

Data Locality API



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](#).

- **Introduction**
- [How DSO Clustering Works](#)
- [Platform Concepts](#)
- [Hello Clustered World](#)
- **Setup and Configuration**
- [Planning for a Clustered App](#)
- [Configuring Terracotta DSO](#)
- [Configuration Reference](#)
- [Installation](#)
- **APIs**
- [Using Annotations](#)
- [Cluster Events](#)
- [Data Locality Methods](#)
- [Distributed Cache](#)
- [Clustered Async Data Processing](#)
- **Tool Guides**
- [Developer Console](#)
- [Operations Center](#)
- [tim-get \(TIM Management Tool\)](#)
- [Platform Statistics Recorder](#)
- [Eclipse Plugin](#)
- [Sessions Configurator](#)
- [Clustering Spring Webapp with Sessions Configurator](#)
- [Maven](#)
- [JMX](#)
- **Testing, Tuning, and Deployment**
- [Top 5 Tuning Tips](#)
- [Testing a Clustered App](#)
- [Tuning a Clustered App](#)
- [Deployment Guide](#)
- [Operations Guide](#)
- **FAQs and Troubleshooting**
- [General FAQ](#)
- [DSO Technical FAQ](#)
- [Troubleshooting Guide](#)
- [Gotchas](#)
- [Non-portable Classes](#)
- **Reference**
- [Migrating From DSO](#)
- [Concept and Architecture Guide](#)
- [Examinator Reference Application](#)
- [Clustered Data Structures Guide](#)
- [Integrating Terracotta DSO](#)
- [Clustering Spring Framework](#)
- [Integration Modules Manual](#)
- [AspectWerkz Pattern Language](#)
- [Glossary](#)

Release: 3.6

Publish Date: November, 2011 [Documentation Archive](#) »

DSO Locality API

- [Introduction](#)
- [Usage](#)
- [The Methods](#)

Introduction

Working with clustered data can present additional challenges to applications which must manage this data across a series of application servers. Using [the data locality methods](#) available through the Terracotta Distributed Shared Objects (DSO) API's `DsoCluster` interface, you can meet these challenges by adding locality awareness to your application code. Adding locality awareness, or visibility, provides a number of advantages, including:

- Making applications more efficient with respect to data location by allowing code to be optimized along locality of reference.
- Exposing data access and usage patterns from within an application.
- Facilitating data commitment and eviction operations by providing information on the real-time location of clustered data or its absence.
- Allowing a cache to send work to the appropriate node by providing the location of data that requires processing.
- Providing insight into the effectiveness of data partitioning techniques by verifying the distribution of clustered data.

For more information on data structures, see the Terracotta [DSO Data Structures Guide](#).

Usage

Classes that are injected with cluster awareness can call data locality methods.

The following code example shows the cluster-aware class `ClusterAwareClass` with a member, `localityAwareFoo()`, that calls the data locality method `getNodesWithObjects()`:

```
import com.tc.cluster.DsoCluster;
import com.tc.cluster.DsoClusterListener;
import com.tc.injection.annotations.InjectDsoInstance;

public class ClusterAwareClass implements DsoClusterListener {
    @InjectDsoInstance
    private DsoCluster cluster;
    // ...
    public void localityAwareFoo(final Collection<Object> myObjects) {
        cluster.getNodesWithObjects(myObjects);
    }
}
```

See [Terracotta DSO Cluster Events](#) for more information on injecting cluster awareness into application classes.

The Methods

The data-locality methods available are:

- [getNodesWithObject\(Object object\)](#)
- [getNodesWithObjects\(Object objects ...\)](#)
- [getNodesWithObjects\(Collection objects\)](#)
- [getKeysForOrphanedValues\(Map map\)](#)
- [getKeysForLocalValues\(Map map\)](#)

Note the following characteristics for these methods:

- None returns `null`, making null checks unnecessary. In cases where there is no data to return, an empty Set or Map of empty Sets is returned (depending on the method's return type).
- An `UnclusteredObjectException` is thrown if any of the objects or a Map passed to one of these methods is [not clustered](#).
- The "nodes" targeted by these methods refer to Terracotta nodes (application nodes with Terracotta running). Terracotta server instances are not subject data-locality checks.
- The values in the Sets or Maps of Sets returned by these methods are `DsoNode` instances. Each `DsoNode` instance is uniquely associated with a node and can identify that node by node ID, IP address, and hostname.

getNodesWithObject(Object object)

Use `getNodesWithObject(Object object)` to find which nodes contain the specified object in memory. This method has the following format:

```
java.util.Set<com.tcclient.cluster.DsoNode>
    getNodesWithObject(java.lang.Object object) throws UnclusteredObjectException
```

For example, `getNodesWithObject(myObject)` returns a `Set` containing the nodes which have `myObject` in memory. If `myObject` is not resident in any node's memory, an empty `Set` is returned. If `myObject` is not clustered, an `UnclusteredObjectException` is thrown.

getNodesWithObjects(Object objects ...)

Use `getNodesWithObjects(Object objects ...)` to find which nodes contain the specified objects in memory. This method has the following format:

```
java.util.Map<?, java.util.Set<com.tcclient.cluster.DsoNode>>  
getNodesWithObjects(java.lang.Object... objects) throws UnclusteredObjectException
```

For example, `getNodesWithObject(myObject1, myObject2, myObject3)` returns a `Map` with each key being one of the specified objects (`myObject1`, `myObject2`, or `myObject3`) and each value being a `Set` of nodes which have that object in memory. If an object is not resident in any node's memory, its corresponding value is an empty `Set`. If any of the objects specified is not clustered, an `UnclusteredObjectException` is thrown.

getNodesWithObjects(Collection objects)

Use `getNodesWithObjects(Collection objects)` to find which nodes contain the objects from the specified `Collection` in memory. This method has the following format:

```
java.util.Map<?, java.util.Set<com.tcclient.cluster.DsoNode>>  
getNodesWithObjects(java.util.Collection<?> objects) throws UnclusteredObjectException
```

For example, `getNodesWithObjects(myObjects)` returns a `Map` with each key being an object from the `Collection myObjects` and each value being a `Set` of nodes which have that object in memory. If an object is not resident in any node's memory, its corresponding value is an empty `Set`. If any of the objects in `myObjects` is not clustered, an `UnclusteredObjectException` is thrown.

getKeysForOrphanedValues(Map map)

Use `getKeysForOrphanedValues(Map map)` to generate a `Set` of keys for `Map` values (objects) that are not resident in the memory of any nodes. The `map` passed to `getKeysForOrphanedValues(Map map)` must be clustered and support partial loading. This method has the following format:

```
<K> java.util.Set<K> getKeysForOrphanedValues(java.util.Map<K, ?> map) throws UnclusteredObjectException
```

For example, `getKeysForOrphanedValues(myMap)` returns a `Set` of keys from `myMap` for any objects not found in memory on any node. If `myMap` is not clustered, an `UnclusteredObjectException` is thrown.

getKeysForLocalValues(Map map)

Use `getKeysForLocalValues(Map map)` to generate a `Set` of keys for `Map` values (objects) that are resident in the memory of the local node. The `map` passed to `getKeysForLocalValues(Map map)` must be clustered and support partial loading. This method has the following format:

```
<K> java.util.Set<K> getKeysForLocalValues(java.util.Map<K, ?> map) throws UnclusteredObjectException
```

For example, `getKeysForLocalValues(myMap)` returns a `Set` of keys from `myMap` for any objects found in the memory of the local node. If no objects from `myMap` are found, an empty `Set` is returned. If `myMap` is not clustered, an `UnclusteredObjectException` is thrown.