JOTM Examples guide

Jeff Mesnil

May 19, 2003

Abstract

This guide describes examples provided with JOTM. It explains the scenarii of the examples, how to setup and run them.

Contents

1 Available Examples

For the moment, four examples are available with JOTM:

- The first one, **basic** example (section 2), is a ... basic example! But it has some interesting features such as to show how to configure JOTM to have clients accessing the same Transaction Manager on both RMI/JRMP and RMI/IIOP.
- The seconde one, **jdbc** example (section 3), shows how to use JOTM and JDBC to provide distributed transactions with a database.
- The third one, **jms** example (section 4), shows how to use JOTM with a JMS provider (JORAM in that case) to support distributed transactional messages for your applications.
- The fourth one, **tomcat** example (section 5), shows how to use JOTM with Tomcat to provide distributed transactions with a database from a Servleto or a JSP.

2 Basic Example

All Ant commands are to be executed from the `examples/basic/` directory of a JOTM *distribution* (examples won't work from JOTM *source* directory).

2.1 Scenario

The **basic** example is a very simple example showing how to use a Transaction Manager. The client application (`BasicExample` class) looks up the `UserTransaction`. Then it makes two transactions:

- the first one is a simple begin/commit

- the second one is also a simple begin/commit but this one is rolled back due to a timeout expiration (we made slept the BasicExample thread for longer than the transaction timeout set for this transaction)

2.2 Setup and compilation

To set up this example, you'll need:

- a name server (either a RMI registry or a CORBA nameserver)
- JOTM is the Transaction Manager which provides a `UserTransaction` through JNDI
- `BasicExample` is the client application
- Ant tool to compile and run the example

To compile the example, in the `examples/basic/` directory, type

```
$ ant compile
```

2.3 Run the example

First, you've to set `JOTM_HOME` to the directory of your JOTM distribution (e.g., `.../jotm/output/dist` from CVS).

```
$ export JOTM_HOME=<JOTM_distribution_directory>
```

To run the example, you have to be in the `examples/basic/` directory.
You can run the example with three different configurations for protocol communication:

- JOTM is accessible only through **RMI/JRMP** (default configuration)
- JOTM is accessible only through **RMI/IIOP**
- JOTM is accessible through *both* **RMI/JRMP** *and* **RMI/IIOP**

There are two commands to run the example:

```
$ ant run.rmi.jrmp
$ ant run.rmi.iiop
```

The first one assumes that JOTM is accessible through RMI/JRMP and that a RMI registry is running on its default port (i.e. 1099).
The second one assumes that JOTM is accessible through RMI/IIOP and that a CORBA name server is running on port 19751.
Both also assume that the `UserTransaction` object is accessible on JNDI

with the name `UserTransaction`.

It has to be noted that these two targets are using the same class, `BasicExample`. They just differ in their settings: one is for RMI/JRMP communication whereas the other is a pure RMI/IIOP client.

2.3.1 On RMI/JRMP

To run the example on RMI/JRMP, type in `$JOTM_HOME/lib/`,

```
$ rmiregistry -J-classpath -Jjotm.jar:jotm_jrmp_stubs.jar \
  -J-Djava.security.policy=../config/java.policy &
```

Then in `$JOTM_HOME/lib/`, type

```
$ java -classpath jotm.jar:jotm_jrmp_stubs.jar:../config/ \
  org.objectweb.jotm.Main -u UserTransaction &
```

(on one line)

And in the `basic/` directory, type

```
$ ant run.rmi.jrmp
```

(by default JOTM is configured to run on RMI/JRMP so you don't have to modify `$JOTM_HOME/config/carol.properties` file to run example on it).

2.3.2 On RMI/IIOP

To run the example on RMI/IIOP, first change the settings of JOTM to activate RMI/IIOP support : in `$JOTM_HOME/config/carol.properties`

- set `carol.rmi.jrmp.activate` to `false`
- set `carol.rmi.jrmp.default` to `false`
- set `carol.rmi.iiop.activate` to `true`
- set `carol.rmi.iiop.default` to `true`

(Now only RMI/IIOP is activated and is the default protocol)

Then type

```
$ tnameserv -ORBInitialPort 19751 &
```

Then in `$JOTM_HOME/lib/`, type

```
$ java -classpath jotm.jar:jotm_iiop_stubs.jar:../config/ \
  org.objectweb.jotm.Main -u UserTransaction &
```

(on one line)

And in the `basic/` directory, type

```
$ ant run.rmi.iiop
```

2.3.3 On both RMI/JRMP and RMI/IIOP

To run the example on both RMI/JRMP and RMI/IIOP, first change the settings of JOTM to activate both of them.
In `$JOTM_HOME/config/carol.properties`

- set `carol.rmi.jrmp.activate` to `true`
- set `carol.rmi.jrmp.default` to `false`
- set `carol.rmi.iiop.activate` to `true`
- set `carol.rmi.iiop.default` to `true`

(Now both RMI/IIOP and RMI/JRMP are activated and RMI/JRMP is the default protocol)

Then type in `$JOTM_HOME/lib/`,

```
$ tnameserver -ORBInitialPort 19751 &
$ rmiregistry -J-classpath -Jjotm.jar:jotm_jrmp_stubs.jar \
  -J-Djava.security.policy=../config/java.policy &
```

Then in `$JOTM_HOME/lib/`, type

```
$ java -classpath \
  jotm.jar:jotm_jrmp_stubs.jar:jotm_iiop_stubs.jar:../config/ \
  org.objectweb.jotm.Main -u UserTransaction &
```

(on one line)

Finally you can acces JOTM on both RMI/JRMP or RMI/IIOP. in the `basic/` directory, type

```
$ ant run.rmi.jrmp
$ ant run.rmi.iiop
$ ant run.rmi.jrmp
$ ...
```

2.4 Output

Whatever configuration, you have chosen, the output of the example is still the same : something like

```
$ ...
$
$ [java] create initial context
$ [java] lookup UserTransaction at : UserTransaction
$
$ [java] a simple transaction wich is committed:
$ [java]        - initial status : STATUS_NO_TRANSACTION
$ [java]        - after begin status : STATUS_ACTIVE
$ [java]        - after commit status : STATUS_NO_TRANSACTION
$
$ [java] a simple transaction which is rolled back.
$ [java] we set a transaction timeout to 1 second, begin the
$ [java] transaction, and wait 5 seconds before commiting it:
$ [java]        - initial status : STATUS_NO_TRANSACTION
$ [java]        - after begin status : STATUS_ACTIVE
$ [java]        - wait for 5 seconds
$ [java]        - after rollback status : STATUS_NO_TRANSACTION
$
$ [java] Basic example is OK.
$
$ ...
```

If you have the message `Basic example is OK.`, the example is working. If it's not the case, double check your JOTM settings. Most of the time, troubles come from incorrect settings (clients try to access JOTM through RMI/JRMP whereas there is no RMI registry but a CORBA name server,...).

3 JDBC Example

JOTM can be used with any database (with a JDBC driver) to provide distributed transactional access to databases.
The JDBC example is a very simple example showing how to use JTA transactions with XAPool to provide transactional access to a database (configuration files for PostgreSQL and MySQL are included).

All Ant commands are to be executed from the `examples/jdbc/` directory of a JOTM *distribution* (examples won't work from JOTM *source* directory).

3.1 Scenario

- a database is started and configured
- a RMI registry is started
- the `JdbcExample` object is started
- it starts a `DatabaseHelper` object with an embedded JOTM. It setups the JDBC objects (i.e. `java.sql.Connection` with XAPool thanks to a configuration file and sets JOTM as their transaction manager. It also binds `UserTransaction` in JNDI
- `JdbcExample` prints a table from the database (without transaction)
- a transaction is started thanks to `UserTransaction`
- an update statement is sent to the database
- the transaction is completed (either commited or rolled back)
- `JdbcExample` prints once again the table from the database (without transaction)

3.2 Setup and compilation

Before starting the example, the database needs to be properly configured. The JDBC example can work with any database providing a JDBC driver. It uses XAPool to take care of the transactional behaviors of JDBC objects. The setup is explained for MySQL. For other databases, it should be straightforward to configure them properly.

3.2.1 Database setup

The example expects:

- a database named `javatest`
- a user of login `mojo` and password `jojo`
- a *transactional* table named `testdata` which looks like

ID	FOO
1	1

  with
  - `id` being an `int` (*prtimary key*)
  - `foo` also being an `int`

  For example on MySQL:

```
mysql> GRANT ALL PRIVILEGES ON *.* TO mojo
    ->   IDENTIFIED BY 'jojo' WITH GRANT OPTION;
mysql> create database javatest;
mysql> use javatest;
mysql> create table testdata (
    ->   id int not null auto_increment primary key,
    ->   foo int)type=bdb;
mysql> insert into testdata values(null, 1);
```

Do not forget to set `testdata` type to `bdb` to enable transaction support. Database configuration are stored in properties file (e.g. `mysql.properties` and `postgresql.properties`) which contains the following properties:

- `driver` - Name of the JDBC driver
- `url` - URL to connect to the data base
- `login` - user login
- `password` - user password

3.2.2 Compilation

In `examples/jdbc/` directory, type

```
$ ant compile
```

to compile the example

3.3 Run the example

Set `JOTM_HOME` to the directory of your JOTM distribution (e.g., `.../jotm/output/dist` from CVS).

To run the example, first check that only RMI protocol will be activated (in the `../../config/carol.properties`, `carol.rmi.activated` should be set to `jrmp`); then type in `$JOTM_HOME/lib/` directory

```
$ rmiregistry -J-classpath -Jjotm.jar:jotm_jrmp_stubs.jar \
  -J-Djava.security.policy=../config/java.policy &
```

to start a RMI registry on default port (i.e. 1099).

Set the classpath

```
$ export CLASSPATH=../../lib/jotm.jar:../../lib/jotm_jrmp_stubs.jar\
  :../../lib/xapool.jar:../../config:.:$JDBC_JARS
```

where `JDBC_JARS` is the location of the JDBC driver jar file(s) you want to use

- `pg73jdbc2.jar` for PostgreSQL 7.3
- `mysql-connector-java-2.0.14-bin.jar` for MySQL 3.23

They are respectively downloadable from

- `http://www.mysql.com/downloads/api-jdbc-stable.html/`
- `http://jdbc.postgresql.org/download/`

Start the example

```
$ java JdbcExample postgresql commit 2
```

to set `foo` value to 2 and commit the transaction on PostgreSQL

```
$ java JdbcExample mysql rollback 0
```

to set `foo` value to 0 but rollback the transaction on MySQL

3.3.1 Usage

```
$ java JdbcExample [database] [completion] [number]
```

where

- `database` can be
  - `postgresql`
  - `mysql` (example will look for a configuration file name `[database].properties`)
- `completion` can be
  - `commit` to commit the transaction
  - `rollback` to rollback the transaction
- `number` has to be a integer

3.4 Output

For example, command line

```
$ java JdbcExample postgresql commit 2
```

will ouptut something like

```
start server

postgresql configuration:
-- listing properties --
login=mojo
url=jdbc:postgresql://localhost/javatest
password=jojo
driver=org.postgresql.Driver
------------------------

create initial context
lookup UserTransaction at : UserTransaction
get a connection
before transaction, table is:
        id      foo
        1       0
begin a transaction
update the table
*commit* the transaction
after transaction, table is:
        id      foo
        1       2
close connection
stop server
JDBC example is ok.
```

As stated, the transaction has been *committed* and `foo` value has been set to 2 in the database.

Another command line like

```
$ java JdbcExample mysql rollback 3
```

will output something like

```
start server

mysql configuration:
-- listing properties --
login=mojo
url=jdbc:mysql://localhost/javatest
password=jojo
driver=org.gjt.mm.mysql.Driver
------------------------

create initial context
```

```
lookup UserTransaction at : UserTransaction
get a connection
before transaction, table is:
        id      foo
        1       1
begin a transaction
update the table
*rollback* the transaction
after transaction, table is:
        id      foo
        1       0
close connection
stop server
JDBC example is ok.
```

Here, the value of `foo` has not been changed in the database because the transaction has been *rolled back*.

4 JMS Example

JOTM can be used with any JMS (Java Message Service) provider to gain advantage of both *message-oriented architecture* and *distributed transactions*.
This example uses JORAM (`http://www.objectweb.org/joram/`) as its JMS provider.

All Ant commands are to be executed from the `examples/jms/` directory of a JOTM *distribution* (examples won't work from JOTM *source* directory).

4.1 Scenario

The **jms** example shows how to use JOTM with a JMS provider (in our case, JORAM) to provide distributed transactional messages.

- a RMI registry is started
- JOTM is started with `UserTransaction` and `TransactionManager` objects accessible through JNDI
- An application (`SimpleJmsXa` class) starts JORAM, setups the JMS objects (`Queue`, `Session`, `ConnectionFactory`) and registers them in JOTM as XA resources
- This application then starts a message sender, `SimpleSender`, and a message receiver, `SimpleReceiver`

On one hand, `SimpleSender` sends 4 messages on a JMS queue:

- one is outside a transaction
- one is inside a transaction with a commit result
- one is inside a transaction with a rollback result
- and the last one with a special text to stop SimpleReceiver

On the other hand, `SimpleReceiver` receives 3 messages from the same JMS queue:

- the one which was outside a transaction
- the one which was inside a transaction with a commit result
- the last one (with the special text)

(SimpleReceiver does not receive the 3rd sent message because it has been rolled back.)

4.2 Setup and compilation

To compile the example, type

```
$ ant compile
```

4.3 Run the example

First, you've to set `JOTM_HOME` to the directory of your JOTM distribution (e.g., `../jotm/output/dist` from CVS).

```
$ export JOTM_HOME=<JOTM_distribution_directory>
```

To run the example, type in `$JOTM_HOME/lib/`

```
$ rmiregistry -J-classpath -Jjotm.jar:jotm_jrmp_stubs.jar \
  -J-Djava.security.policy=../config/java.policy &
```

Then in `$JOTM_HOME/lib/`, type

```
$ java -classpath jotm.jar:jotm_jrmp_stubs.jar:../config/ \
  org.objectweb.jotm.Main \
  -u UserTransaction -m TransactionManager&
```

(on one line)

And in the `jms/` directory, type

```
$ ant run.jms
```

Since the client application of the jms example is a simple RMI/JRMP client, you've to use default protocol configuration for JOTM (i.e RMI/JRMP) in `$JOTM_HOME/config/carol.properties` file.

4.4 Output

the output of the **jms** example should be somethnig like

```
$ ...
$
$ [java] [SimpleJmsXa] lookup the TransactionManager.
$ [java] [SimpleJmsXa] start the JMS server.
$ [java] [SimpleJmsXa] JMS server started.
$ [java] [SimpleJmsXa] create JMS objects, register them in JOTM and bind them.
$ [java] [SimpleJmsXa] JMS objects available.
$ [java] [SimpleJmsXa] start simple sender.
$ [java] [SimpleSender] send : non transactional message
$ [java] [SimpleSender] send : transactional message with commit
$ [java] [SimpleSender] send : transactional message with rollback
$ [java] [SimpleSender] send : LAST message
$ [java] [SimpleJmsXa] start simple receiver.
$ [java] [SimpleReceiver] received: non transactional message
$ [java] [SimpleReceiver] received: transactional message with commit
$ [java] [SimpleReceiver] received: LAST message
$ [java] [SimpleJmsXa] JMS server stopped
$
$ ...
```

If `SimpleSender` has effectively sent 4 messages and `SimpleReceiver` has effectively received only 3 messages, then the **jms** example is working!

5 Tomcat Example

JOTM can be integrated with Tomcat 4.1.x to provide distributed transactional acces to resources from Servlets or JSP.// As for the jdbc example, the Tomcat one can work with any database providing a JDBC driver. It uses XAPool to take care of the transactional behaviors and the pooling of JDBC objects.

5.1 Scenario

The scenario of the Tomcat is very simple. It is based on the example provided by Tomcat in their JNDI Datasource HOW-TO with the addition of some transaction code.

The user send a request to a JSP file (`test.jsp`) which asks to *commit* or *rollback* the incrementation of an integer stored in a database. The JSP delegates the JDBC and transaction code to a JavaBean (`foo.DBTest` class).

The code of `foo.DBTest` is simple:

- a JDBC `Connection` is created from a `DataSource` retrieved through JNDI
- a `UserTransaction` is also retrieved from JNDI
- a transaction is started
- the value of the integer `foo` which is stored in the database is read (SQL query)
- we increment the value of `foo` by 1 in the database (SQL update)
- depending of the choice of the user (`commit` or `rollback`), the transaction is either *committed* or *rolled back*
- we read once more `foo` value from the database (SQL query) and display it in the JSP.

5.2 Setup and compilation

5.2.1 Tomcat setup

The tomcat example uses Tomcat 4.1.18. It can be downloaded from `http://jakarta.apache.org/builds/jakarta-tomcat-4.0/release/v4.1.18/bin/`.
There's no setup needed for Tomcat. You just have to unzip it to use it.

5.2.2 Database setup

The database setup for the Tomcat example is exactly the same than for the JDBC example. Please refer to the Database setup (section 3.2.1) of the JDBC example.
You also need to copy the JDBC driver jar file of your database in the `common/lib/` directory of Tomcat.

5.2.3 Compilation

In `example/tomcat/` directory, type

```
$ ant war
```

to compile Java files and creates a WAR file (`examples/tomcat/output/dbtest.war`) containing your web application and all that is needed to use JOTM.

5.2.4 Deployment

Copy the `output/dbtest.war` file which has just been created to the `webapps/` directory of Tomcat.

Also copy `example/tomcat/dbtest.xml` XML file to the `webapps/` directory of Tomcat.
This files describes the context associated with your web application. In this file, you set the properties to access your database:

- `driverClassName` - Name of the JDBC driver
- `url` - URL to connect to the data base
- `username` - user login
- `password` - user password

By default, `dbtest.xml` is configured to use PostgreSQL as its database.
To use another database, you just have to change these properties (especially `driverClassName` and `url`).

This file also described the resource factories (JDBC and Transaction) used by your web application.

5.2.5 JOTM jar files

You also need to copy JOTM jar files so that Tomcat can see them.
Copy the following jars located in the `lib/` directory of JOTM:

- `jotm.jar`
- `jotm_jrmp_stubs.jar`
- `jonas_timer.jar`
- `carol.jar`
- `jta-spec1_0_1.jar`
- `jts1_0.jar`
- `commons-logging.jar`
- `log4j.jar`
- `objectweb-datasource.jar`
- `xapool.jar`

They have to be placed on `common/lib/` directory of Tomcat.

You've also to copy the file in `config/`

- `trace.properties`

You've have to created a file name `carol.properties` with the following properties:

```
# lmi stands for Local Method Invocation
carol.rmi.activated=lmi

# do not use CAROL JNDI wrapper
carol.start.jndi=false

# do not start a name server
carol.start.ns=false
```

These files have to be placed in `common/classes/` directory of Tomcat.

5.3 Run the example

Go to the `bin/` directory of Tomcat and type

```
$ ./catalina.sh run
```

Use your favorite browser to go to the URL

`http://localhost:8080/dbtest/test.jsp`

Choose if you want to `commit` or `rollback` the incrementation of the value of the integer and click on the `completion` button.

If you've chosen `commit`, the integer value displayed on the page should have been incremented by one.
If you've chosen `rollback`, the integer value displayed on the page should be the same as before.

5.4 Integration of JOTM and Tomcat

For a more technical explanation on the integration of JOTM in Tomcat, please refer to the Tomcat/JOTM HOW-TO.

6 Contacts

If you have some trouble to make the examples work or want to contribute to JOTM, do not hesitate to contact us (`mailto:jotm@objectweb.org`).