JBoss Enterprise Application Platform Common Criteria Certification 5

Transactions Administrators Guide

for use with JBoss Enterprise Application Platform 5 Common Criteria Certification Edition 5.1.0

Last Updated: 2017-11-20

JBoss Enterprise Application Platform Common Criteria Certification 5 Transactions Administrators Guide

for use with JBoss Enterprise Application Platform 5 Common Criteria Certification
Edition 5.1.0

Mark Little
mlittle@redhat.com

Legal Notice

Abstract

This book is the JBoss Transactions Administrators Guide for JBoss Enterprise Application Platform 5.1.0.

Table of Contents

CHAPTER 1. INTRODUCTION

JBoss Transaction Service generates few administrative tasks. It relies on proper functioning of the underlying operating system and infrastructure. As the administrator, keep the following things in mind:

1. JBoss Transaction Service does not provide a security layer. The objects stored in the JBoss Transactions Object Store are typically owned by the user running the application that created them. The Object Store and Object Manager facilities do not enforce ownership of objects. Ownership of objects is not checked or enforced by the Transaction Manager.

2. Persistent objects created in the Object Store are never deleted unless the **`StateManager.destroy`** method is invoked on an object, or an application explicitly deletes them. This means that the Object Store gradually accumulates garbage, especially during application development and testing phases. JBoss Transaction Service provides no automated garbage collection facility. This lack of garbage collection can create dangling references. Here is an example of a dangling reference. A persistent object called ObjectA stores a Uid for ObjectB, which is also a persistent object, in its passive representation on disk. An application can delete ObjectB even though ObjectA still contains a reference to it. When ObjectA is next activated and attempts to access ObjectB, a run-time error occurs.

3. JBoss Transaction Service includes no version control of objects or database reconfiguration in the event of class structure changes. If you change the definition of a class of persistent objects, you must ensure that existing instances of the object in the Object Store are converted to the new representation. The JBoss Transactions Service software cannot detect or correct references to old object state by new operation versions or vice versa.

4. Object store management is critically important to the transaction service.

CHAPTER 2. OBJECTSTORE MANAGEMENT

Within the Transaction Service installation, the Object Store is updated regularly, whenever transactions are created, or when Transactional Objects for Java is used. In a failure-free environment, the only object states within the object store are those representing objects created with the Transactional Objects for Java API.

However, if failures occur, transaction logs may remain in the object store until crash recovery facilities have resolved the transactions they represent. Therefore, it is very important that the contents of the object store are not deleted inadvertently, as this will make it impossible to resolve in-doubt transactions. In addition, if multiple users share the same object store, they must understand that it is not an exclusive resource, and not delete transaction logs without careful consideration.

CHAPTER 3. OTS AND J2EE TRANSACTION SERVICE MANAGEMENT

3.1. STARTING THE RUN-TIME SYSTEM

Run-time support for the JBoss Transaction Service consists of run-time packages and the OTS Transaction Manager server. By default JBoss Transaction Service does not use a separate Transaction Manager server. Instead, transaction managers are co-located with each application process. This improves performance and application fault-tolerance by removing external dependencies of applications upon other services for proper function.

If your application requires a separate transaction manager, set the **`com.arjuna.ats.jts.transactionManager`** environment variable to **`yes`**. The system locates the transaction manager using an ORB-specific mechanism. It might be registered with a name server, added to the ORB's initial references, listed in a references file specific to JBoss Transaction Service, or located by the ORB's specific location mechanism.

You can override the default registration mechanism by setting the **`com.arjuna.orbportability.resolveService`** environment variable to one of the following values:

`CONFIGURATION_FILE`, the default value

causes the system to use the **`CosServices.cfg`** file.

`NAME_SERVICE`

JBoss Transaction Service attempts to use a name service to register the transaction factory. If this is not supported by the ORB, an exception will be thrown.

`BIND_CONNECT`

JBoss Transaction Service uses the ORB-specific bind mechanism. If this is not supported by the ORB, an exception will be thrown.

`RESOLVE_INITIAL_REFERENCES`

JBoss Transaction Service attempts to register the transaction service with the ORB's initial service references.

3.2. OTS CONFIGURATION FILE

Similar to the **`RESOLVE_INITIAL_REFERENCES`** option, JBoss Transaction Service supports an initial reference file where references for specific services can be stored and used at run-time. The file, **`CosServices.cfg`**, consists of two columns: the **`service name`** (which is always **`TransactionService`** if you are using OTS server) and the IOR, separated by a single space. **`CosServices.cfg`** normally resides in the **`etc`** directory of the JBoss Transaction Service installation, although the actual location is determined at run-time by searching the **`CLASSPATH`** for a copy of the file in an **`etc`** directory or the location of the TransactionService properties file directory.

The OTS server creates this file if not found, and registers itself within the file. Stale information is automatically removed. Machines sharing the same transaction server need access to either this file itself, or a copy of it.

You can override the name and location of the file using the **com.arjuna.orbportability.initialReferencesFile** and **com.arjuna.orbportability.initialReferencesRoot** variables, respectively.

```
com.arjuna.orbportability.initialReferencesFile=myFile
com.arjuna.orbportability.initialReferencesRoot=c:\\temp
```

3.3. NAME SERVICE

If your ORB supports a name service, and JBoss Transaction Service has been configured to use it, the transaction manager will automatically be registered with it.

NOTE

This option is not used for JacORB

3.4. RESOLVE_INITIAL_REFERENCES

Currently this option is only supported for JacORB.

3.5. RESOLUTION SERVICE TABLE

The following table summarizes the different ways in which the OTS transaction manager may be located on specific ORBs:

Table 3.1. Locating the OTS transaction manager server

Resolution Mechanism	ORB
OTS configuration file	All available ORBs
Name Service	JacORB
resolve_initial_references	JacORB

CHAPTER 4. XA SPECIFIC MANAGEMENT

Each XA Xid that JBoss Transaction Service creates needs a unique node identifier encoded within it. JBoss Transaction Service can only recover transactions and states that match a specified node identifier, which is passed to JBoss Transaction Service via the **`com.arjuna.ats.arjuna.xa.nodeIdentifier`** property. This value must be unique across all your JBoss Transaction Service instances. If no value is given, JBoss Transaction Service will generate one and report the value via the logging infrastructure.

4.1. XA RECOVERY

When running XA recovery, JBoss Transaction Service must be told which types of Xid it can recover. The node identifier to use should be provided to JBoss Transaction Service in a property that starts with the name **`com.arjuna.ats.jta.xaRecoveryNode`**. You can pass multiple values. A value of **`*`** forces JBoss Transaction Service to recover (and possibly rollback) all transactions, regardless of their node identifier. Use this option with extreme caution.

More information about recovery can be found in the Failure and Recovery chapter.

CHAPTER 5. WEB SERVICE TRANSACTION SERVICE MANAGEMENT

The basic building blocks of a transactional Web Services application are:

- the application itself
- the Web services that the application consumes
- the Transaction Manager
- the transaction participants which support those Web services

In a typical deployment, a single developer is not likely to support all of these roles. An overview is presented because developers often produce services, or applications that consume services, while system administrators run the transaction-management infrastructure.

5.1. THE TRANSACTION MANAGER

The transaction manager is a Web service which coordinates JBoss Transaction Service transactions. It is the only software component in the JBoss Transaction Service that is meant to be run directly as a network service, instead of supporting end-user code. The transaction manager runs as a JAXM request/response Web service.

IMPORTANT

When starting up an application server instance that has JBoss Transaction Service deployed within it, you may see various error messages in the console or log. Here is one such message:

```
16:53:38,850 ERROR [STDERR] Message Listener Service: started,
message listener jndi name activationcoordinator
```

Messages like these are for information purposes, not actual errors.

5.1.1. Configuring the Transaction Manager

The Transaction Manager and related infrastructure are configured by means of properties files:

- **`wscf.xml`**
- **`wst.xml`**
- **`wstx.xml`**

These files are located in the **`conf`** directory and are used to configure the demo application and the standalone module.

For the most part, the default values in these files do not need to be altered. However, the com.arjuna.ats.arjuna.objectstore.objectStoreDir property determines the location of the persistent store used to record transaction state. The default value of **`C:/temp/ObjectStore`** should be changed to a value appropriate to your system. In a production environment, this directory should be located on fault-tolerant media, such as a RAID array.

When a standalone coordinator is being used by an application, you must enable and modify two additional properties in the **wstx.xml** file:

- com.arjuna.mw.wst.coordinatorURL
- com.arjuna.mw.wst.terminatorURL

These properties represent the URLs used by the client application to contact the standalone coordinator. They should be configured with the correct host name and port of the standalone coordinator.

NOTE

JBoss Transaction Service is highly modular. In order to allow for flexible deployment of individual components, the same property values are sometimes needed in more than one configuration file. For the majority of configurations, you should maintain consistent values for properties that are defined in more than one file.

5.1.2. Deploying the Transaction Manager

The Web Service Transaction Manager component of the Transaction Service consists of a number of JAR files which contain the application's class files, as well as Web service (WAR) files which expose the necessary services. Programmers include all these components in their application EAR file during application development, to simplify deployment of the transaction infrastructure. In production, you can install the Transaction Manager as a stand-alone application, to centralize configuration and management at the server level. The demonstration application shipped with JBoss Transaction Server includes a sample deployment descriptor which includes the Transaction Manager components in an application.

JBoss Transaction Service uses fixed endpoints for its underlying protocol communication. Therefore, problems may arise if multiple applications using the Transaction Service are deployed to the same server concurrently. To deploy several transactional applications in the same server, deploy the Transaction Manager as a separate application, rather than embedding it within the deployment of individual applications.

The coordinator directory in the JBoss Transaction Service installation assists in the configuration and deployment of a stand-alone transaction manager. In order to use this, you must:

- Have JBoss Enterprise Application Platform 5 installed
- Install ant 1.4 or later.

IMPORTANT

The application server installation must be different from the one that clients and services are deployed into. Otherwise, conflicts may occur between the various JBoss Transaction Service components.

Edit the **build.xml** included with the coordinator, so that it points to the application server installation where the transaction coordinator will be deployed, and the location of the JBoss Transaction Service installation. The files **ws-c_jaxm_web-app.xml** and **ws-t_jaxm_web-app.xml** in the **dd/** directory of the coordinator are the deployment descriptors for the WS-C and WS-T WAR files. They contain templated URLs. During the build phase, **ant** substitutes the **hostname** and **port** values from the **build.xml** into these files.

Run **ant**, with ***target deploy-weblogic***, ***deploy-jboss*** or ***deploy-webmethods***, to create and deploy a new coordinator into the correct application server installation.

Next, point your client at the required coordinator, by generating the demo application and specifying the port and hostname of the coordinator.

5.1.3. Deployment descriptors

It is not generally necessary to change the contents of the various deployment descriptors used by JBoss Transaction Service. However, if you do need to modify them they are all included in the coordinator module.

Not all JBoss Transaction Service components have ready access to the information in the deployment descriptors. Therefore, if you modify the JNDI names used by any of the WS-C or WS-T deployment descriptors, you may need to inform other JBoss Transaction Service components at run-time, by setting an appropriate property in the **wstx.xml** configuration file.

The Table 5.1, “Deployment descriptor values and properties” table shows the default JNDI names used by the deployment descriptors and the corresponding property to set if the default value is changed.

Table 5.1. Deployment descriptor values and properties

JNDI Name	Property
Activationrequester	com.arjuna.mw.wst.at.activationrequester
Activationcoordinator	com.arjuna.mw.wst.at.activationcoordinator
Completionparticipant	com.arjuna.mw.wst.at.completionparticipant
Registrationrequester	com.arjuna.mw.wst.at.registrationrequester
durable2pcdispatcher	com.arjuna.mw.wst.at.durable2pcdispatcher
durable2pcparticipant	com.arjuna.mw.wst.at.durable2pcparticipant
volatile2pcdispatcher	com.arjuna.mw.wst.at.volatile2pcdispatcher
volatile2pcparticipant	com.arjuna.mw.wst.at.volatile2pcparticipant
businessagreementwithparticipantcompletiondispatcher	com.arjuna.mw.wst.ba.businessagreementwpcdispatcher
businessagreementwithparticipantcompletionparticipant	com.arjuna.mw.wst.ba.businessagreementwpcparticipant
businessagreementwithcoordinatorcompletiondispatcher	com.arjuna.mw.wst.ba.businessagreementwccdispatcher
businessagreementwithcoordinatorcompletionparticipant	com.arjuna.mw.wst.ba.businessagreementwccparticipant

CHAPTER 6. JBOSS TRANSACTION SERVER RUN-TIME INFORMATION

Each JBoss Transaction Server module contains an **`Info`** class. This class provides a **`toString`** method, which returns an XML dump of the configuration information for the module in question. See Example 6.1, “Using the **`toString`** Method” for an example of the output.

Example 6.1. Using the `toString` Method

```
<module-info>
  <source-identifier>unknown</source-identifier>
  <build-information>
    Arjuna Technologies [mlittle] (Windows 2000 5.0)
  </build-information>
  <version>unknown</version>
  <date>2002/06/15 04:06 PM</date>
  <notes></notes>
  <configuration>
    <properties-file dir="null">arjuna.properties</properties-file>
    <object-store-root>null</object-store-root>
  </configuration>
</module-info>
```

CHAPTER 7. RESOURCE RECOVERY IN JBOSS TRANSACTION SERVICE

7.1. INTRODUCTION

JBoss Transaction Service is a crash tolerant transaction manager. When enlisting XAResources such as JDBC connections defined with <xa-datasource>, or JMS connections using JBoss Messaging in a 2-phase transaction, JBoss Transaction Service keeps a transaction log that allows recovery if the application server crashes during a transaction. If an appropriate recovery module is configured, most failed transactions can be recovered automatically when all resources become available again.

7.2. ASSUMPTIONS

The configuration options mentioned here are contained by default in the **jbossts-properties.xml** file, located in your server configuration's **conf** directory. For a server installed at *JBOSS_HOME* using the *default* configuration, the correct file path is: ***JBOSS_HOME*/server/*default*/conf/jbossts-properties.xml**

7.3. A NOTE ABOUT CLUSTERS

Each application server instance is responsible for recovering transactions it was coordinating. Commonly, a single database serves multiple application servers, and thus participates in transactions from multiple coordinators. During recovery, JBoss Transaction Service requests a list of in-doubt transactions which it can potentially recover, from each application server. The database returns *all* in-doubt transactions, including ones that may not have been coordinated by a given instance. To effectively separate each node's transactions, you must configure a unique node id for each application server instance that shares a common database, by setting a unique value for the following property:

```
<property name="com.arjuna.ats.arjuna.xa.nodeIdentifier" value="1"/>
```

You also need an element that indicates what node needs the recovery. This needs to match the **nodeIdentifier** configured above.

```
<property name="com.arjuna.ats.jta.xaRecoveryNode" value="1"/>
```

7.4. RECOVERY MODULES

Each XA resource for which recovery is desired needs a corresponding recovery module configured in the "jta" section of **jbossjta-properties.xml**. Each recovery module must extend **com.arjuna.ats.jta.recovery.XAResourceRecovery**. We provide implementations for JDBC and JMS XA resources.

7.4.1. JDBC Recovery

JBoss Enterprise Application Platform now includes recovery auto-registration in the JCA. Thus, the AppServerJDBCXARecovery which was used in previous releases is disabled by default, and will be removed entirely from future releases of the Platform.

7.4.1.1. Vendor-Specific Database Information

Oracle

If Oracle is configured incorrectly, you will experience the following error in your log files:

```
  WARN  [com.arjuna.ats.jta.logging.loggerI18N]
[com.arjuna.ats.internal.jta.recovery.xarecovery1] Local
XARecoveryModule.xaRecovery  got XA exception
javax.transaction.xa.XAException, XAException.XAER_RMERR
```

To resolve this error, be sure that the Oracle user has access to the appropriate tables to accomplish the recovery:

```
GRANT SELECT ON sys.dba_pending_transactions TO user;
GRANT SELECT ON sys.pending_trans$ TO user;
GRANT SELECT ON sys.dba_2pc_pending TO user;
GRANT EXECUTE ON sys.dbms_xa TO user;
```

The above assumes that *user* is the user defined to connect from JBoss to Oracle. It also assumes that either Oracle 10g R2 (patched for bug 5945463) or 11g is in use. If an unpatched version prior to 11g is used then change the last GRANT EXECUTE to:

```
GRANT EXECUTE ON sys.dbms_system TO user;
```

PostgreSQL

See the PostgreSQL documentation for instructions on enabling prepared (i.e. XA) transactions. Version 8.4-701 of PostgreSQL's JDBC driver has a bug in **`org.postgresql.xa.PGXAConnection`** which breaks recovery in certain situations. This is fixed in newer versions.

MySQL

Based on http://bugs.mysql.com/bug.php?id=12161, XA transaction recovery does not appear to be possible using MySQL. This is scheduled to be addressed in MySQL 6.1. See also JBPAPP-2576 in the release notes for JBoss Enterprise Application Platform 5.

DB2

DB2 expects **`XAResource.recover`** calls only during designated resynchronization stage which occurs when application server is restarted after crash/failure. This is a design flaw in DB2, and out of the scope of this documentation.

7.4.2. JMS Recovery

Refer to the JBoss Messaging Guide for recovery as it relates to Messaging. These guides are available in the suite of documentation for the Enterprise Application Platform, on http://docs.redhat.com..

7.5. NOTES FOR JMS CLUSTERS

When one node in JBoss Messaging cluster goes down, its buddy in the cluster loads all of the dead server's messages from the database.

If the dead node was performing an XA transaction when it went down, the transaction log may have been written but the associated messages may move to another server in the cluster.

When the dead server comes back to life, the recovery manager may try to recover the transactions

stored in the transaction log. If the messages have been moved to another server, it is impossible to acquire the proper XAResource from the local JMS provider, because the associated messages are no longer on that server. The result is that JBoss Transaction Service returns:

```
      Could not find new XAResource to use for recovering non-serializable
XAResource
```

To resolve this, add a JMS provider and a Recovery Manager for each node in the cluster. For example, if the cluster had three nodes, add this to **jbossts-properties.xml**:

```
      <property
name="com.arjuna.ats.jta.recovery.XAResourceRecovery.JBMESSAGINGREMOTE1"

value="org.jboss.jms.server.recovery.MessagingXAResourceRecovery;java:/Rem
oteJMSProvider1"/>

      <property
name="com.arjuna.ats.jta.recovery.XAResourceRecovery.JBMESSAGINGREMOTE2"

value="org.jboss.jms.server.recovery.MessagingXAResourceRecovery;java:/Rem
oteJMSProvider2"/>
```

The remote providers can be configured in ***JBOSS_HOME*/server/default/deploy/jms-ds.xml**:

```
<properties depends="arjuna" name="jta">
  <!--
      Support subtransactions in the JTA layer?
      Default is NO.
  -->
  <property name="com.arjuna.ats.jta.supportSubtransactions" value="NO"/>
  <property name="com.arjuna.ats.jta.jtaTMImplementation"

value="com.arjuna.ats.internal.jta.transaction.arjunacore.TransactionManag
erImple"/>
  <property name="com.arjuna.ats.jta.jtaUTImplementation"

value="com.arjuna.ats.internal.jta.transaction.arjunacore.UserTransactionI
mple"/>
  <!--
      *** Add this line to enable recovery for JMS resources using
DefaultJMSProvider ***
  -->
  <property
name="com.arjuna.ats.jta.recovery.XAResourceRecovery.JBMESSAGING1"

value="org.jboss.jms.server.recovery.MessagingXAResourceRecovery;java:/Def
aultJMSProvider"/>

</properties>
```

The JNDI properties are configured to connect to the remote nodes in the cluster. Add providers and recovery managers to each node in the cluster for all the other nodes of the cluster in order to get proper recovery.

CHAPTER 8. SELECTING THE JTA IMPLEMENTATION

Two variants of the JTA implementation are provided and accessible through the same interface. These are:

1. A local JTA, which only allows non-distributed JTA transactions to be executed. This is the only version available with the JBoss Transaction Service.

2. A remote, CORBA-based JTA, which allows distributed JTA transactions to be executed. This version is only available with the ArjunaJTS product and requires a supported CORBA ORB.

Both of these implementations are fully compatible with the transactional JDBC driver provided with the JBoss Transaction Service.

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

1. Set the **`com.arjuna.ats.jta.jtaTMImplementation`** property to **`com.arjuna.ats.internal.jta.transaction.arjunacore.TransactionManagerImple`**.

2. Set the **`com.arjuna.ats.jta.jtaUTImplementation`** to **`com.arjuna.ats.internal.jta.transaction.arjunacore.UserTransactionImple`**.

These settings are the default values and do not need to be set to use the local implementation.

CHAPTER 9. ORB SPECIFIC CONFIGURATIONS

JacORB

For JacORB to function correctly, ensure there is a valid **`jacorb.properties`** or **`.jacorb_properties`** file in one of the following places:

- The CLASSPATH.
- The home directory of the user running the JBoss Transaction Service. The home directory is retrieved using **`System.getProperty( “user.home” );`**
- The current directory.
- The **`lib`** directory of the JDK used to run your application. This is retrieved using **`System.getProperty( “java.home” );`**

The above places are searched in the order given. A template **`jacorb.properties`** file can be found in the JacORB installation directory.

The JacORB properties file contains two important properties which must be configured appropriately for your application, they are:

- jacorb.poa.thread_pool_max
- jacorb.poa.thread_pool_min

These properties specify the minimum and maximum number of request processing threads that JacORB will use in its thread pool. If too few threads are available, the application may become deadlocked. For more information on configuring JacORB please reference the JacORB documentation.

NOTE

JacORB comes with its own implementation of the classes defined in the **`CosTransactions.idl`** file. Unfortunately these are incompatible with the version shipped with JBoss Transaction Service. Therefore, the JBoss Transaction Service JAR files must appear in the **`CLASSPATH`** before any JacORB JARs.

The recovery manager must always use the same well-known port for each machine on which it runs. You should not use the **`OAPort`** property provided by JacORB unless the recovery manager has its own **`jacorb.properties`** file or the port is provided on the command line when starting the recovery manager. If the recovery manager and other components of JBoss Transaction Service share the same **`jacorb.properties`** file, then you should use the **`com.arjuna.ats.jts.recoveryManagerPort`** and **`com.arjuna.ats.jts.recoveryManagerAddress`** properties.

CHAPTER 10. INITIALIZING JBOSS TRANSACTION SERVICE APPLICATIONS

JBoss Transaction Service needs to be correctly initialized before any application object is created. To guarantee this, use the **`ORB_init`** and **`create_POA`** methods.

CHAPTER 11. ERRORS AND EXCEPTIONS

Errors and Exceptions During Transactional Applications

NO_MEMORY

The application has run out of memory, and thrown an **`OutOfMemoryError`** exception. JBoss Transactions has attempted to do some garbage collection before re-throwing the exception. This is sometimes a transient problem and retrying the invocation might succeed.

com.arjuna.ats.arjuna.exceptions.FatalError

The transaction system has encountered a fatal error and must shut down. Prior to this error, the transaction service ensures that all running transactions have rolled back. If caught, the application should tidy up and exit. If further work is attempted, application consistency may be violated.

com.arjuna.ats.arjuna.exceptions.LicenceError

An attempt has been made to use the transaction service in a manner inconsistent with the current license. The transaction service will not allow further forward progress for existing or new transactions.

com.arjuna.ats.arjuna.exceptions.ObjectStoreError

An error occurred while the transaction service attempted to use the object store. Further forward progress is not possible.

Object store warnings about access problems on states

This error may occur during the normal execution of crash recovery, as the result of multiple concurrent attempts to perform recovery on the same transaction. It can be safely ignored.

APPENDIX A. REVISION HISTORY

Revision 5.1.0-113.400 **2013-10-31** **Rüdiger Landmann**

Rebuild with publican 4.0.0

Revision 5.1.0-113 **2012-07-18** **Anthony Towns**

Rebuild for Publican 3.0

Revision 5.1.0-112 **Wed Sep 15 2010** **Misty Stanley-Jones**

Changed version number in line with new versioning requirements.

Revised for JBoss Enterprise Application Platform 5.1.0.GA.