Troubleshooting Guide

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 »**

Troubleshooting Guide

Firewall Connection Issues

If you are having connection or network related-problems that may be due to a firewall, see the topic for one of the following:

- Terracotta Clusters
- Terracotta Update Center (tim-get)
- Terracotta Developer Console or Terracotta Operations Center

Two Common Errors

Getting **TCNonPortableObjectError**? See Resolving TCNonPortableObjectError.
Getting **UnlockedSharedObjectException**? See Resolving UnlockedSharedObjectException.

Potentially Outdated Information

Some of the information provided in this document may be out of date. While this document waits to be updated, start with the Terracotta documentation page for more information.

The issues covered in this guide are divided into the following sections:

An expanded contents list showing every covered issue is included below.

If your question isn't answered by this guide, try one of the following resources:

- DSO Technical FAQ
- Terracotta Forums
- Users mailing list
- Reference documentation listed in the navigation list on the left of this page.

Contents

EXCEPTIONS AND ERRORS...

ClassCastException or ClassNotFoundException

These errors can be caused when differing clustered applications share object graphs but use different classloaders.

These errors can also be caused when a running web application is redeployed, such as when its container detects a change to a context file. Terracotta does not support "hot" redeployment of web applications.

See this topic for more information.

UnsupportedOperationException

You are attempting to use the deprecated JMX interface for cluster events. See Cluster Events for more information.

On Windows, I get the following error message right after the first "preparing to install" dialog box finishes: Windows error 3 occurred while loading the Java VM.

You probably have several versions of the Java SDK and runtime installed and something in the Microsoft Windows registry entries on your machine is confusing the installer. (Even if the JAVA_HOME environment variable is set correctly). To get past this issue, clean up your environment or add the "LAX_VM" parameter and the location of the correct Java executable to the command line, like this:

terracotta-windows-2.6.0.exe LAX_VM "c:/jdk1511/bin/java.exe"

The DSO boot JAR you need for this platform does not exist. You may need to run the 'make-boot-jar' script.

You need to create the boot JAR. Terracotta includes a set of standard boot JARS, but none may match your environment's JDK version and/or OS. The make-boot-jar tool fixes this error by creating the appropriate boot JAR for your environment. You should run make-boot-jar each time relevant changes, such as a JVM upgrade, occur in the Terracotta installation environment.

$TC_INSTALL_DIR%/common/lib/dso-boot/ is the default output location for make-boot-jar. To specify a different output location for the boot JAR, use the -o option (output).

Windows
`%TC_INSTALL_DIR%\dso\bin\make-boot-jar.bat [-o My/Output/Location]`

Linux

```
$TC_INSTALL_DIR/platform/bin/make-boot-jar.sh [-o My/Output/Location]
```


Make sure to delete the existing boot JAR files under TC_INSTALL_DIR/lib/dso-boot. The make-boot-jar tool may be unable to replace existing files.

See the Concept and Architecture Guide for more information about the boot JAR file.

Related Issues:

See the Terracotta Boot JAR section.

com.tc.exception.TCNonPortableObjectError

See Resolving TCNonPortableObjectError for a practical discussion on how to prevent this exception.

The following example includes a general discussion of situations that can generate a non-portable exception.

Example

Consider these example classes:

```
class Person {
   String firstName;
   String lastName;
... }

class Customer extends Person {
   long customerID;
... }

class MyProperties extends java.util.Properties {
... }

class MyThread extends Thread {
... }

class Address {
   private Logger logger;
   private String stree1;
   private String street2;
   private State state;
... }

class Logger {
   private FileOutputStream out;
... }
```

The following situations can cause TCNonPortableException.

Class Not Included

If you want to share an instance of the Customer class, but the Customer class has not been included for instrumentation in the Terracotta configuration, simply add or modify an `<include>` declaration in the Terracotta configuration file that matches the Customer classname.

Superclass Not Included

If you want to share an instance of the Customer class, but the Person class hasn't been included for instrumentation in the Terracotta configuration, simply add or modify an `<include>` declaration in the Terracotta configuration file that matches the Person classname.

Subclass of a Logically-Managed Class

If you want to share an instance of the MyProperties class you will get a portability exception because it extends java.util.Properties which, in turn, extends java.util.Hashtable which is a logically-managed class.

The general solution is to refactor your code so there are no logically-managed classes in the type hierarchy of the objects you want to share. In this case, you can change from an inheritance model to an aggregation model by modifying the MyProperties class to contain a reference to a Properties object rather than extending the Properties class.

It is only subclasses of logically-managed classes that cannot be shared. For example, direct instances of `java.util.HashMap` can always be shared without issue.

Subclass of a Non-Shareable Class

If you want to share an instance of the MyThread class, you will get a portability exception because it extends Thread which is not shareable. Thread is not shareable because it represents a JVM-specific resource.

The general solution is to refactor your code so there are no non-shareable classes in the type hierarchy of the objects you want to share. In this case, you can make `MyThread` an implement `Runnable` and pass it to a `Thread` object.

Reference to Non-Shareable Class

In order to share an instance of the `Address` class, when the object graph of the `Address` object is traversed and checked for portability, the traverser will follow the `Address.logger` reference and try to share all of the fields of the `Logger` object. When it reaches the `Logger.out` reference, it will throw a portability error because the `Logger.out` field is a reference to `FileOutputStream` which is a host machine-specific resource and cannot be shared.

In this case, it's not practical to remove the reference to the logger object. Terracotta recommends making a portion of the object graph transient so the unshareable `FileOutputStream` object is ignored by Terracotta.

A Terracotta server instance restarted, but the clients did not automatically reconnect.

If you've restarted Terracotta server instances and clients do not automatically reconnect without a restart, the Terracotta server instances are not configured for permanent store.

com.tc.object.tx.UnlockedSharedObjectException

See Resolving UnlockedSharedObjectException. If the exception refers to third-party code, see this issue.

I get com.tc.object.tx.UnlockedSharedObjectException on a third party JAR and do not have access to code.

Use Sparingly

While the following solution suggests the use of "named" locks, this type of lock is a last resort. There can be a hefty performance penalty to using named locks, so use named locks sparingly and only where absolutely necessary.

You can use "named" locks instead of autolocks to accommodate code that should be synchronized but cannot be. Named locks will create one global lock across the cluster for the named lock. This can cause contention, and can, in some cases, decrease throughput. When using named locks, you do not need to use the synchronize keyword to isolate the transaction because the method becomes the point of synchronization, and the name of the method becomes the name of the lock. Due to there only being one named lock per method, *ALL* object instances will serialize access through the single named lock.

Here is an example:

tc-config.xml - named locks

```
<locks>
  <named-lock>
    <lock-name>mutateFoo</lock-name>
    <method-expression>void com.system.services.MutateService.mutate(Object)</method-expression>
    <lock-level>write</lock-level>
  </named-lock>
</locks>
```

See the Concept and Architecture Guide for more information about locks.
See the Configuration Guide and Reference for more information on how to configure locks.

java.lang.IllegalMonitorStateException

An IllegalMonitorStateException implies a lock is missing or cannot be applied to a shared object. If the code in question has synchronization, check for the following causes:

1. A lock-then-share operation occurred. A lock has been specified but was applied to an object before that object was shared. See this Gotcha for more information.
2. Read-level or named locks are being used where write-level locks or autolocks are necessary. Refactor your Terracotta configuration to use the appropriate locks.

java.io.IOException: No locks available

This is a problem with NFS lock mgr.

- Solution: Modify `tc-config.xml` to use different log files

I'm getting a Hyperic (Sigar) exception, and the Terracotta Developer Console or Terracotta Operations Center is not showing certain metrics?

These two problems are related to starting Java from a location different than the value of JAVA_HOME. To avoid the Hyperic error and restore metrics to the Terracotta consoles, invoke Java from the location specified by JAVA_HOME.

Unable to locate bundle dependency

You may get an error similar to the following, which seems to indicate a missing Terracotta module:

> **Missing Terracotta Module**
>
> 2008-01-18 09:47:55,240 FATAL - Unable to locate bundle dependency: 'jdk15_preinst_config', version '2.5.0', group-id 'org.terracotta.modules'; Tried to resolve the location using the following repositories: 'file:/www/terracotta/terracotta-2.5.0/bin/../modules/'

See the following sections for suggested solutions.

On Solaris platform:

This is a result of the way the default tar utility on Solaris works. Terracotta loads a few modules located in $TC_INSTALL_DIR/modules directory on startup and if you look in that directory, you should see a collection of .jar files. The default tar utility causes some of these file names to be truncated which results in the above scenario as Terracotta is unable to load these modules.

To get past this, use an alternative tar utility like gtar or download the patch from Sun which fixes this issue.

> **Links**
>
> 1. The patch for Solaris tar is Solaris 2.6 patch 105792-03. The SunSoft patch page is at: http://sunsolve.sun.com/pub-cgi/retrieve.pl?doc=fpatches/105792
> 2. You can download the Solaris version of GNUtar from: http://www.ibiblio.org/pub/packages/solaris/sparc/

More generic solution:

If it is not an unpacking issue and the kit is installed properly, check the following:

1. The user has read access to all files in the Terracotta installation (specifically the modules directory).
2. Make sure that the module specified in the tc-config.xml is actually located and read-accessible in the appropriate repository (default is $TC_INSTALL_DIR/modules).
3. Make sure that the module version number is specified appropriately in the tc-config.xml and there are no typos.

Why is the Terracotta Developer Console or Terracotta Operations Center timing out when it tries to connect to a Terracotta server?

If you've verified that your Terracotta cluster is up and running, but your attempt to monitor it remotely using a Terracotta console is unsuccessful, a firewall may be the cause. Firewalls that block traffic from Terracotta servers' JMX ports prevent monitoring tools from seeing those servers. To avoid this and other connection issues that may also be attributable to firewalls, ensure that the JMX and DSO ports configured in Terracotta are unblocked on your network.

Why can't certain nodes on my Terracotta cluster see each other on the network?

A firewall may be preventing different nodes on a cluster from seeing each other. If Terracotta clients attempt to connect to a Terracotta server, for example, but the server seems to not not have any knowledge of these attempts, the clients may be blocked by a firewall. Another example is a backup Terracotta server that comes up as the active server because it is separated from the active server by a firewall.

What if I can't use the Terracotta Update Center (tim-get) because of a firewall?

The Terracotta Update Center is the facility by which Terracotta Integration Modules are installed and updated using the `tim-get.sh` or `tim-get.bat` script. Firewalls can block the script. See Troubleshooting Connection Problems for more information.

When I start a Terracotta server, why does it fail with a schema error?

You may get an error similar to the following when a Terracotta server fails to start:

```
 Error Message:
Starting BootJarTool...
2008-10-08 10:29:29,278 INFO - Terracotta 2.7.0, as of 20081001-101049 (Revision 10251 by cruise@rh4mo0 from
2.7)
2008-10-08 10:29:30,459 FATAL -
********************************************************************************
The configuration data in the file at '/opt/terracotta/conf/tc-config.xml' does not obey the Terracotta schema:
[0]: Line 8, column 3: Element not allowed: server in element servers

********************************************************************************
```

This error occurs when there's a schema violation in the Terracotta configuration file, at the line indicated by the error text. To confirm that your configuration file follows the required schema, see the schema file included with the Terracotta kit. The kit includes schema files (*.xsd) for Terracotta, Ehcache, and Quartz configurations.

I have a setup with one active Terracotta server instance and a number of standbys, so why am I getting errors because more than one active server comes up?

Due to network latency or load, the Terracotta server instances may not may be have adequate time to hold an election. Increase the <election-time> property in the Terracotta configuration file to the lowest value that solves this issue.

If you are running on Ubuntu, see the note at the end of the UnknownHostException topic.

Why is a crashed Terracotta server instance failing to come up when I restart it?

If it's running in persistent mode, clear the crashed server instance's data directory.

Why is java.net.UnknownHostException thrown when I try to run Terracotta sample applications?

If an UnknownHostException occurs, and you experience trouble running the Terracotta Welcome application and the included sample applications on Linux (especially Ubuntu), you may need to edit the etc/hosts file.

The UnknownHostException may be followed by "unknown-ip-address".

For example, your etc/hosts file may contain settings similar to the following:

```
127.0.0.1       localhost
127.0.1.1    myUbuntu.usa myUbuntu
```

If myUbuntu is the host, you must change 127.0.1.1 to the host's true IP address.

You may be able to successfully start Terracotta server instances even with the "invalid" etc/hosts file, and receive no exceptions or errors, but other connectivity problems can occur. For example, when starting two Terracotta servers that should form a mirror group (one active and one standby), you may see behavior that indicates that the servers cannot communicate with each other.

Why does the JVM on my Sun SPARC machines crash regularly?

You may be encountering a known issue with the Sun Hotspot JVM for SPARC. The problem is expected to occur with Hotspot 1.6.0_08 and higher, but may have been fixed in a later version. For more information, see this Sun bug report and this possible fix.

SHARING IN-MEMORY OBJECTS...

It doesn't look like my application data is being shared. What do I do?

The object browsers available in the Terracotta Developer Console allow you to view the shared object graphs in your cluster and in each client. The roots of each shared object graph in your cluster are shown as the roots of each graph displayed in the browser. Client object browsers also indicate which object graphs each client is sharing, and which are not being shared.

If clients appear in the Terracotta Developer Console but certain roots are missing from the object browsers, ensure that the fully qualified name of each root is accurate. For example, if the fully qualified name of a class you want to cluster is `org.system.services.foo`, check that the Terracotta configuration file declares foo as a root using that full name. Because of dynamic class loading in Java, it's impossible for Terracotta to detect invalid roots due to issues such as missing package names, although scanning your project in Eclipse may locate missing roots.

If both clients and roots appear in the console, but problems with object sharing appear, check for the following causes:

- Static fields – A static field of a shared class is not shared unless that field is configured to be a Terracotta root. Optionally, make the field non-static if that is acceptable in your application.
- Differing configuration files – Clients using different Terracotta configuration files may not share objects properly. This may be the case, for example, if the configuration files do not have the same sets of root declarations and instrumented classes. Align the content in the configuration files and restart each application instance.
- Changed application code – If your application code has changed, the Terracotta configuration file may be out of date. Update the configuration file and restart each application instance.

  > ⚠ If your application code has changed, restarting your Terracotta server instances is also recommended. While this may not be necessary in all cases, it is safer to restart your entire cluster following code changes.

- Changed configuration file – If a Terracotta configuration file has changed, every client or server instances using that configuration file should be restarted to avoid clustering anomalies.
- Uninstrumented code – If uninstrumented code is writing or reading to shared objects, data may not be updated on all nodes. This can cause stale data to appear in some application servers. See the Uninstrumented Access Gotcha for more information.

How do I share Terracotta roots between a POJO application and a web application?

You can get applications to successfully share roots by placing them in application groups. See the Terracotta Configuration Guide and Reference for more information on configuring Terracotta app-groups.

> ⚠ **Switch to app-groups**
>
> With previous versions of Terracotta, you might have prevented classloader-related exceptions by setting the `com.tc.loader.system.name` property (`-Dcom.tc.loader.system.name`) to name classloaders. However, this property has been deprecated and support for it may be dropped.

How do I confirm that my application data is being shared?

The object browsers available in the Terracotta Developer Console allow you to view the shared object graphs in your cluster and in each client.

How do I know if my POJOs are being shared or not?

Use the cluster object browser and the client object browser, available in the Terracotta Developer Console. Cluster and client object browsers display the Terracotta root objects at the root of shared-object graphs.

How do I know if EHCache data is being shared? I can see a bunch of EHCache objects in the Terracotta Developer Console but cannot find my application data.

First thing to look for is the Terracotta root objects. Connect the Terracotta Developer Console ($TC_INSTALL_DIR/bin/dev-console.sh or dev-console.bat in Microsoft Windows) to the Terracotta server instance and look under the Roots. If you see the following roots, EHCache (Hibernate 2nd level cache) is being clustered properly:

```
net.sf.ehcache.CacheManager.ALL_CACHE_MANAGERS (java.util.Collections$SynchronizedRandomAccessList)
net.sf.ehcache.store.MemoryStoreEvictionPolicy.LRU (net.sf.ehcache.store.MemoryStoreEvictionPolicy)
net.sf.ehcache.store.MemoryStoreEvictionPolicy.LFU (net.sf.ehcache.store.MemoryStoreEvictionPolicy)
net.sf.ehcache.store.MemoryStoreEvictionPolicy.FIFO (net.sf.ehcache.store.MemoryStoreEvictionPolicy)
net.sf.ehcache.store.MemoryStoreEvictionPolicy.DSO (net.sf.ehcache.store.MemoryStoreEvictionPolicy)
net.sf.ehcache.CacheManager.singleton (net.sf.ehcache.CacheManager)
```

EHCache breaks down the key-value pair and stores the value object wrapped by Element objects in a fairly deep object graph, as it does a lot of bookkeeping with timestamps etc. I am breaking down below exactly where you should be looking for the actual application data stored in the cache:

```
net.sf.ehcache.CacheManager.ALL_CACHE_MANAGERS (java.util.Collections$SynchronizedRandomAccessList)
|
|_____ java.util.Collections$SynchronizedCollection.c (java.util.ArrayList) [1/1]
          |
          |_____ 0 (net.sf.ehcache.CacheManager)
                |
                |_____ net.sf.ehcache.CacheManager.caches (java.util.HashMap) [3/3]
                      |
                      |_____ 0 (MapEntry)
                            |
                            |_____ value (net.sf.ehcache.Cache)
                                  |
                                  |_____ net.sf.ehcache.Cache.memoryStore (net.sf.ehcache.store.
TimeExpiryMemoryStore)
(continued)
|
|_____ net.sf.ehcache.store.MemoryStore.map (net.sf.ehcache.store.TimeExpiryMemoryStore$SpoolingTimeExpiryMap)
       |
       |_____ com.tcclient.ehcache.TimeExpiryMap.timeExpiryDataStore (com.tcclient.cache.CacheDataStore)
              |
              |_____ com.tcclient.cache.CacheDataStore.store (java.util.Map[])
                     |
                     |_____ 0 (java.util.HashMap) [1/1]
                            |
                            |_____ 0 (MapEntry)
                                  |
                                  |_____ key=XML_0K_1551
                                         value (com.tcclient.cache.CacheData)
                                         |
                                         |_____ com.tcclient.cache.CacheData.value (net.sf.ehcache.Element)
                                                |
                                                |_____ net.sf.ehcache.Element.value (Value object type)
                                                       |
                                                       |_____ The value object put in the cache by the app
```

Why can't the shutdown thread in my application modify shared objects?

In a Terracotta cluster, an application server (or client node) isn't allowed to modify shared data as it is exiting the cluster. However, a second client node can modify the shared data on detecting the exit of the first node. See the Terracotta Cluster Events documentation for more information on implementing cluster events for the purpose of detecting topographical changes such as node exits.

THE TERRACOTTA BOOT JAR...

Why is there a boot JAR file?

The boot JAR instruments classes that are loaded early in the JVM's startup. These classes can't be instrumented in the usual way because they are instantiated before normal instrumentation can take place. Terracotta provides a set of standard boot JARs for a variety of environments.

See the Concept and Architecture Guide for more information about the boot JAR file.

Related Issues:

If you get the error **The DSO boot JAR you need for this platform does not exist. You may need to run the 'make-boot-jar' script.**, then non of the standard Terracotta boot JARs may be right for your environment. See this issue on creating a boot JAR that works in your environment.

Do I ever have to rebuild the boot JAR?

The boot JAR may have to be rebuilt under the following conditions:

- Before the first time you run Terracotta (see this issue for more information.)
- You've updated the <additional-boot-jar-classes> property in `tc-config.xml`
- You've made changes to Java locks that affect Java core classes such as java.util.Hashtable.

There is no need to rebuild the boot JAR if the changes you've made affect only application classes.

So I need to add a certain class to the boot JAR. How do I do it?

First, add the class in the configuration file in the <additional-boot-jar-classes> section:

tc-config.xml - boot JAR

```
<additional-boot-jar-classes>
    <include>java.awt.datatransfer.Clipboard</include>
</additional-boot-jar-classes>
```

Next, provide the configuration file with the above entry as a parameter to the make-boot-jar script:

Windows

```
%TC_INSTALL_DIR%\dso\bin\make-boot-jar.bat -f tc-config.xml
```

Linux

```
$TC_INSTALL_DIR/platform/bin/make-boot-jar.sh -f tc-config.xml
```


Make sure to delete the existing boot JAR files under TC_INSTALL_DIR/lib/dso-boot as the make-boot-jar script is sometimes unable to replace existing files.

TERRACOTTA FOR SPRING...

Improvements to Spring Integration Coming

As Terracotta moves toward framework-based integration, support for products such as Spring is being moved out of the core product. Instead of requiring an understanding of how to interact with the Terracotta platform, integration tasks will all follow an overall common procedure. This has the advantage of simplifying and normalizing a developer's work when integrating an application with Terracotta and closely supported technologies such as Spring.

To see how Spring is integrated with Terracotta 3.1 and above, and to learn about migrating from Spring integrations with pre-3.1 versions of Terracotta, see

Error rendering macro 'html'

Notify your Confluence administrator that "Bob Swift Atlassian Add-ons - HTML" requires a valid license. Reason: EXPIRED

.

For an example of how to use Spring effectively in an application clustered with Terracotta, see the Examinator Reference Application.

How is logical identity for the Spring application context maintained?

Terracotta for Spring creates DSO root for each Spring application context. Because Spring framework does not provide unique Id for the application context, matching patterns are used to select web application name and to specify matching config names to identify application contexts that need to be clustered.

Name of the DSO root is calculated based on the web application name and names of all config resources used to initialize application context. If any name of the Spring configs changes or if new configs been added, the generated id will be different.
See the following method for more details com.tcspring.DistributableBeanFactoryMixin.addLocation(String location)

When different configs are used but contexts need to have the same identity, the root name can be assigned manually using the following configuration element:

```
<application>
   <spring>
     <jee-application>
       <application-contexts>
         <application-context>
           <root-name>myClusteredSpringContext</root-name>
           ...
```

What if the Spring application context is not distributed?

Terracotta console or a web server logs shows that Spring application context is not distributed:

```
INFO [com.tcspring.ApplicationHelper] - Application name Tomcat.shared
INFO [com.tcspring.DistributableBeanFactoryMixin] - 09080A0C0905030C0E05070F0805050A Context is NOT distributed
```

To debug that you can enable DEBUG logging for org.springframework and or.tcspring categories using either log4j settings for your web application, or if running on Tomcat global logging settings at <tomcat home>\conf\logging.properties

Then you can check the actual names for the web application and config resources. For example:

```
 2007-12-27 18:33:43,507 INFO [com.tcspring.ApplicationHelper] - Application name Tomcat.context:/foo
 2007-12-27 18:33:43,804 INFO [org.springframework.beans.factory.xml.XmlBeanDefinitionReader] -
Loading XML bean definitions from ServletContext resource [/WEB-INF/spring/servlet.xml]
 2007-12-27 18:33:45,710 INFO [org.springframework.beans.factory.xml.XmlBeanDefinitionReader] -
Loading XML bean definitions from ServletContext resource [/WEB-INF/spring/dao.xml]
 2007-12-27 18:33:45,804 INFO [org.springframework.beans.factory.xml.XmlBeanDefinitionReader] -
Loading XML bean definitions from ServletContext resource [/WEB-INF/spring/services.xml]
 2007-12-27 18:33:46,273 INFO [org.springframework.beans.factory.xml.XmlBeanDefinitionReader] -
Loading XML bean definitions from ServletContext resource [/WEB-INF/spring/datasource.xml]
 2007-12-27 18:33:46,335 INFO [org.springframework.beans.factory.xml.XmlBeanDefinitionReader] -
Loading XML bean definitions from ServletContext resource [/WEB-INF/spring/cache.xml]
 2007-12-27 18:33:47,257 INFO [org.springframework.beans.factory.xml.XmlBeanDefinitionReader] -
Loading XML bean definitions from ServletContext resource [/WEB-INF/spring/captcha.xml]
 2007-12-27 18:33:47,320 INFO [org.springframework.beans.factory.xml.XmlBeanDefinitionReader] -
Loading XML bean definitions from ServletContext resource [/WEB-INF/spring/validators.xml]
 2007-12-27 18:33:47,351 INFO [org.springframework.beans.factory.xml.XmlBeanDefinitionReader] -
Loading XML bean definitions from ServletContext resource [/WEB-INF/spring/storedprocs.xml]
 2007-12-27 18:33:47,648 INFO [org.springframework.beans.factory.xml.XmlBeanDefinitionReader] -
Loading XML bean definitions from ServletContext resource [/WEB-INF/spring/pagers.xml]
 2007-12-27 18:33:47,695 INFO [com.tcspring.DistributableBeanFactoryMixin] - 09080A0C0905030C0E05070F0805050A
Matching
locations:[/WEB-INF/spring/servlet.xml, /WEB-INF/spring/dao.xml, /WEB-INF/spring/services.xml,
/WEB-INF/spring/datasource.xml, /WEB-INF/spring/cache.xml, /WEB-INF/spring/captcha.xml, WEB-INF/spring
/validators.xml,
/WEB-INF/spring/storedprocs.xml, /WEB-INF/spring/pagers.xml]
 2007-12-27 18:33:47,695 INFO [com.tcspring.DistributableBeanFactoryMixin]
- 09080A0C0905030C0E05070F0805050A Context is distributed
 2007-12-27 18:33:47,695 INFO [com.tcspring.ClassHierarchyWalker] - 09080A0C0905030C0E05070F0805050A
registering include for com.planetpope.beans.SystemStatus
 2007-12-27 18:33:47,695 INFO [com.tcspring.DistributableBeanFactoryMixin] - 09080A0C0905030C0E05070F0805050A
registering transient fields for systemStatus com.planetpope.beans.SystemStatus
```

In the above example application name is "foo" and the corresponding configuration would look like this:

```
  <application>
    <spring>
      <jee-application name="foo">
        <application-contexts>
          <application-context>
            <paths>


              <path>*/spring/servlet.xml
              <path>*/spring/dao.xml</path>
              <path>*/spring/services.xml</path>
              <path>*/spring/datasource.xml</path>
              <path>*/spring/cache.xml</path>
              <path>*/spring/captcha.xml</path>
              <path>*/spring/validators.xml</path>
              <path>*/spring/storedprocs.xml</path>
              <path>*/spring/pagers.xml</path>
            </paths>
            ...
```

When context is distributed, the Terracotta Developer Console shows a special spring_info root for each clustered context where you can see the web application name, matching config names, and clustered Spring bean names.
For example:

```
tc:spring_info:05000C00060B090F090D040B0D0D0501 (java.util.ArrayList) [4/4]
  0 (com.tcspring.ApplicationContextEventProtocol)
      com.tcspring.ApplicationContextEventProtocol.appName (java.lang.String)=jmx
      com.tcspring.ApplicationContextEventProtocol.beanNames (java.util.ArrayList) [2/2]
        0 (java.lang.String)=clusteredHistory
        0 (java.lang.String)=clusteredCounder
      com.tcspring.ApplicationContextEventProtocol.configs (java.util.ArrayList) [1/1]
        0 (java.lang.String)=demo/jmx/applicationContext.xml
      com.tcspring.ApplicationContextEventProtocol.eventTypes (java.util.ArrayList) [0/0]
```

How can I share data between a standalone application and web application?

See this topic.

BEST PRACTICES...

Terracotta clients are not showing up after I start my application servers.

If you use a startup script (such as startup.sh or catalina.sh with Apache Tomcat), make sure it does not overwrite the JAVA_OPTS environment variable, which must have -Xbootclasspath for Terracotta to start properly. If your startup script must set JAVA_OPTS, be sure it includes -Xbootclasspath.

If you've restarted Terracotta server instances and clients do not automatically reconnect without a restart, the Terracotta server instances are not configured for permanent store.

Is there a faster way to declare configurations?

The Eclipse plugin shipped with Terracotta makes the task of creating the Terracotta configuration seamless. You can also leverage the AspectWerkz pattern language for a regex approach to this task. For example, define read locks around all getters in a Class:

tc-config.xml - patterns

```
<locks>
  <autolock>
    <method-expression>* com.system.services.MutateService.get*(..)</method-expression>
    <lock-level>read</lock-level>
  </autolock>
</locks>
```

My code uses the singleton pattern. I get exceptions on restart.

If your code uses the singleton pattern, make sure that the root is the field in the Singleton and not a reference to the Singleton elsewhere in your code. Static fields are not replicated by Terracotta unless you explicitly mark them as roots (given that static fields belong to a "class" and not an "object" instance).

I lost some data after my entire cluster lost power and went down. How can I ensure that all data persists through a failure?

First, if you've lost only *some* data, then you have configured Terracotta to persist data (which is highly recommended in most production environments). The cause for losing a small amount of data could be disk "write" caching on the machines running the Terracotta server instances. If every Terracotta server instance lost power when the cluster went down, data remaining in the disk cache of each machine was obliterated.

Turning off disk caching is not an optimal solution because the machines running Terracotta server instances will suffer a substantial performance degradation. A better solution is to ensure that power is never interrupted at any one time to every Terracotta server instance in the cluster. This can be achieved through techniques such as the use of uninterruptible power supplies and geographically subdividing cluster members.

What's the best way for my application to listen to Terracotta cluster events such as lost application nodes?

Beginning with Terracotta 3.0.0, you can set up listening for cluster events by using the Terracotta API. See Cluster Events for more information.

Terracotta server instance restarts create double notifications.

It is possible to receive two notifications on server restart because of the following chain of events:

1. Thread1 is selected to be the notified thread in the server and the notification succeeds.
2. The server crashes.
3. When the server comes back up and the outstanding transactions are re-sent, the server chooses a different thread to be the notified thread and that notification also succeeds.

The solution involves using the server ID to distinguish between different and duplicate notifications:

- On the receiving client side, keep track of the server transaction ID that caused the notification.
- On client reconnect, the client lock state should contain the notifying server's transaction ID.

The server lock manager can then decide whether or not to choose a notified thread for a given transaction based on whether that transaction's notified thread has already been applied

HOW DO I...

How do I use Terracotta to cluster my POJOs? Is there an API?

There are three different ways to add clustering to your application using Terracotta:

- Add shared classes and to the Terracotta configuration file, `tc-config.xml`, and add locking to methods that read from or write to shared objects.
- Instrument classes directly in your code using annotations.
- Install a Terracotta Integration Module that includes the instrumented classes your application needs for clustering.

If your code is already optimized for data coherency by implementing synchronization designs provided by Java, Terracotta may already support those designs and little configuration may be necessary. For more information, see the Configuration Guide and Reference.

How do I confirm that my Terracotta servers and clients are up and running correctly?

Here are some ways to confirm that your Terracotta servers are running:

- Connect to the servers using the Terracotta Developer Console.
- When you start a Terracotta server, look at the standard output messages to see that each server started and did not exit.
- Check each server's logs to see that each server started and did not exit.
- Use a tool such as wget to access the /config or /version servlet. For example, for a server running on localhost and using DSO port 9510, use the following wget command to connect to the version servlet:

```
[PROMPT] wget http://localhost:9510/version
```

Here are some ways to confirm that your application servers are running with Terracotta:

- After connecting to the cluster using the Terracotta Developer Console, check that the clients are displayed as expected.
- When you start your application, look at the standard output messages to see that each Terracotta client started and did not exit.
- Check each client's logs to see that each server started and did not exit.

If an exception appears in any of the logs, search for information on it in this document or elsewhere in the Terracotta documentation.

How do I get failover to work properly with two Terracotta servers?

Configure both servers in the <servers> section of the Terracotta configuration file. Start the two Terracotta server instances that use that configuration file, one server assumes control of the cluster (the ACTIVE) and the second becomes the backup (the PASSIVE). See the high-availability chapter in the product documentation for more information.

How do I reduce planned downtime with Terracotta?

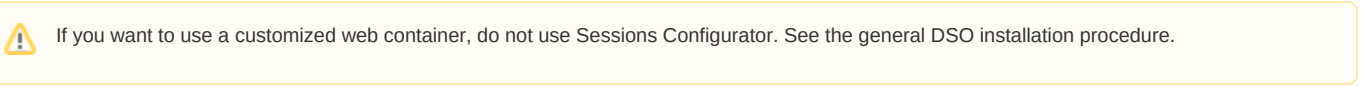
Terracotta persists objects in a new format (field name/value pairings), that does not use Java serialization. It is decoupled from specifics of class versioning. For example - adding/deleting a field to a class will require no flush of Terracotta caches. Additional use-cases in the table below.

Type of Upgrade	Currently Supported	Notes
Server JVM Upgrade	No	Servers always run with the supplied JVM version. (Shut primary L2 and upgrade it, while live traffic goes to secondary L2 and vice-versa)
Client JVM Upgrade	Yes	No issues here, the server is agnostic to the client JVM. The client JVM must be a supported platform. Some testing required if the implementation of the core-class changes between 1 JVM and another - quite rare.
Terracotta Version Upgrade	No	Future Terracotta versions may include rolling upgrade support.

Class Versioning - add /delete/modify methods	Yes	Terracotta is only concerned with changes to the attributes stored in a class.
Class Versioning - add a field	Yes	If you need to initialize the field you can use the onLoad feature (see the Configuration Guide and Reference).
Class Versioning - delete a field	Yes	If your class no longer references a field this does not cause any problems. Note that the data is preserved. For example, if Node1 has a class with field f1 and Node2 has a class without field f1, Node2 is not aware of f1.
Class Versioning - modify a field name	No	This is similar to adding a new field - the onLoad mechanism (see the Configuration Guide and Reference) will be used, however support will be needed to be able to retrieve the value of the old field. Also means that the old field is deleted.
Class Versioning - modify a field type	Partial	This will work if the types are compatible, for example an int to an Integer, an int to a String, etc. This may be fully supported in the future using the onSet feature.
Class Versioning - modify a field to a static	No	
Class Versioning - modify a field from a static	Yes	This is equivalent to adding a new field.
Class Versioning - change the name of a class	No	
Change the type of a root	No	

How do I use the Terracotta Sessions Configurator for any container/app server?

Terracotta Sessions Configurator is preconfigured to use a bundled Apache Tomcat web container. Other web containers may be available from the Terracotta Sessions Configurator **File | Servers** menu option.



Each web container generally requires two properties for successful operation. The first is the location of the product home directory (such as CATALINA_HOME or BEA_HOME) and the second is the location of the Java Development Kit (JDK) to use when running the web container (speciffied by the JAVA_HOME property).

Apache Tomcat 5.5
If CATALINA_HOME is not set it will default to the bundled version. If JAVA_HOME is not set it will default to the bundled version.

Apache Tomcat 5.0:
CATALINA_HOME must be set to a valid Tomcat 5.0 installation directory. JAVA_HOME must be set to a valid JDK installation directory. Note that a JDK is required for Tomcat 5.0, as opposed to a JRE.

BEA WebLogic 8.1:
BEA_HOME must be set to a valid BEA 8.1 installation directory. JAVA_HOME must be set to a valid JDK 1.4 installation directory. Note that WLS 8.1 does not work with Java 5.

Web applications that you import into the Configurator are deployed to the currently selected web container. The Terracotta provided sample web applications are pre-deployed to each of the supported web containers.

PERFORMANCE TUNING...

Why does my application run so slowly with Terracotta?

One of the main reasons applications run slowly on Terracotta is due to lock configuration. You must protect critical sections of your code with distributed locks (i.e., synchronization or named locks). If you hold a lock around more code than you need, then you will have high contention for that lock and reduced concurrency. If that is the case, try to minimize the scope of the lock.

If you have too much synchronization or grab more locks than you need to, you will suffer from the overhead of grabbing distributed locks over the network. If that is the case, try to synchronize only where you need to to reduce the number of times you request distributed locks.

Why are the Terracotta server instances in my cluster running very slowly?

If you turn off disk "write" caching on the machines running Terracotta server instances, they'll suffer a substantial performance degradation. Disk caching is often on by default, but may have been turned off in response to this issue.