DSO Technical FAQ

About Terracotta Documentation

This documentation is about Terracotta DSO, an advanced distributed-computing technology aimed at meeting special clustering requirements.

Terracotta products without the overhead and complexity of DSO meet the needs of almost all use cases and clustering requirements. To learn how to migrate from Terracotta DSO to standard Terracotta products, see Migrating From Terracotta DSO. To find documentation on non-DSO (standard) Terracotta products, see Terracotta Documentation. Terracotta release information, such as release notes and platform compatibility, is found in Product Information.

Release: 3.6
Publish Date: November, 2011 **Documentation Archive »**

Did this FAQ help you?

Send your questions and comments to info@terracotta.org.

Technical FAQ

Can I run the Terracotta process as a Microsoft Windows service?

While running an application as a Microsoft Windows service has many advantages, such as scheduling and automatic start and restart, there is no official supported configuration for doing this with Terracotta software. However, there are solutions available that have been tried successfully, including some with Java Service Wrapper. This blog entry includes a short procedure, and this blog entry shows how to do the same on a 64-bit system.

Try an Internet search on "windows java service" to find other possible solutions and articles.

How can my application check that the Terracotta process is alive at runtime?

Your application can check to see if the system property `tc.active` is true. For example, the following line of code would return true if Terracotta is active at the time it is run:

```
Boolean.getBoolean("tc.active");
```

Is there a maximum number of objects that can be held by one Terracotta server instance?

The number of objects that can be held by a Terracotta server instance is two billion, a limit imposed by the design of Java collections. It is unlikely that a Terracotta cluster will need to approach even 50 percent of that maximum. However, if it does, other issues may arise that require the rearchitecting of how application data is handled in the cluster.

I know Ubuntu isn't a supported platform, but do you have any advice for running Terracotta software on it?

The known issues when trying to run Terracotta software on Ubuntu are:

- Default shell is **dash bash**. Terracotta scripts don't behave under dash. You might solve this issue by setting your default shell to bash or changing `/bin/sh` in our scripts to `/bin/bash`.
- The Ubuntu default JDK is from GNU. Terracotta software supports only the Sun JDKs.
- See the UnknownHostException in the Troubleshooting Guide.

Clustered Objects

Q: What is a "root"?

A: A root is a connection between a field and a tree of objects. When a field in a class is marked as a root, it and all of its references become shared across all client applications that reference that root.

more »

Q: What is a "managed object" or "shared object"?

A: A "managed object" or "shared object" is any object that has yet to be garbage collected by the Terracotta garbage collector in the Terracotta Server.

more »

Q: Is the environment garbage collected?

A: Yes. We have a full distributed garbage collector.
more »

Q: What will happen if I remove an object from a shared root? Is it still shared?

A: Yes. Once an object becomes shared, it is always shared until it is garbage collected by the Terracotta distributed garbage collector.
more »

Q: When is an object eligible for garbage collection by the Terracotta distributed garbage collector?

A: When it is not reachable by reference from any root and there are no references to it in any client JVM (i.e., the object has been garbage collected from every client JVM).

more »

Q: What will happen if I remove an object from a shared root and then add it again?

A: Everything will work just fine.

Q: Can I make fields transient?

A: Yes. For more information on field transience in Terracotta, see the Concept and Architecture Guide

Q: Can an object be referenced by more than one root?

A: Yes.

Q: Can a shared object be referenced by non-shared objects?

A: Yes.

Q: Will changes to shared objects made by non-shared objects be visible to the cluster?

A: Yes, provided that the class of the non-shared object has been instrumented with our bytecode instrumentation and the changes were made under a clustered lock.

more about changes and locks »

Q: Are all my objects in every client?

A: Not necessarily. We dynamically swap partial trees of objects in and out of the client JVMs and from the Terracotta Server to the disk.

more »

Q: Do all changes get sent to every client?

A: No. Clients only receive changes for objects that they currently have in memory.

Clustered Locks

Q: What is a "lock"?

A: Just like in a regular multi-threaded Java application, a lock serves as a memory boundary and a critical (protected) region. In DSO, the batch of changes made in a lock are applied transactionally to the cluster. Note that Terracotta locks can do more than the exclusive locking that the JVM is capable of on its own.

more »

Q: Can I nest locks?

A: Yes, though you must be careful to avoid lock upgrades. Also, nesting concurrent locks is not currently supported.

more »

Q: How do locks work - what guarantees of data coherency do I have?

A: In general, locks behave the same way as the Java Memory Model. At the beginning of a lock boundary, you are guaranteed to read the fields of an object in a consistent state. For more details, see this Lock Sequence Diagram.

more »

Q: How can I start or stop transaction boundaries independent of code paths? (aka synchronized is not good enough, what can I do?)

A: You can use Java 1.5's ReentrantReadWriteLock. Example:

```
int foo = 0;
int bar = 0;
ReentrantReadWriteLock lock = new ReentrantReadWriteLock();

public void incrementFoo()
{
    foo++;
}

public void increment()
{
    bar++;
}

public void startSession()
{
    lock.writeLock().lock();
}

public void endSession()
{
    lock.writeLock().unlock();
}
```

Q: What semantics are associated with the Concurrent Lock?

A: A Concurrent Lock in Terracotta defines a transaction boundary but gives no visibility or correctness guarantees.
There is no conversation with the Terracotta server - but the "Terracotta transaction" begins when the concurrent lock is acquired and ends when the concurrent lock is released and all mutations within are batched and committed to the Terracotta server. So with a concurrent lock, 2 threads could read or write to a shared object graph concurrently (i.e no correctness guarantees)

Defining no lock implies GETS/READS might work but there is no guarantee of correctness (in that one might be able to get stale or partially committed data) - but then you would not be able to write to it. With Concurrent Lock in place, GET/READS/WRITES can all work with no guarantees around correctness.

Designing and Developing for Clustering

Q: How do I do listener style notifications for GUIs?

A: We have distributed method calls for just this reason. Take a look at the Shared Graphics Editor sample applicaton to get a feel for how to use them for this purpose.

Running and Tuning Your Application With Terracotta

Q: How do I start my application using Terracotta DSO?

A: In your installation 'bin' directory there is a 'dso-java' script. This script can be used exactly like your regular 'java' executable. It puts the necessary DSO start-up information in your command line. Or, you can add the necessary DSO start-up information in your command line.

Q: Can I tell what locks have been created?

A: While developing your application you can find out about all the locks that were created by setting /tc-config/clients/dso/debugging/instrumentation-logging/locks to 'true' in your tc-config.xml file. (The logging can be controlled at a finer level; see the reference config under config-sample/tc-config-reference.xml.) The output shows up in your terracotta-l1.log file.

Q: Can I tell what roots have been created?

A: While developing your application you can find out about all the roots that were created by setting /tc-config/clients/dso/debugging/instrumentation-logging/roots to 'true' in your tc-config.xml file. (The logging can be controlled at a finer level; see the reference config under config-sample/tc-config-reference.xml.). The output shows up in your terracotta-l1.log file.

Q: Can I see the entering and exiting of locks in my Terracotta Client?

A: Yes. While developing your application you can see lock acquires and releases by turning on runtime logging. This is done by setting the /tc-config /clients/dso/debugging/runtime-logging/lock-debug to 'true' in your tc-config.xml file. (The logging can be controlled at a finer level; see the reference config under config-sample/tc-config-reference.xml.) The output shows up in your terracotta-l1.log file.

Q: What transactionality guarantees does Terracotta provide?

A: In DSO, the batch of changes made to the clustered object graph within the boundaries of a lock (begin/end of a "synchronized" block of java code, ReentrantReadWriteLock lock(), OR begin/end of a cluster-wide "named-lock") are replicated (to the Terracotta-Server and other nodes in the cluster which have a reference to that object) either in an Asynchronous or Synchronous mode. Terracotta DSO passes the classic ACID (Atomic, Consistent, Isolated, Durable) test. DSO does not support semantics of an external transaction manager such as JTA or Spring JdoTransactionManager. There is no notion of a "rollback" in the Java-heap, if a transaction fails. However, the standard (non-DSO) installation of Terracotta with Enterprise Ehcache supports multiple JTA options.

Q: How can I reduce contention?

A: Lock on fine-grained objects as much as possible. Use our variety of lock types (write, concurrent, read) where appropriate to avoid contention.

Q: Why do all of my objects disappear when I restart the server?

A: If you are not running the server in persistent mode, the server will remove the object data when it restarts. If you want object data to persist across server restarts, run the server in persistent mode.

Q: How do I enable persistent mode?

In the servers section of your config.xml, add the following lines:

```
<server host="host1" name="server1">
...
  <dso>
  ...
    <persistence>
      <mode>permanent-store</mode>
    </persistence>
  ...
  </dso>
...
</server>
```

See the 'persistence' section of the Configuration Guide and Reference for more details.

Q: Why are old objects still there when I restart the server?

A: If you are running the server in persistent mode, the server keeps the object data across restarts. If you want objects to disappear when you restart the server you can either run in non-persistent mode or remove the data files from disk before you restart the server. See the 'persistence' and 'data' sections of the Configuration Guide and Reference

Terracotta and Other Technologies

Q: Can I mix Terracotta DSO with AOP frameworks?

A: Because of collisions that can occur due to both DSO and AOP instrumentation of classes, mixing the two is not an option. However, if instead of DSO the standard Terracotta installation is used, then no collisions can occur and the AOP framework can be used.

Q: Can I substitute Terracotta clustering for JMS? How do you do messaging in Terracotta?

A: Using Terracotta with a simple data structure (such as java.util.concurrent.LinkedBlockingQueue), you can easily create message queues that can replace JMS. Your particular use case should dictate whether to replace JMS or continue using it alongside Terracotta.

Q: Does Terracotta clustering work with Hibernate?

A: Terracotta software supports second-level cache clustering and also clustering of POJO data created by Hibernate. For more information, see .

Q: What other technologies does Terracotta software work with?

A: Terracotta software integrates with most popular Java technologies being used today. For a full list, contact us at info@terracotta.org.