JBossJTA Quick Start Guide

by Mark Red Hat Little, Jonathan Red Hat Halliday,
Andrew Red Hat Dinn, and Kevin Red Hat Connor

1. Prerequisites

JBossJTA works in conjunction with the rest of the JBoss Transactions suite. In addition to the documentation here, consult the JBossJTA documentation, which ships as part of JBossJTA and is also available on the JBoss Transaction Service website at *http://www.jboss.org/jbosstm* .

2. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts* [https://fedorahosted.org/liberation-fonts/] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

2.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight keycaps and key combinations. For example:

> To see the contents of the file `my_next_bestselling_novel` in your current working directory, enter the `cat my_next_bestselling_novel` command at the shell prompt and press **Enter** to execute the command.

The above includes a file name, a shell command and a keycap, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from keycaps by the hyphen connecting each part of a key combination. For example:

> Press **Enter** to execute the command.
>
> Press **Ctrl**+**Alt**+**F2** to switch to the first virtual terminal. Press **Ctrl**+**Alt**+**F1** to return to your X-Windows session.

The first paragraph highlights the particular keycap to press. The second highlights two key combinations (each a set of three keycaps with each set pressed simultaneously).

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in `mono-spaced bold`. For example:

File-related classes include `filesystem` for file systems, `file` for files, and `dir` for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialog box text; labeled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

Choose **System** → **Preferences** → **Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).

To insert a special character into a **gedit** file, choose **Applications** → **Accessories** → **Character Map** from the main menu bar. Next, choose **Search** → **Find…** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit** → **Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in proportional bold and all distinguishable by context.

`Mono-spaced Bold Italic` or ***`Proportional Bold Italic`***

Whether mono-spaced bold or proportional bold, the addition of italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

To connect to a remote machine using ssh, type `ssh` *`username@domain.name`* at a shell prompt. If the remote machine is `example.com` and your username on that machine is john, type `ssh john@example.com`.

The `mount -o remount` *`file-system`* command remounts the named file system. For example, to remount the `/home` file system, the command is `mount -o remount /home`.

To see the version of a currently installed package, use the `rpm -q` *`package`* command. It will return a result as follows: *`package-version-release`*.

Note the words in bold italics above — username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> Publican is a *DocBook* publishing system.

2.2. Pull-quote Conventions

Terminal output and source code listings are set off visually from the surrounding text.

Output sent to a terminal is set in `mono-spaced roman` and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in `mono-spaced roman` but add syntax highlighting as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }
}
```

2.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled 'Important' will not cause data loss but may cause irritation and frustration.

Warning

Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

3. We Need Feedback!

You should over ride this by creating your own local Feedback.xml file.

About This Guide

The Stand-alone Quick Start Guide contains information on how to use JBossJTA.

1.1. Audience

This guide is most relevant to engineers who are responsible for administering JBossJTA installations. It is intended for those who are familiar with transactions in general and the OTS/JTS and JTA in particular.

1.2. Prerequisites

Familiarity with the JTA, OTS, transactions and ORBs.

Quick Start to JTA

2.1. Introduction

This chapter will briefly cover the key features required to construct a JTA application. It is assumed that the reader is familiar with the concepts of the JTA.

2.2. Package layout

The key Java packages (and corresponding jar files) for writing basic JTA applications are:

- com.arjuna.ats.jts: this package contains the JBossTS implementations of the JTS and JTA.
- com.arjuna.ats.jta: this package contains local and remote JTA implementation support.
- com.arjuna.ats.jdbc: this package contains transactional JDBC support.

All of these packages appear in the lib directory of the JBossTS installation, and should be added to the programmer's CLASSPATH.

In order to fully utilize all of the facilities available within JBossTS, it will be necessary to add some additional jar files to your classpath. See bin/setup-env.sh or bin\setup-env.bat for details.

2.3. Setting properties

has also been designed to be configurable at runtime through the use of various property attributes. These attributes can be provided at runtime on command line or specified through a properties file.

2.3.1. Specifying the object store location

requires an object store in order to persistently record the outcomes of transactions in the event of failures. In order to specify the location of the object store it is necessary to specify the location when the application is executed; for example:

```
java #DObjectStoreEnvironmentBean.objectStoreDir=/var/tmp/ObjectStore myprogram
```

The default location is a directory under the current execution directory.

By default, all object states will be stored within the defaultStore subdirectory of the object store root, e.g., /usr/local/Arjuna/TransactionService/ObjectStore/defaultStore. However, this subdirectory can be changed by setting the ObjectStoreEnvironmentBean.localOSRoot property variable accordingly.

2.4. Demarcating Transactions

The Java Transaction API consists of three elements: a high-level application transaction demarcation interface, a high-level transaction manager interface intended for application server, and a standard Java mapping of the X/Open XA protocol intended for transactional resource manager. All of the JTA classes and interfaces occur within the javax.transaction package, and the corresponding implementations within the com.arjuna.ats.jta package.

2.4.1. UserTransaction

The UserTransaction interface provides applications with the ability to control transaction boundaries.

In , UserTransaction can be obtained from the static com.arjuna.ats.jta.UserTransaction.userTransaction() method. When obtained the UserTransaction object can be used to control transactions

Example 2.1. User Transaction Example

```
 //get UserTransaction
UserTransaction utx = com.arjuna.ats.jta.UserTransaction.userTransaction();
// start transaction work..
utx.begin();

// perform transactional work

utx.commit();
```

2.4.2. TransactionManager

The TransactionManager interface allows the application server to control transaction boundaries on behalf of the application being managed.

In , transaction manager implementations can be obtained from the static com.arjuna.ats.jta.TransactionManager.transactionManager() method

2.4.3. The Transaction interface

The Transaction interface allows operations to be performed on the transaction associated with the target object. Every top-level transaction is associated with one Transaction object when the transaction is created. The Transaction object can be used to:

- enlist the transactional resources in use by the application.
- register for transaction synchronization call backs.

- commit or rollback the transaction.
- obtain the status of the transaction.

A Transaction object can be obtained using the TransactionManager by invoking the method getTransaction() method.

```
Transaction txObj = TransactionManager.getTransaction();
```

2.5. Local vs Distributed JTA implementations

In order to ensure interoperability between JTA applications, it is recommended to rely on the JTS/OTS specification to ensure transaction propagation among transaction managers.

In order to select the local JTA implementation it is necessary to perform the following steps:

- make sure the property JTAEnvironmentBean.jtaTMImplementation is set to com.arjuna.ats.internal.jta.transaction.arjunacore.TransactionManagerImple.
- make sure the property JTAEnvironmentBean.jtaUTImplementation is set to com.arjuna.ats.internal.jta.transaction.arjunacore.UserTransactionImple.

In order to select the distributed JTA implementation it is necessary to perform the following steps:

- make sure the property JTAEnvironmentBean.jtaTMImplementation is set to com.arjuna.ats.internal.jta.transaction.jts.TransactionManagerImple.
- make sure the property JTAEnvironmentBean.jtaUTImplementation is set to com.arjuna.ats.internal.jta.transaction.jts.UserTransactionImple.

2.6. JDBC and Transactions

ArjunaJTS supports the construction of both local and distributed transactional applications which access databases using the JDBC APIs. JDBC supports two-phase commit of transactions, and is similar to the XA X/Open standard. The JDBC support is found in the com.arjuna.ats.jdbc package.

The ArjunaJTS approach to incorporating JDBC connections within transactions is to provide transactional JDBC drivers through which all interactions occur. These drivers intercept all invocations and ensure that they are registered with, and driven by, appropriate transactions. (There is a single type of transactional driver through which any JDBC driver can be driven. This driver is com.arjuna.ats.jdbc.TransactionalDriver, which implements the java.sql.Driver interface.)

Once the connection has been established (for example, using the java.sql.DriverManager.getConnection method), all operations on the connection will be monitored by . Once created, the driver and any connection can be used in the same way as any other JDBC driver or connection.

connections can be used within multiple different transactions simultaneously, i.e., different threads, with different notions of the current transaction, may use the same JDBC connection. does connection pooling for each transaction within the JDBC connection. So, although multiple threads may use the same instance of the JDBC connection, internally this may be using a different connection instance per transaction. With the exception of close, all operations performed on the connection at the application level will only be performed on this transaction-specific connection.

will automatically register the JDBC driver connection with the transaction via an appropriate resource. When the transaction terminates, this resource will be responsible for either committing or rolling back any changes made to the underlying database via appropriate calls on the JDBC driver.

2.7. Configurable options

The following table shows some of the configuration features, with default values shown in italics. For more detailed information, the relevant section numbers are provided. You should look at the various Programmers Guides for more options.

Note

You need to prefix certain properties in this table with the string com.arjuna.ats.internal.jta.transaction. The prefix has been removed for formatting reasons, and has been replaced by ...

Configuration Name	Possible Values
com.arjuna.ats.jta.supportSubtransactions	*YES* NO
com.arjuna.ats.jta.jtaTMImplementation	...arjunacore.TransactionManagerImple ...jts.TransactionManagerImple
com.arjuna.ats.jta.jtaUTImplementation	...arjunacore.UserTransactionImple ...jts.UserTransactionImple
com.arjuna.ats.jta.xaBackoffPeriod	Time in seconds.
com.arjuna.ats.jdbc.isolationLevel	Any supported JDBC isolation level.

Appendix A. Revision History

Revision History
Revision 1 Wed Apr 13 2010 TomJenkinson<tom.jenkinson@redhat.com>
Taken from installation guide