U1 TECHNOLOGIES

AmbrosiaMQ

JMS Quick Installation Guide

April 2009

AmbrosiaMQ™

Revision: 042409

U1 Technologies
204 Tiburon Boulevard
San Rafael, CA 94901

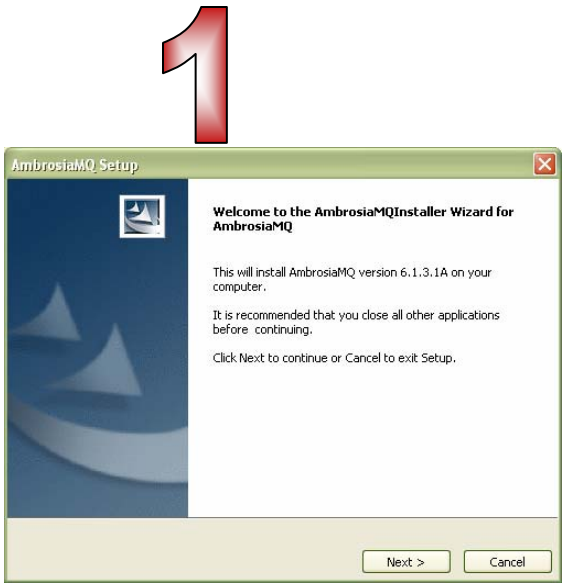
phone: (415) 480-0318
fax: (415) 704-3275
web: http://www.u1.com
e-mail: info@u1.com

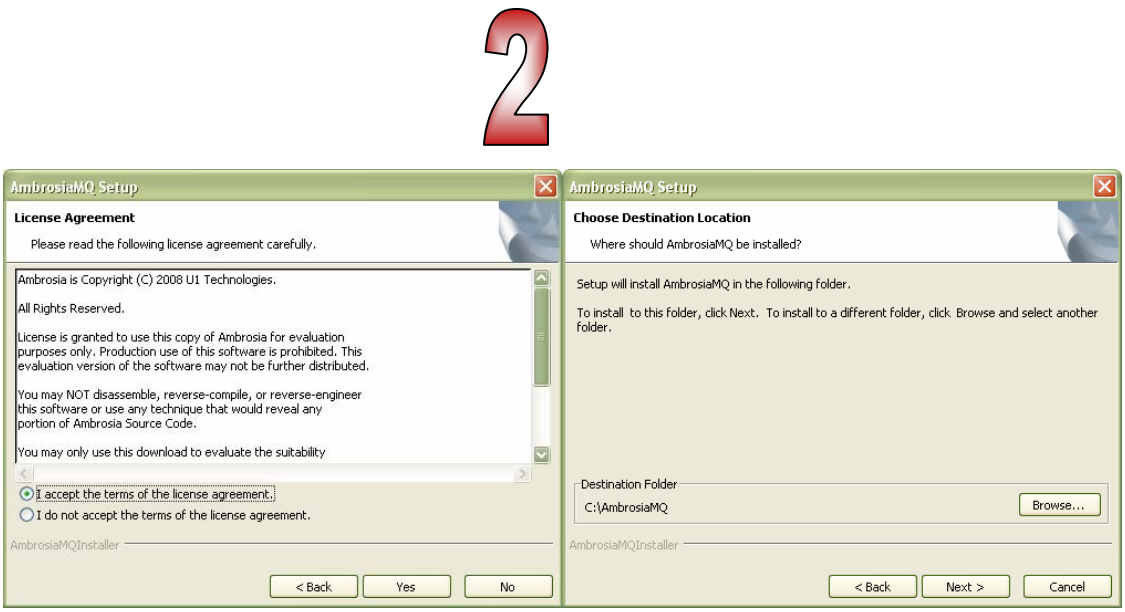
AmbrosiaMQ – JMS Quick Installation Guide

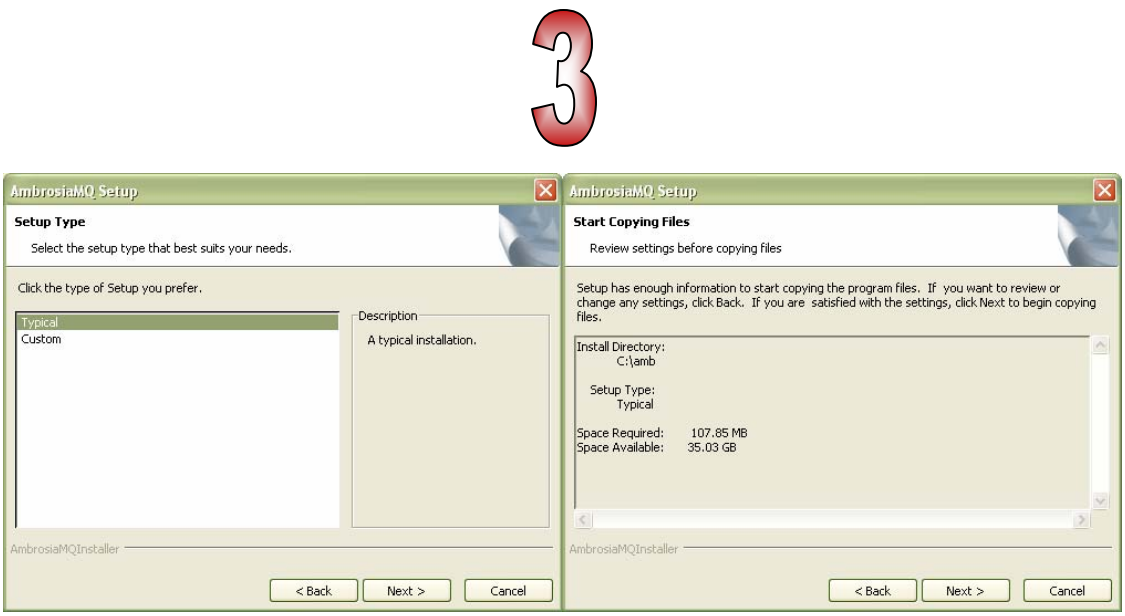
The purpose of this guide is to provide a very basic set of instructions so that a JMS developer can quickly start using AmbrosiaMQ. There are many other advanced features of AmbrosiaMQ that are covered in the documents *AmbrosiaMQ – Concepts and Capabilities* and *AmbrosiaMQ – Administration Guide*, as well as *AmbrosiaMQ – API Javadoc*.

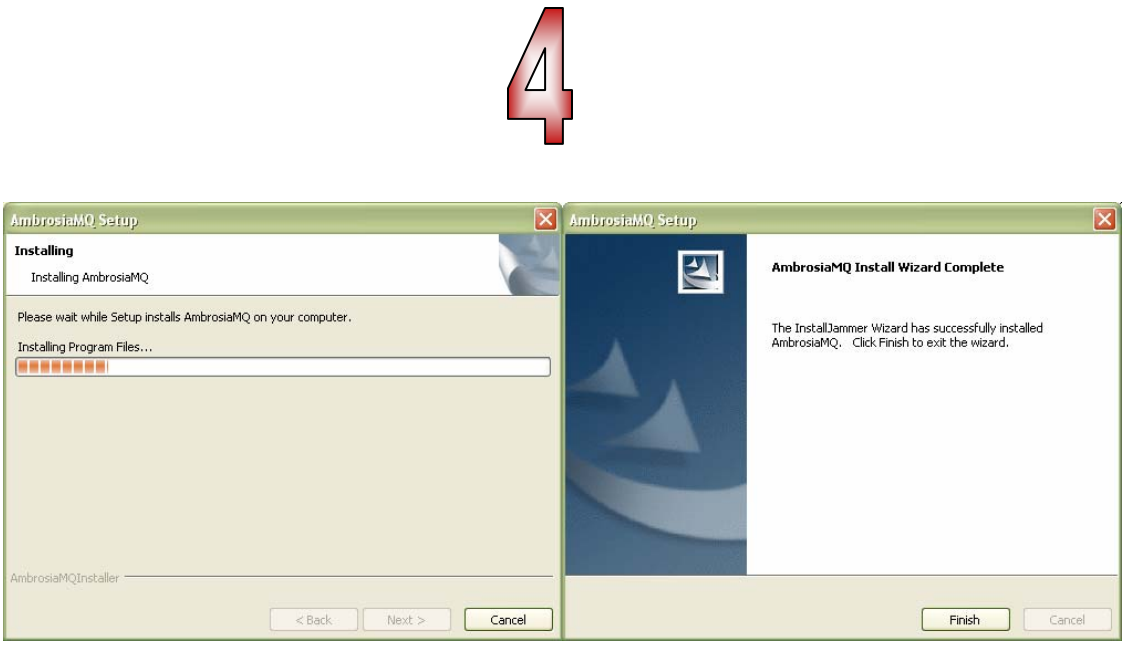
1 Installation

AmbrosiaMQ can be installed very easily and quickly using a four step process. Simply execute the command **`setup.bin`** (for Solaris and Linux) or **`setup.exe`** (for windows) and follow the installation instructions as illustrated below.









2 Running the AmbrosiaMQ Message Broker

To run the AmbrosiaMQ message broker you must complete the following step. Refer to the document *AmbrosiaMQ Administration Guide* for additional details.

1. In the install directory edit the file `ambroker.ini` and ensure values for broker name, broker password, IP address, port number, and security enablement are correctly set. Note that lines that start with the # character are comments and should be uncommented if you want to set the property's value. For example:
    - `BROKER_NAME=master`
    - `BROKR_PASSWORD=secret`
    - `ACCEPTORS = 8001, ssl://8002`
    - `ENABLE_SECURITY=true`

2. Ensure that a file called *jndi.properties* with appropriate attributes exists in the AmbrosiaMQ message broker's installation directory (in the same directory where the file `ambroker.ini` exists). A simple file-based JNDI provider is bundled with AmbrosiaMQ and can be copied from `INSTALL_DIR/samples/jms/jndi.properties`.
3. Initialize the broker's database by executing the command `initdb` in the directory named after the operating system of the host machine (i.e., `solaris/initdb` or `linux/initdb` or `windows\initdb`). This will use the Derby embedded database. Of course, AmbrosiaMQ works with MySQL, Oracle and other JDBC-compliant databases
4. Initialize the broker's security database by executing the command `initsecdb` in the directory named after the operating system of the host machine (i.e., `solaris/initsecdb` or `linux/initsecdb` or `windows\initsecdb`).
5. Run the broker by executing the command `runbroker` (for windows) or `runambrosia` (for solaris or linux) in the directory named after the operating system of the host machine (i.e., `solaris/runambrosia` or `linux/runambrosia` or `windows\runbroker`).

3 ConnectionFactory

To start using JMS, a connection to the JMS server (i.e., an AmbrosiaMQ broker) must first be established. `ConnectionFactory` allows Connections to be created to a JMS provider. There are two methods of obtaining a `ConnectionFactory` as described in the following subsections. Please note that the JMS application must provide a valid user ID and password. You can use the AmbrosiaMQ Broker Administration Console (BAC) to add users and set their password (See Section 5 for how to run the BAC).

3.1 JNDI

A `ConnectionFactory` may be administratively created and stored in a JNDI repository (See Section 5.1 for details). JMS applications may subsequently use JNDI services to obtain the ConnectionFactory and create a Connection to the AmbrosiaMQ message broker. For example:

```
// get InitialContext
try {
    InitialContext jndi = new InitialContext();
    ConnectionFactory factory = (ConnectionFactory)
jndi.lookup("MyConnectionFactory");
    Connection connection = factory.createConnection("MyUser",
"MyPassword");
} catch (Exception e) {
    e.printStackTrace();
}
```

3.2 Native

Alternatively, the ConnectionFactory may be created directly by the JMS application using the native API. Note that this is not portable JMS code.

```
try {
    ConnectionFactory factory = new
    com.u1.jms.ConnectionFactory("MyAppId", "MyJmsHost");
    Connection connection = factory.createConnection("MyUser",
    "MyPassword");
} catch (Exception e) {
    e.printStackTrace();
}
```

4 Topics and Queues

As with `ConnectionFactory`, `Topic` and `Queue` are generally administratively configured (See Sections 5.2 and 5.3), but may also be accessed using the native API.

4.1 JNDI

Given an `InitialContext` (see above), a JMS application may retrieve the administratively configured `Topic` and/or `Queue` by looking them up using JNDI.

```
try {
    InitialContext jndi = new InitialContext();
    Topic myTopic = (Topic) jndi.lookup("MyTopic");
    Queue myQueue = (Queue) jndi.lookup("MyQueue");
} catch (Exception e) {
    e.printStackTrace();
}
```

4.2 Native

By forgoing portability, an application may access `Topic` and `Queue` without having to administratively configuring before hand.

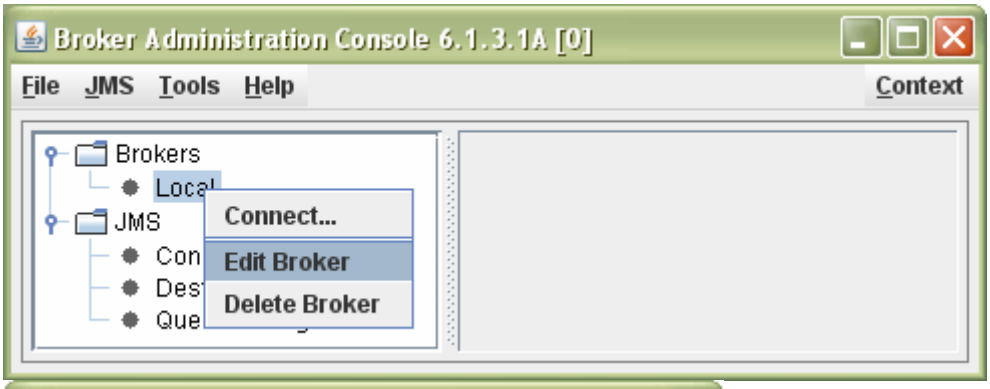
```
// assume given a Connection object called connection
try {
    Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);
    Topic myTopic = session.createTopic("MyTopic");
    Queue myQueue = session.createQueue("MyQueue");
} catch (Exception e) {
    e.printStackTrace();
}
```

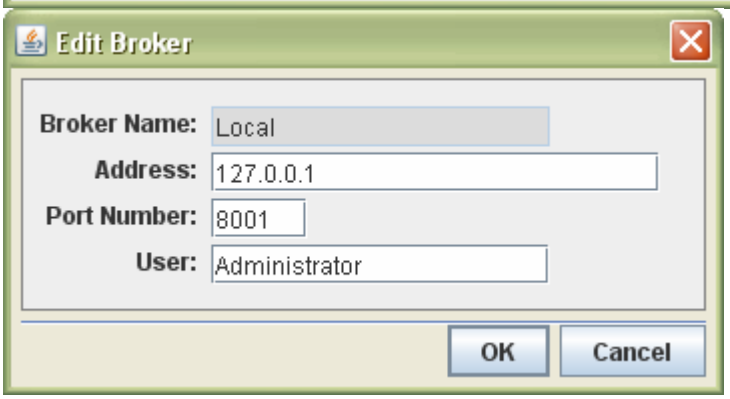
Note that the code above only creates a Queue reference. The Queue must have been previously created, usually administratively.

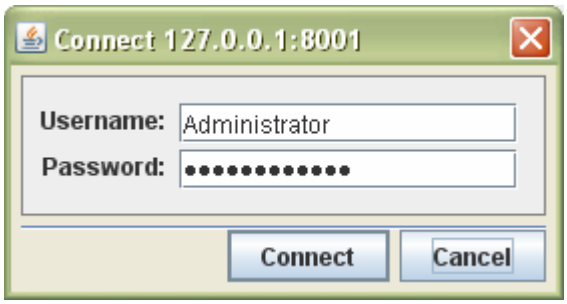
5 JMS Administration

The AmbrosiaMQ Broker Administration Console (BAC) allows administrators to manage all aspects of AmbrosiaMQ brokers, including creating users, managing groups, and configuring JMS objects such as `ConnectionFactory`, `Topic` and `Queue` for use by JMS applications.

To launch the BrokerAdminConsole, simply execute the script `bac` the directory named after the operating system of the host machine (i.e., `solaris/bac` or `linux/bac` or `windows\bac`). One you bring up the BAC, you can connect to the broker by first editing the specification of the broker to which you want to connect and then connecting to that broker. Below are screen shots showing these steps (note that editing the broker specification is only performed once). The default administrator user ID/password is `Administrator/Administrator`.







A file called *jndi.properties* with appropriate attributes must exist in the AmbrosiaMQ message broker's installation directory (in the same directory where the file `ambroker.ini` exists). Below is a sample jndi.properties file for LDAP.

```
java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory
java.naming.provider.url=ldap://localhost:389/ou=Ambrosia,dc=u1,dc=com
java.naming.security.authentication=simple
java.naming.security.principal=cn=directory manager
java.naming.security.credentials=dirmanager
```

AmbrosiaMQ can work with any standard implementation of JNDI. A simple file-based JNDI provider is bundled with AmbrosiaMQ and can be used by default. For refer to `INSTALL_DIR/samples/jms/jndi.properties`.

5.1 Adding a Connection

Right click on "Connection Factories" under the "JMS" folder (see Figure 1) and select "New JMS 1.1 ConnectionFactory …"

Enter the name for the ConnectionFactory. This will be used as the lookup name in JNDI.

Enter the application ID. This must be a unique string per application.

Enter the AmbrosiaMQ broker IP address and port number (JMS server). Connections will be made to this server.

Enter any Connection Property for the Connection. If left blank, the Connection will receive the default Connection Property.

Click "OK." (See Figure 2)

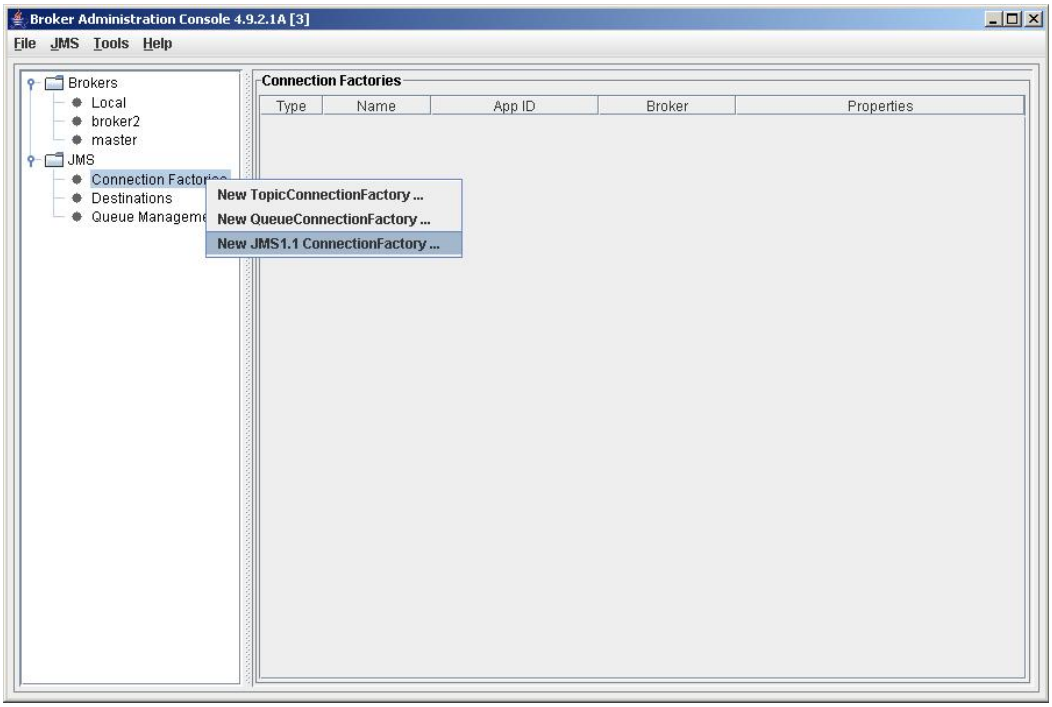


Figure 1 – Adding a new Connection Factory

Figure 2 – Details of a Connection Factory

5.1.1 Connection Properties

All native Connection properties are generally applicable to JMS Connection. The following are specific to JMS:

ADVANCED_JMS_BRIDGING (default=false) – Set this to true to enable detection of Text and Object Messages when the native setBody function is used to fill messages. This allows better evaluation of message types when a JMS application receives messages published by a native application.

QUEUE_RECEIVER_BATCHSIZE (default=10) – This property specifies the number of messages a QueueReceiver will prefetch from a Queue for better Queue performance. This value is a balance between performance and receiver fairness when multiple receivers reads from the same Queue. This property also affects native Queue receivers.

5.2 Adding a Topic

Right click on “Destinations” under the “JMS” folder and select “New Topic Destination…”

Enter the name of the Topic. This will also be used as the JNDI key for lookup. Click “OK.”

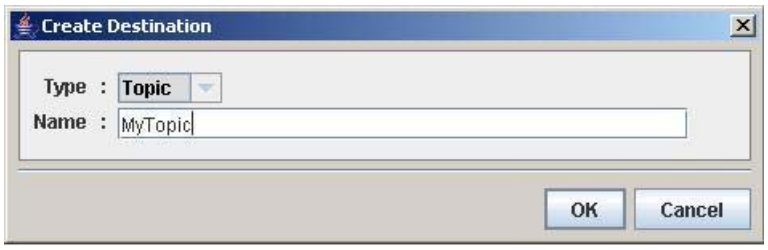


Figure 3 – Adding a Topic

5.3 Adding a Queue

Right click on “Desinations” under the “JMS” folder and select “New Queue Destination…”

Enter the name of the Queue. This will also be used as the JNDI key for lookup. A Queue must not contain the “@” character.

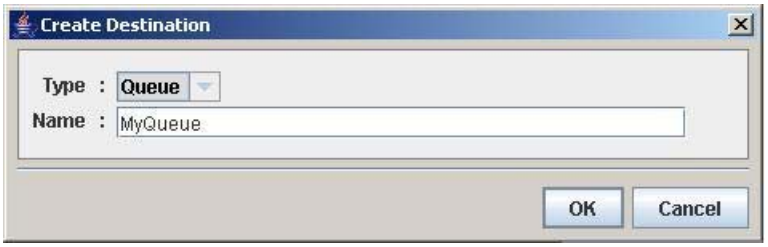


Figure 4 – Adding a Queue

5.4 Creating a Queue

Right click on “Queue Management” under the “JMS” tab and select “New Queue…” Note, that a connection to a broker hosting the queue must be established. The Queue will be created on the connected broker. Enter the name of the Queue and click “OK.” (See Figure 5)

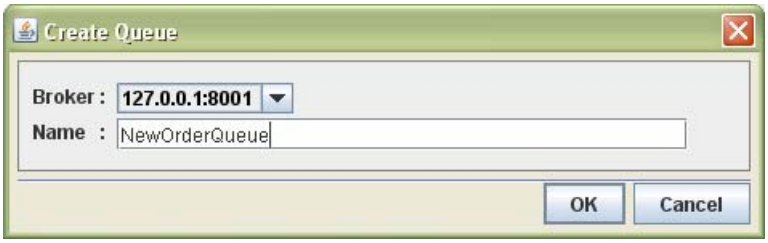


Figure 5 Creating a queue

6 Configuring JNDI

AmbrosiaMQ can work with any standard implementation of JNDI. A simple file-based JNDI provider is bundled with AmbrosiaMQ and can be used by default. Please note that a file called *jndi.properties* with appropriate attributes must exist in the AmbrosiaMQ message broker's installation directory (in the same directory where the file ambbroker.ini exists). For a sample *jndi.properties* file please see `INSTALL_DIR/samples/jms/jndi.properties`