◆ ◆ ◆ CHAPTER 18

Administering Transactions

This chapter discusses how to manage the transaction service for the GlassFish Server Open Source Edition environment by using the `asadmin` command-line utility. Instructions for manually recovering transactions are also included.

The following topics are addressed here:

Instructions for accomplishing the tasks in this chapter by using the Administration Console are contained in the Administration Console online help.

For more information about the Java Transaction API (JTA) and Java Transaction Service (JTS), see the following sites: `http://java.sun.com/javaee/technologies/jta/index.jsp` and `http://java.sun.com/javaee/technologies/jts/index.jsp`.

You might also want to read Chapter 28, "Transactions," in *The Java EE 6 Tutorial*.

About Transactions

A *transaction* is a series of discreet actions in an application that must all complete successfully. By enclosing one or more actions in an indivisible unit of work, a transaction ensures data integrity and consistency. If all actions do not complete, the changes are rolled back.

For example, to transfer funds from a checking account to a savings account, the following steps typically occur:

1. Check to see if the checking account has enough money to cover the transfer.
2. Debit the amount from the checking account.

3. Credit the amount to the savings account.
4. Record the transfer to the checking account log.
5. Record the transfer to the savings account log.

These steps together are considered a single transaction.

If all the steps complete successfully, the transaction is *committed*. If any step fails, all changes from the preceding steps are rolled back, and the checking account and savings account are returned to the states they were in before the transaction started. This type of event is called a *rollback*. A normal transaction ends in either a committed state or a rolled back state.

The following elements contribute to reliable transaction processing by implementing various APIs and functionalities:

- **Transaction Manager.** Provides the services and management functions required to support transaction demarcation, transactional resource management, synchronization, and transaction context propagation.
- **GlassFish Server.** Provides the infrastructure required to support the application runtime environment that includes transaction state management.
- **Resource Manager.** Through a resource adapter, the resource manager provides the application access to resources. The resource manager participates in distributed transactions by implementing a transaction resource interface used by the transaction manager to communicate transaction association, transaction completion, and recovery work. An example of such a resource manager is a relational database server.
- **Resource Adapter.** A system-level software library is used by GlassFish Server or a client to connect to a resource manager. A resource adapter is typically specific to a resource manager. The resource adapter is available as a library and is used within the address space of the client using it. An example of such a resource adapter is a Java Database Connectivity (JDBC) driver. For information on supported JDBC drivers, see "Configuration Specifics for JDBC Drivers" on page .
- **Transactional User Application.** In the GlassFish Server environment, the transactional user application uses Java Naming and Directory Interface (JNDI) to look up transactional data sources and, optionally, the user transaction). The application might use declarative transaction attribute settings for enterprise beans, or explicit programmatic transaction demarcation. For more information, see "The Transaction Manager, the Transaction Synchronization Registry, and UserTransaction" in *GlassFish Server Open Source Edition 3.1 Application Development Guide*.

The following topics are addressed here:

- "Transaction Resource Managers" on page 23
- "Transaction Scope" on page 23

Transaction Resource Managers

There are three types of transaction resource managers:

- Databases - Use of transactions prevents databases from being left in inconsistent states due to incomplete updates. For information about JDBC transaction isolation levels, see "Using JDBC Transaction Isolation Levels" in *GlassFish Server Open Source Edition 3.1 Application Development Guide*.

  The GlassFish Server supports a variety of JDBC XA drivers. For a list of the JDBC drivers currently supported by the GlassFish Server, see the *GlassFish Server Open Source Edition 3.1 Release Notes*. For configurations of supported and other drivers, see "Configuration Specifics for JDBC Drivers" on page .
- Java Message Service (JMS) Providers - Use of transactions ensures that messages are reliably delivered. The GlassFish Server is integrated with Open Message Queue, a fully capable JMS provider. For more information about transactions and the JMS API, see Chapter 16, "Administering the Java Message Service (JMS)."
- J2EE Connector Architecture (CA) components - Use of transactions prevents legacy EIS systems from being left in inconsistent states due to incomplete updates. For more information about connectors, see Chapter 12, "Administering EIS Connectivity."

Transaction Scope

A *local* transaction involves only one non-XA resource and requires that all participating application components execute within one process. Local transaction optimization is specific to the resource manager and is transparent to the Java EE application.

In the GlassFish Server, a JDBC resource is non-XA if it meets either of the following criteria:

- In the JDBC connection pool configuration, the DataSource class does not implement the javax.sql.XADataSource interface.
- The Resource Type setting is not set to javax.sql.XADataSource.

A transaction remains local if the following conditions remain true:

- One and only one non-XA resource is used. If any additional non-XA resource is used, the transaction is aborted, because the transaction manager must use XA protocol to commit two or more resources.
- No transaction importing or exporting occurs.

Transactions that involve multiple resources or multiple participant processes are *distributed* or *global* transactions. A global transaction can involve one non-XA resource if last agent optimization is enabled. Otherwise, all resourced must be XA. The `use-last-agent-optimization` property is set to `true` by default. For details about how to set this property, see "Configuring the Transaction Service" on page 24.

If only one XA resource is used in a transaction, one-phase commit occurs, otherwise the transaction is coordinated with a two-phase commit protocol.

A two-phase commit protocol between the transaction manager and all the resources enlisted for a transaction ensures that either all the resource managers commit the transaction or they all abort. When the application requests the commitment of a transaction, the transaction manager issues a `PREPARE_TO_COMMIT` request to all the resource managers involved. Each of these resources can in turn send a reply indicating whether it is ready for commit (`PREPARED`) or not (`NO`). Only when all the resource managers are ready for a commit does the transaction manager issue a commit request (`COMMIT`) to all the resource managers. Otherwise, the transaction manager issues a rollback request (`ABORT`) and the transaction is rolled back.

Configuring the Transaction Service

You can configure the transaction service in the GlassFish Server in the following ways:

- To configure the transaction service using the Administration Console, open the Transaction Service component under the relevant configuration. For details, click the Help button in the Administration Console.
- To configure the transaction service, use the `set`(1) subcommand to set the following attributes.

  The following examples show the `server-config` configuration, but values for any configuration can be set. For example, if you create a cluster named `cluster1` and a configuration named `cluster1-config` is automatically created for it, you can use `cluster1-config` in the `set` subcommand to get the transaction service settings for that cluster.

  ```
  server-config.transaction-service.automatic-recovery = false
  server-config.transaction-service.heuristic-decision = rollback
  server-config.transaction-service.keypoint-interval = 2048
  server-config.transaction-service.retry-timeout-in-seconds = 600
  server-config.transaction-service.timeout-in-seconds = 0
  server-config.transaction-service.tx-log-dir = domain-dir/logs
  ```

  You can also set these properties:

```
server-config.transaction-service.property.oracle-xa-recovery-workaround = true
server-config.transaction-service.property.sybase-xa-recovery-workaround = false
server-config.transaction-service.property.disable-distributed-transaction-logging = false
server-config.transaction-service.property.xaresource-txn-timeout = 0
server-config.transaction-service.property.pending-txn-cleanup-interval = -1
server-config.transaction-service.property.use-last-agent-optimization = true
server-config.transaction-service.property.delegated-recovery = false
server-config.transaction-service.property.wait-time-before-recovery-insec = 60
server-config.transaction-service.property.purge-cancelled-transactions-after = 0
server-config.transaction-service.property.commit-one-phase-during-recovery = false
server-config.transaction-service.property.add-wait-point-during-recovery = 0
server-config.transaction-service.property.db-logging-resource = jdbc/TxnDS
server-config.transaction-service.property.xa-servername = myserver
```

Default property values are shown where they exist. For `db-logging-resource` and `xa-servername`, typical values are shown. Values that are not self-explanatory are as follows:

- The `xaresource-txn-timeout` default of `0` means there is no timeout. The units are seconds.
- The `pending-txn-cleanup-interval` default of `-1` means the periodic recovery thread doesn't run. The units are seconds.
- The `purge-cancelled-transactions-after` default of `0` means cancelled transactions are not purged. The units are the number of cancellations in between purging attempts.
- The `add-wait-point-during-recovery` property does not have a default value. If this property is unset, recovery does not wait. The units are seconds.
- The `db-logging-resource` property does not have a default value. It is unset by default. However, if you set `db-logging-resource` to an empty value, the value used is `jdbc/TxnDS`.
- The `xa-servername` property does not have a default value. Use this property to override server names that can cause errors.

You can use the `get`(1) subcommand to list all the transaction service attributes and the properties that have been set. For details, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

Changing `keypoint-interval`, `retry-timeout-in-seconds`, or `timeout-in-seconds` does not require a server restart. Changing other attributes or properties requires a server restart.

- You can also set the following system properties:

```
ALLOW_MULTIPLE_ENLISTS_DELISTS=false
JTA_RESOURCE_TABLE_MAX_ENTRIES=8192
JTA_RESOURCE_TABLE_DEFAULT_LOAD_FACTOR=0.75f
```

The `JTA_RESOURCE_TABLE_DEFAULT_LOAD_FACTOR` default is the default `Map` resizing value.

Managing the Transaction Service for Rollbacks

You can roll back a single transaction by using the `asadmin` subcommands described in this section. To do so, the transaction service must be stopped (and later restarted), allowing you to see the active transactions and correctly identify the one that needs to be rolled back.

The following topics are addressed here:

▼ To Stop the Transaction Service

Use the `freeze-transaction-service` subcommand in remote mode to stop the transaction service. When the transaction service is stopped, all in-flight transactions are immediately suspended. You must stop the transaction service before rolling back any in-flight transactions.

Running this subcommand on a stopped transaction subsystem has no effect. The transaction service remains suspended until you restart it by using the `unfreeze-transaction-service` subcommand.

1. **Ensure that the server is running.**

   Remote subcommands require a running server.

2. **Stop the transaction service by using the `freeze-transaction-service`(1) subcommand.**

Example 18–1 Stopping the Transaction Service

This example stops the transaction service.

```
asadmin> freeze-transaction-service
Command freeze-transaction-service executed successfully
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help freeze-transaction-service` at the command line.

▼ To Roll Back a Transaction

In some situations, you might want to roll back a particular transaction. Before you can roll back a transaction, you must first stop the transaction service so that transaction operations are suspended. Use the `rollback-transaction` subcommand in remote mode to roll back a specific transaction.

1. **Ensure that the server is running.**

   Remote subcommands require a running server.

2. **Enable monitoring using the `set` subcommand. For example:**

```
asadmin> set clstr1-config.monitoring-service.module-monitoring-levels.transaction-service=HIGH
```

3. **Use the `freeze-transaction-service` subcommand to halt in-process transactions. See "To Stop the Transaction Service" on page 26.**

4. **Identify the ID of the transaction you want to roll back.**

   To see a list of IDs of active transactions, use the `get` subcommand with the `--monitor` option to get the monitoring data for the `activeids` statistic. See "Transaction Service Statistics" on page . For example:

   ```
   asadmin> get --monitor inst1.server.transaction-service.activeids-current
   ```

5. **Roll back the transaction by using the `rollback-transaction`(1) subcommand.**

   The transaction is not rolled back at the time of this command's execution, but only marked for rollback. The transaction is rolled back when it is completed.

Example 18–2 Rolling Back a Transaction

This example rolls back the transaction with transaction ID `0000000000000001_00`.

```
asadmin> rollback-transaction 0000000000000001_00
Command rollback-transaction executed successfully
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help rollback-transaction` at the command line.

▼ To Restart the Transaction Service

Use the `unfreeze-transaction-service` subcommand in remote mote to resume all the suspended in-flight transactions. Run this subcommand to restart the transaction service after it has been frozen.

1. **Ensure that the server is running.**

   Remote subcommands require a running server.

2. **Restart the suspended transaction service by using the `unfreeze-transaction-service`(1) subcommand.**

Example 18–3 Restarting the Transaction Service

This example restarts the transaction service after it has been frozen.

```
asadmin> unfreeze-transaction-service
Command unfreeze-transaction-service executed successfully
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help unfreeze-transaction-service` at the command line.

Determining Local Transaction Completion at Shutdown

When you shut down a GlassFish Server instance, all database connections are closed. When an Oracle JDBC driver-based database connection is closed in the middle of a non-XA transaction, all pending changes are committed. Other databases usually roll back pending changes when a connection is closed without being explicitly committed. To determine the exact behavior for your database, refer to the documentation from your JDBC driver vendor.

To explicitly specify whether GlassFish Server commits or rolls back non-XA transactions at server shutdown, set the `com.sun.enterprise.in-progress-local-transaction.completion-mode` JVM option to either `commit` or `rollback` using the `create-jvm-options`(1) subcommand. For example:

```
asadmin> create-jvm-options -Dcom.sun.enterprise.in-progress-local-transaction.completion-mode=rollback
```

Recovering Transactions

There are some situations where the commit or rollback operations might be interrupted, typically because the server crashed or a resource manager crashed. Crash situations can leave some transactions stranded between steps. GlassFish Server is designed to recover from these failures. If the failed transaction spans multiple servers, the server that started the transaction can contact the other servers to get the outcome of the transaction. If the other servers are unreachable, the transaction uses heuristic decision information to determine the outcome.

The following topics are addressed here:

Automatic Transaction Recovery

GlassFish Server can perform automatic recovery in these ways:

- Pending transactions are completed upon server startup if `automatic-recovery` is set to `true`.
- Periodic automatic recovery is performed by a background thread if the `pending-txn-cleanup-interval` property is set to a positive value.

Changing these settings requires a server restart. For more information about how to change these settings, see "Configuring the Transaction Service" on page 24.

If commit fails during recovery, a message is written to the server log.

▼ To Manually Recover Transactions

Use the `recover-transactions` subcommand in remote mode to manually recover transactions that were pending when a resource or a server instance failed.

For a standalone server, do not use manual transaction recovery to recover transactions after a server failure. For a standalone server, manual transaction recovery can recover transactions only when a resource fails, but the server is still running. If a standalone server fails, only the full startup recovery process can recover transactions that were pending when the server failed.

For an installation of multiple server instances, you can use manual transaction recovery from a surviving server instance to recover transactions after a server failure. For manual transaction recovery to work properly, transaction logs must be stored on a shared file system that is accessible to all server instances. See "Transaction Logging" on page 31.

When you execute `recover-transactions` in non-delegated mode, you can recover transactions that didn't complete two-phase commit because of a resource crash. To use manual transaction recovery in this way, the following conditions must be met:

- The `recover-transactions` command should be executed after the resource is restarted.
- Connection validation should be enabled so the connection pool is refreshed when the resource is accessed after the recovery. For more information, see "Configuring the Transaction Service" on page 24.

If commit fails during recovery, a message is written to the server log.

Note –

A JMS resource crash is handled the same way as any other resource.

You can list in-doubt Open Message Queue transactions using the `imqcmd list txn` subcommand. For more information, see "Managing Transactions" in *Open Message Queue 4.5 Administration Guide*.

---

1 **Ensure that the server is running.**

Remote subcommands require a running server.

2 **Manually recover transactions by using the `recover-transactions`(1) subcommand.**

Example 18–4 Manually Recovering Transactions

This example performs manual recovery of transactions on `sampleserver`.

```
asadmin recover-transactions sampleserver
Transaction recovered.
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help recover-transactions` at the command line.

Distributed Transaction Recovery

To enable cluster-wide automatic recovery, you must first facilitate storing of transaction logs in a shared file system. See "Transaction Logging" on page 31.

Next, you must set the transaction service's `delegated-recovery` property to `true` (the default is `false`). For information about setting `tx-log-dir` and `delegated-recovery`, see "Configuring the Transaction Service" on page 24.

Recovery Workarounds and Limitations

The GlassFish Server provides workarounds for some known issues with transaction recovery implementations.

Note – These workarounds do not imply support for any particular JDBC driver.

General Recovery Limitations

The following general limitations apply to transaction recovery:

- Recovery succeeds if there are no exceptions during the process. This is independent of the number of transactions that need to be recovered.
- Only transactions that did not complete the two-phase commit can be recovered (one of the XA resources failed or GlassFish Server crashed after resources were prepared).
- Manual transaction recovery cannot recover transactions after a server crash on a stand-alone server instance. Manual operations are intended for cases when a resource dies unexpectedly while the server is running. In case of a server crash, only startup recovery can recover in-doubt transactions.
- It is not possible to list transaction IDs for in-doubt transactions.
- Delegated transaction recovery (by a different server instance in a cluster) is not possible if the failed instance used an `EMBEDDED` Message Queue broker, or if it used a `LOCAL` or `REMOTE` Message Queue broker and the broker also failed. In this case, only automatic recovery on server instance restart is possible. This is because for conventional Message Queue clustering, state information in a failed broker is not available until the broker restarts.

Oracle Setup for Transaction Recovery

You must configure the following `grant` statements in your Oracle database to set up transaction recovery:

```
grant select on SYS.DBA_PENDING_TRANSACTIONS to user;
grant execute on SYS.DBMS_SYSTEM to user;
grant select on SYS.PENDING_TRANS$ to user;
grant select on SYS.DBA_2PC_NEIGHBORS to user;
grant execute on SYS.DBMS_XA to user;
grant select on SYS.DBA_2PC_PENDING to user;
```

The *user* is the database administrator. On some versions of the Oracle driver the last `grant execute` fails. You can ignore this.

Oracle Thin Driver

In the Oracle thin driver, the `XAResource.recover` method repeatedly returns the same set of in-doubt Xids regardless of the input flag. According to the XA specifications, the Transaction Manager initially calls this method with TMSTARTSCAN and then with TMNOFLAGS repeatedly until no Xids are returned. The `XAResource.commit` method also has some issues.

To disable the GlassFish Server workaround, set the `oracle-xa-recovery-workaround` property value to `false`. For details about how to set this property, see "Configuring the Transaction Service" on page 24. This workaround is used unless explicitly disabled.

Delegated Recovery After Server Crash Doesn't Work on MySQL

The MySQL database supports XA transaction recovery only when the database crashes. When a GlassFish Server instance crashes, MySQL rolls back prepared transactions.

Transaction Logging

The transaction service writes transactional activity into transaction logs so that transactions can be recovered. You can control transaction logging in these ways:

- Set the location of the transaction log files in one of these ways:
  - Set the GlassFish Server's `log-root` setting to a shared file system base directory and set the transaction service's `tx-log-dir` attribute to a relative path.
  - Set `tx-log-dir` to an absolute path to a shared file system directory, in which case `log-root` is ignored for transaction logs.
  - Set a system property called `TX-LOG-DIR` to a shared file system directory. For example:

    ```
    asadmin> create-system-properties --target server TX-LOG-DIR=/inst1/logs
    ```

  For information about setting `log-root` and other general logging settings, see Chapter 7, "Administering the Logging Service."

- Turn off transaction logging by setting the `disable-distributed-transaction-logging` property to `true` and the `automatic-recovery` attribute to `false`. Do this *only* if performance is more important than transaction recovery.

Note – All instances in a cluster must be owned by the same user (`uid`), and read/write permissions for that user must be set on the transaction log directories.

Transaction logs should be stored in a high-availability network file system (NFS) to avoid a single point of failure.

▼ To Store Transaction Logs in a Database

For multi-core machines, logging transactions to a database may be more efficient.

This feature is intended for resource recovery on a stand-alone server instance while the instance is healthy, not for a server crash. It is not intended for use on a cluster.

1 Create a JDBC connection Pool, and set the `non-transactional-connections` attribute to `true`.

2 Create a JDBC resource that uses the connection pool and note the JNDI name of the JDBC resource.

3 Create a table named `txn_log_table` with the following schema:

Column Name	JDBC Type
`LOCALTID`	`BIGINT`
`SERVERNAME`	`VARCHAR(n)`
`GTRID`	`VARBINARY`

The size of the `SERVERNAME` column should be at least the length of the GlassFish Server host name plus 10 characters.

The size of the `GTRID` column should be at least 64 bytes.

4 Add the `db-logging-resource` property to the transaction service. For example:

```
asadmin set server-config.transaction-service.property.db-logging-resource="jdbc/TxnDS"
```

The property's value should be the JNDI name of the JDBC resource configured previously.

5 To disable file synchronization, use the following `asadmin create-jvm-options` command:

```
asadmin create-jvm-options -Dcom.sun.appserv.transaction.nofdsync
```

6 **Restart the server.**

Next Steps To define the SQL used by the transaction manager when it is storing its transaction logs in the database, use the following flags:

`-Dcom.sun.jts.dblogging.insertquery=sql` *statement*

`-Dcom.sun.jts.dblogging.deletequery=sql` *statement*

The default statements are as follows:

```
-Dcom.sun.jts.dblogging.insertquery=insert into txn_log_table values ( ?, ?, ? )
```

```
-Dcom.sun.jts.dblogging.deletequery=delete from txn_log_table where localtid =
? and servername = ?
```

To set one of these flags using the `asadmin create-jvm-options` command, you must quote the statement. For example:

```
create-jvm-options '-Dcom.sun.jts.dblogging.deletequery=delete from
txn_log_table where gtrid = ?'
```

You can also set JVM options in the Administration Console. Select the JVM Settings component under the relevant configuration. These flags and their statements must also be quoted in the Administration Console. For example:

```
'-Dcom.sun.jts.dblogging.deletequery=delete from txn_log_table where gtrid = ?'
```

See Also For information about JDBC connection pools and resources, see Chapter 11, "Administering Database Connectivity." For more information about the `asadmin create-jvm-options` command, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.