" % msg.getFloatProperty("float")
        print "long property value: '%d'" % msg.getLongProperty("long")
        print "destination was %s" % msg.getJMSDestination()
        msg.acknowledge()
    else:
        print "got noting!"
finally:
    if sess:
        sess.close()
    if con:
        con.close()
```

Compared to chapter 7.2, the code is quite similar. Instead of creating a producer, a consumer is used to receive messages from the queue. In this example, a non transacted session is used with message acknowledge mode **client**. Only when the acknowledge method on the message object is called, the message will be removed from the queue.

References

[mom4j]	http://mom4j.sourceforge.net
[jms]	http://java.sun.com/products/jms
[jms-spec]	Java Message Service, Mark Hapner et al, April 2002, Sun Microsystems (available at [jms])
[jmx]	Java Management Extension, http://www.java.sun.com/products/JavaManagement