

JBossJTS 4.15.0

JBossJTS Quick Start Guide



Mark Little

Jonathan Halliday

Andrew Dinn

Kevin Connor

JBossJTS 4.15.0 JBossJTS Quick Start Guide

Author	Mark Little	mlittle@redhat.com
Author	Jonathan Halliday	jhallida@redhat.com
Author	Andrew Dinn	adinn@redhat.com
Author	Kevin Connor	kconnor@redhat.com

Copyright © 2011 JBoss.org.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at <http://creativecommons.org/licenses/by-sa/3.0/>. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, MetaMatrix, Fedora, the Infinity Logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

XFS® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

All other trademarks are the property of their respective owners.

Preface	v
1. Prerequisites	v
2. Document Conventions	v
2.1. Typographic Conventions	v
2.2. Pull-quote Conventions	vi
2.3. Notes and Warnings	vii
3. We Need Feedback!	vii
1. About This Guide	1
1.1. Audience	1
1.2. Prerequisites	1
2. Quick Start to JTS/OTS	3
2.1. Introduction	3
2.2. Package layout	3
2.3. Setting properties	3
2.4. Starting and terminating the ORB and BOA/POA	4
2.5. Specifying the object store location	5
2.6. Implicit transaction propagation and interposition	5
2.7. Obtaining Current	6
2.8. Transaction termination	6
2.9. Transaction factory	7
2.10. Recovery manager	7
A. Revision History	9

Preface

1. Prerequisites

JBossJTS works in conjunction with the rest of the JBoss Transactions suite. In addition to the documentation here, consult the JBossJTS documentation, which ships as part of JBossJTS 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/)¹ 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.

¹ <https://fedorahosted.org/liberation-fonts/>

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!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in the JBoss Issue Tracker: <https://jira.jboss.org/> against the product **JBossJTS**.

When submitting a bug report, be sure to mention the manual's identifier:

JBossJTS_Quick_Start_Guide

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

About This Guide

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

1.1. Audience

This guide is most relevant to engineers who are responsible for administering JBossJTS 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 JTS/OTS

2.1. Introduction

This chapter will briefly cover the key features required to construct a basic OTS application using the raw OTS interfaces defined by the OMG specification. It is assumed that the reader is familiar with the concepts of the JTS/OTS and has read the relevant ORB specific portions of the JBossJTS Programmer's Guide. Further topics and the advanced facilities provided by JBossJTS will be described in subsequent sections of this manual; references to chapters in the other manuals of the document set will be given in the relevant sections.

2.2. Package layout

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

- `com.arjuna.orbportability`: this package contains the classes which constitute the ORB portability library and other useful utility classes.
- `org.omg.CosTransactions`: this package contains the classes which make up the `CosTransactions.idl` module.
- `com.arjuna.ats.jts`: this package contains the JBossJTS implementations of the JTS and JTA.
- `com.arjuna.ats.arjuna`: this package contains further classes necessary for the JBossJTS implementation of the JTS.
- `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 JBossJTS installation, and should be added to the programmer's `CLASSPATH`.

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

2.3. Setting properties

JBossJTS has been designed to be highly configurable at runtime through the use of various property attributes, which will be described in subsequent sections. Although these attributes can be provided at runtime on the command line, it is possible (and may be more convenient) to specify them through the single properties file `JBossJTS-properties.xml`. At runtime JBossJTS looks for its property file in the following order:

- a location specified by a system property, allowing the normal search path to be overridden.
- the current working directory, i.e., where the application was executed from.
- the user's home directory.
- `java.home`
- the `CLASSPATH`, which normally includes the installations etc dir
- A default set of properties embedded in the `.jar` file.

Where properties are defined in both the system properties e.g. -Dfoo=bar and in the properties file, the value from the system property takes precedence. This facilitates overriding individual properties easily on the command line.

2.4. Starting and terminating the ORB and BOA/POA

It is important that JBossJTS is correctly initialized prior to any application object being created. In order to guarantee this, the programmer must use the `initORB` and `initBOA/initPOA` methods of the `ORBInterface` class described in the ORB Portability Manual. Using the `ORB_init` and `BOA_init/create_POA` methods provided by the underlying ORB will not be sufficient, and may lead to incorrectly operating applications. For example:

Example 2.1. Initialize ORB

```
public static void main (String[] args) {  
    ORBInterface.initORB(args, null);  
    ORBInterface.initOA();  
    // . . .  
}
```

The `ORBInterface` class has operations `orb()` and `boa()/poa()/rootPoa()` for returning references to the orb and boa/child POA/root POA respectively after initialization has been performed. If the ORB being used does not support the BOA (e.g., Sun's JDK 1.2) then `boa()` does not appear in the class definition, and `initBOA` will do nothing.

In addition, it is necessary to use `shutdownOA` and `shutdownORB` (in that order) prior to terminating an application to allow JBossJTS to perform necessary cleanup routines. `shutdownOA` routine will either shutdown the BOA or the POA depending upon the ORB being used.

Example 2.2. Shutdown ORB

```
public static void main (String[] args) {  
    // . . .  
    ORBInterface.shutdownOA();  
    ORBInterface.shutdownORB();  
}
```

No further CORBA objects should be used once shutdown has been called. It will be necessary to re-initialise the BOA/POA and ORB in such an event.



Note

In the rest of this document we shall use the term Object Adapter to mean either the Basic Object Adapter (BOA) or the Portable Object Adapter (POA). In addition, where possible we shall use the ORB Portability classes which attempt to mask the differences between POA and BOA.

2.5. Specifying the object store location

JBossJTS 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.

2.6. Implicit transaction propagation and interposition

Transactions can be created within one domain (e.g., process) and used within another. Therefore, information about a transaction (the transaction context) needs to be propagated between these domains. This can be accomplished in two ways:

- *Explicit propagation* means that an application propagates a transaction context by passing context objects (instances of the Control interface or the PropagationContext structure) defined by the Transaction Service as explicit parameters. Note, for efficiency reasons it is recommended that the PropagationContext be passed, rather than the Control.
- *Implicit propagation* means that requests on objects are implicitly associated with the client's transaction; they share the client's transaction context. The context is transmitted implicitly to the objects, without direct client intervention.

OTS objects supporting the Control interface are standard CORBA objects. When the interface is passed as a parameter in some operation call to a remote server only an object reference is passed. This ensures that any operations that the remote object performs on the interface (such as registering resources as participants within the transaction) are performed on the real object. However, this can have substantial penalties for the application if it frequently uses these interfaces due to the overheads of remote invocation. To avoid this overhead JBossJTS supports interposition, whereby the server creates a local object which acts as a proxy for the remote transaction and fields all requests that would normally have been passed back to the originator. This surrogate registers itself with the original transaction coordinator to enable it to correctly participate in the termination of the transaction. Interposed coordinators effectively form a tree structure with their parent coordinators. This is shown in the figure below.

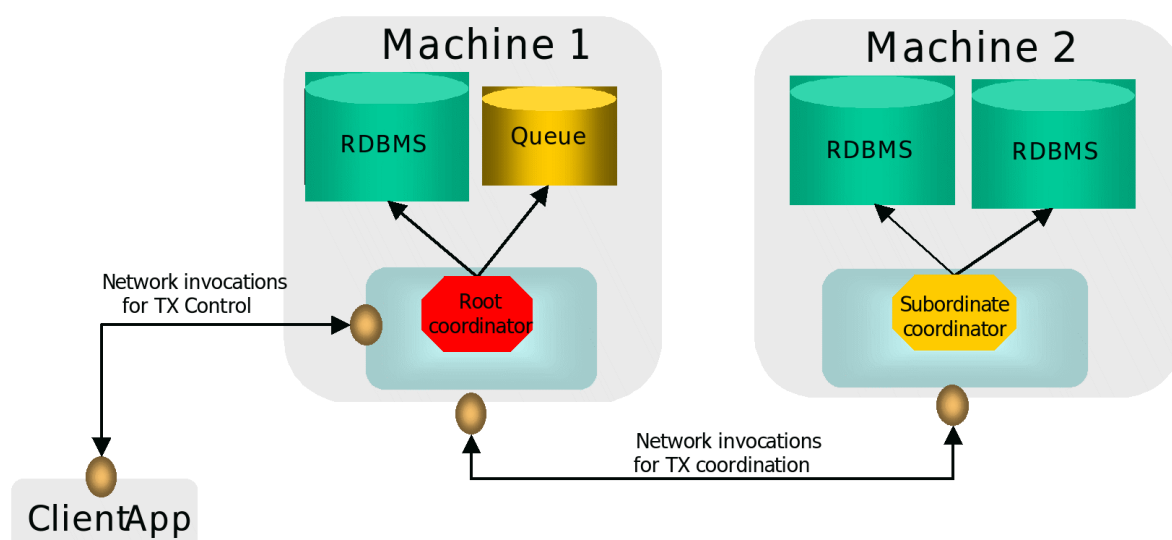


Figure 2.1. Interposition relationship

**Note**

implicit transaction propagation does not imply interposition will be used at the server, but (typically) interposition requires implicit propagation.

If implicit context propagation and interposition are required, then the programmer must ensure that JBossJTS is correctly initialised prior to objects being created; obviously it is necessary for both client and server to agree on which, if any, protocol (implicit or interposition) is being used. Implicit context propagation is only possible on those ORBs which either support filters/interceptors, or the CostSPPortability interface. Currently this is JacORB and the JDK miniORB. Depending upon which type of functionality is required, the programmer must perform the following:

- Implicit context propagation:
set the `JTSEnvironmentBean.contextPropMode` property variable to `CONTEXT`.
- Interposition:
set the `JTSEnvironmentBean.contextPropMode` property variable to `INTERPOSITION`.

If using the JBossJTS advanced API then interposition is *required*.

2.7. Obtaining Current

The Current pseudo object can be obtained from the `com.arjuna.ats.jts.OTSManager` class using its `get_current()` method.

2.8. Transaction termination

It is implementation dependant as to how long a Control remains able to access a transaction after it terminates. In JBossJTS, if using the Current interface then all information about a transaction is destroyed when it terminates. Therefore, the programmer should not use any Control references to the transaction after issuing the commit/rollback operations.

However, if the transaction is terminated explicitly using the Terminator interface then information about the transaction will be removed when all outstanding references to it are destroyed. However, the programmer can signal that the transaction information is no longer required using the `destroyControl` method of the `OTS` class in the `com.arjuna.CosTransactions` package. Once the program has indicated that the transaction information is no longer required, the same restrictions to using Control references apply as described above.

2.9. Transaction factory

By default, JBossJTS does not use a separate transaction manager when creating transactions through the Current interface. Each transactional client essentially has its own transaction manager (`TransactionFactory`) which is co-located with it. By setting the `com.arjuna.ats.jts.transactionManager` property variable to YES this can be overridden at runtime. The transaction factory is located in the `bin` directory of the JBossJTS distribution, and should be started by executing the `start-transaction-service` script located in `<ats_root>/bin`.

Typically Current locates the factory using the `CosServices.cfg` file. This file is similar to `resolve_initial_references`, and is automatically updated (or created) when the transaction factory is started on a particular machine. This file must be copied to the installation of all machines which require to share the same transaction factory. `CosServices.cfg` is located at runtime by the `OrbPortabilityEnvironmentBean` properties `initialReferencesRoot` (a directory, defaulting to the current working directory) and `initialReferencesFile` (a name relative to the directory, '`CosServices.cfg`' by default).

It is possible to override the default location mechanism by using the `OrbPortabilityEnvironmentBean.resolveService` property variable. This can have one of the following values:

- `CONFIGURATION_FILE`: the default, this causes the system to use the `CosServices.cfg` file.
- `NAME_SERVICE`: JBossJTS will attempt to use a name service to locate the transaction factory. If this is not supported, an exception will be thrown.
- `BIND_CONNECT`: JBossJTS will use the ORB-specific bind mechanism. If this is not supported, an exception will be thrown.

If `OrbPortabilityEnvironmentBean.resolveService` is specified when the transaction factory is run, then the factory will register itself with the specified resolution mechanism.

2.10. Recovery manager

You will need to start the recovery manager subsystem to ensure that transactions are recovered despite failures. In order to do this, you should run the `start-recovery-manager` script in `<ats_root>/bin`.

Appendix A. Revision History

Revision 1 **Wed Apr 13 2010**

Tom Jenkinson
tom.jenkinson@redhat.com

Taken from installation guide

