Oracle® Coherence

Developer's Guide for Oracle Coherence

Release 3.4

E13818-01

November 2008

ORACLE®

Oracle Coherence Developer's Guide for Oracle Coherence, Release 3.4

E13818-01

Primary Author: Thomas Pfaeffle

Contributing Author: Noah Arliss, Jason Howes, Mark Falco, Alex Gleyzer, Gene Gleyzer, David Leibs, Andy Nguyen, Brian Oliver, Patrick Peralta, Cameron Purdy, Jonathan Purdy, Everet Williams, Tom Beerbower, John Speidel

Contents

25 How to Modify Report Batch

26 Analyzing Reporter Content

27 How to Run a Report on Demand

28 Configuring Custom MBeans

29 How to Manage Custom MBeans Within the Cluster

A Production Checklist

B Types of Caches in Coherence

C Cache Semantics

D Cache Configuration Elements

E Cache Configuration Parameter Macros

F Sample Cache Configurations

G Sample CacheStores

H Operational Configuration Elements

K MBean Configuration Elements

L Command Line Overrides

M Platform-Specific Deployment Considerations

N Best Practices for Coherence Extend

O Scaling Out Your Data Grid Aggregations Linearly

List of Figures

Preface

Oracle Coherence is a JCache-compliant in-memory caching and data management solution for clustered J2EE applications and application servers. Coherence makes sharing and managing data in a cluster as simple as on a single server. It accomplishes this by coordinating updates to the data using clusterwide concurrency control, replicating and distributing data modifications across the cluster using the highest performing clustered protocol available, and delivering notifications of data modifications to any servers that request them. Developers can easily take advantage of Coherence features using the standard Java collections API to access and modify data, and use the standard JavaBean event model to receive data change notifications. Functionality such as HTTP Session Management is available out-of-the-box for applications deployed to WebLogic, WebSphere, Tomcat, Jetty and other Servlet 2.2, 2.3 and 2.3 compliant application servers.

Audience

This document is targeted at software developers and architects. It provides detailed technical information on creating and using the Coherence cache and for writing and deploying Java applications that interact with it.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

To reach AT&T Customer Assistants, dial 711 or 1.800.855.2880. An AT&T Customer Assistant will relay information between the customer and Oracle Support Services at 1.800.223.1711. Complete instructions for using the AT&T relay services are available at `http://www.consumer.att.com/relay/tty/standard2.html`. After the AT&T Customer Assistant contacts Oracle Support Services, an Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process.

Related Documents

For more information, see the following documents in the Oracle Coherence Release 3.4 documentation set:

- *User's Guide for Oracle Coherence*
- *Getting Started with Oracle Coherence*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Part I

Coherence Features

This section contains the following chapters:

1

Create and Use Coherence Caches

The simplest and most flexible way to create caches in Coherence is to use the cache configuration descriptor to define attributes and names for your application's or cluster's caches, and to instantiate the caches in your application code referring to them by name that matches the names or patterns as defined in the descriptor.

This approach to configuring and using Coherence caches has several very important benefits. It separates the cache initialization and access logic for the cache in your application from its attributes and characteristics. This way your code is written in a way that is independent of the cache type that will be used in your application deployment and changing the characteristics of each cache (such as Rich Text cache type, cache eviction policy, and cache type-specific attributes, and so on) can be done without making any changes to the code whatsoever. It enables you to create multiple configurations for the same set of named caches and to instruct your application to use the appropriate configuration at deployment time by specifying the descriptor to use in the java command line when the node JVM is started.

Creating a Cache in Your Application

To instantiate a cache in your application code, you need to:

1. Make sure that `coherence.jar` is in your classpath.
2. Use `CacheFactory.getCache()` to access the cache in your code.

Your code will look similar to the following:

```
import com.tangosol.net.CacheFactory;
import com.tangosol.net.NamedCache;

...

NamedCache cache = CacheFactory.getCache("VirtualCache");
```

Now you can retrieve and store objects in the cache, using the `NamedCache` API, which extends the standard `java.util.Map` interface, adding several additional capabilities that provide concurrency control (`ConcurrentMap` interface), ability to listen for cache changes (`ObservableMap` interface) and ability to query the cache (`QueryMap` interface).

Example 1–1 illustrates typical cache operations in a Java program.

Example 1–1 Typical Cache Operations in a Java Program

```
...
// simple retrieve and update cycle
String key = "key";
```

```
// retrieve the object
MyValue value = (MyValue) cache.get(key);

// Use and modify the object
// ...

// put the new value back
cache.put(key, value);
...
```

Configuring the Caches

The cache attributes and settings are defined in the cache configuration descriptor. Cache attributes determine the cache type (what means and resources the cache will use for storing, distributing and synchronizing the cached data) and cache policies (what happens to the objects in the cache based on cache size, object longevity and other parameters).

The structure of the cache configuration descriptor (described in detail by the `cache-config.dtd` included in the `coherence.jar`) consists of two primary sections: `caching-schemes` section and `caching-scheme-mapping` section.

The `caching-schemes` section is where the attributes of a cache or a set of caches get defined. The caching schemes can be of several types, each with its own set of attributes. The caching schemes can be defined completely from scratch, or can incorporate attributes of other existing caching schemes, referring to them by their `scheme-names` (using a `scheme-ref` element) and optionally overriding some of their attributes to create new caching schemes. This flexibility enables you to create caching scheme structures that are easy to maintain, foster reuse and are very flexible.

The `caching-scheme-mapping` section is where the specific cache name or a naming pattern is attached to the cache scheme that defines the cache configuration to use for the cache that matches the name or the naming pattern. So if we would like to define the cache descriptor for the cache we mentioned in the previous section (`VirtualCache`), it may look something like the following:

Example 1–2 Sample Cache Configuration File

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">

<cache-config>
    <caching-scheme-mapping>
    <!--
    Caches with any name will be created as default replicated.
    -->
    <cache-mapping>
        <cache-name>*</cache-name>
        <scheme-name>default-replicated</scheme-name>
    </cache-mapping>
    </caching-scheme-mapping>

    <caching-schemes>
        <!--
        Default Replicated caching scheme.
        -->
        <replicated-scheme>
            <scheme-name>default-replicated</scheme-name>
```

```
            <service-name>ReplicatedCache</service-name>
            <backing-map-scheme>
                <class-scheme>
                    <scheme-ref>default-backing-map</scheme-ref>
                </class-scheme>
            </backing-map-scheme>
        </replicated-scheme>

        <!--
        Default backing map scheme definition used by all
        The caches that do not require any eviction policies
        -->
        <class-scheme>
            <scheme-name>default-backing-map</scheme-name>
            <class-name>com.tangosol.util.SafeHashMap</class-name>
        </class-scheme>

    </caching-schemes>
</cache-config>
```

The above cache configuration descriptor specifies that all caches will be created (including our `VirtualCache` cache) using the default-replicated caching scheme. It defines the default-replicated caching scheme as a replicated-scheme, using a service named `ReplicatedCache` and using the backing map named default-backing-map, which is defined as a class `com.tangosol.util.SafeHashMap` (the default backing map storage that Coherence uses when no eviction policies are required).

Then, at a later point, let's say we decide that, since the number of entries that our cache is holding is too large and updates to the objects too frequent to use a replicated cache, we want our VirtualCache cache to become a distributed cache instead (while keeping all other caches replicated). To accommodate these new circumstances, we can change the cache configuration by adding the following `cache-scheme` definition for the distributed cache to the `caching-schemes` section:

Example 1–3 cache-scheme Definition for a Distributed Cache

```
<!--
Default Distributed caching scheme.
-->
<distributed-scheme>
    <scheme-name>default-distributed</scheme-name>
    <service-name>DistributedCache</service-name>
    <backing-map-scheme>
        <class-scheme>
            <scheme-ref>default-backing-map</scheme-ref>
        </class-scheme>
    </backing-map-scheme>
</distributed-scheme>
```

Then mapping the `VirtualCache` cache to it in the `caching-schemes-mapping` section:

Example 1–4 Mapping to a Distributed Cache

```
<cache-mapping>
    <cache-name>VirtualCache</cache-name>
    <scheme-name>default-distributed</scheme-name>
</cache-mapping>
```

The resulting cache definition descriptor will look similar to Example 1–5:

Example 1–5 Configuration for a Distributed Cache

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">

<cache-config>
    <caching-scheme-mapping>
        <!--
        Caches with any name will be created as default replicated.
        -->
        <cache-mapping>
            <cache-name>*</cache-name>
            <scheme-name>default-replicated</scheme-name>
        </cache-mapping>
        <cache-mapping>
            <cache-name>VirtualCache</cache-name>
            <scheme-name>default-distributed</scheme-name>
        </cache-mapping>
    </caching-scheme-mapping>

    <caching-schemes>
        <!--
        Default Replicated caching scheme.
        -->
        <replicated-scheme>
            <scheme-name>default-replicated</scheme-name>
            <service-name>ReplicatedCache</service-name>

            <backing-map-scheme>
                <class-scheme>
                    <scheme-ref>default-backing-map</scheme-ref>
                </class-scheme>
            </backing-map-scheme>
        </replicated-scheme>

        <!--
        Default Distributed caching scheme.
        -->
        <distributed-scheme>
            <scheme-name>default-distributed</scheme-name>
            <service-name>DistributedCache</service-name>

            <backing-map-scheme>
                <class-scheme>
                    <scheme-ref>default-backing-map</scheme-ref>
                </class-scheme>
            </backing-map-scheme>
        </distributed-scheme>

        <!--
        Default backing map scheme definition used by all
        The caches that do not require any eviction policies
        -->
        <class-scheme>
            <scheme-name>default-backing-map</scheme-name>

            <class-name>com.tangosol.util.SafeHashMap</class-name>
        </class-scheme>

    </caching-schemes>
</cache-config>
```

When we revise and deploy the descriptor and restart the cluster, the `VirtualCache` cache will be a distributed cache instead of replicated, all without any changes to the code we wrote.

Cache Configuration Descriptor Location

A few words about how to instruct Coherence where to find the cache configuration descriptor. Without specifying anything in the Java command line, Coherence will attempt to use the cache configuration descriptor named `coherence-cache-config.xml` that it finds in the classpath. Since Coherence ships with this file packaged into the coherence.jar, unless you place another file with the same name in the classpath location preceding coherence.jar, that is the one that Coherence will use. You can tell Coherence to use a different default descriptor by using the `-Dtangosol.coherence.cacheconfig` java command line property as follows:

```
java -Dtangosol.coherence.cacheconfig=/cfg/my-config.xml AppServer
```

The above command instructs Coherence to use `my-config.xml` file in `/cfg` directory as the default cache configuration descriptor. As you can see, this capability can give you the flexibility to modify the cache configurations of your applications without making any changes to the application code and by simply specifying different cache configuration descriptors at application deployment or start-up.

Putting It all Together: Your First Coherence Cache Example

Let's try walking through creating a working example cache using the caches and the cache configuration descriptor we described in the previous section. The easiest way to initially do that is to use the Coherence command line application. A couple of general comments regarding this example before we get started:

- In the examples we refer to the 'nodes' or 'JVMs'. We make no assumption regarding where they will run - you can run all of them on the same machine multiple machines or a combination of multiple nodes per machine and multiple machines. To see the clustered cache in action you will need at least 2 nodes to see the JVMs sharing data (all the following examples were captured with 2 JVMs on a single machine).
- This example uses Windows conventions and commands but it will work equally well in any of the UNIX environments (with the appropriate adjustments for the UNIX commands and conventions) and we encourage you to try it on multiple machines with different operating systems, as this is the way Coherence is designed to function: on multiple platforms simultaneously.

Setting Up Your Test Environment

To set up the test environment, you will need install Coherence by unzipping the software distribution in the desired location on one or more machines.

The `coherence/examples` directory contains the following examples that will be used in this exercise:

- `examples/config/explore-config.xml`

  is the configuration descriptor used by the test environment example.

- `examples/java/com/tangosol/examples/explore/SimpleCacheExplorer.java`

  is the Java class that demonstrates how you can access the cache from a command line.

To deploy and run it, you need to execute the following Java command line (from the coherence directory):

- **In Windows:**

```
java -cp ./lib/coherence.jar;./examples/java
     -Dtangosol.coherence.cacheconfig=./examples/config/explore-config.xml
     com.tangosol.examples.explore.SimpleCacheExplorer
```

- **In UNIX:**

```
java -cp ./lib/coherence.jar:./examples/java
     -Dtangosol.coherence.cacheconfig=./examples/config/explore-config.xml
     com.tangosol.examples.explore.SimpleCacheExplorer
```

You should see something like the following when you bring it up:

Example 1–6 Output from Starting a Coherence Server

```
D:\coherence>java -cp ./lib/coherence.jar;./examples/java
-Dtangosol.coherence.cacheconfig=./examples/config/explore-config.xml
com.tangosol.examples.explore.SimpleCacheExplorer
2008-09-15 16:54:18.745 Oracle Coherence 3.4/405(thread=main, member=n/a): Loaded
operational configuration from
resource "jar:file:/D:/coherence/lib/coherence.jar!/tangosol-coherence.xml"
2008-09-15 16:54:18.745 Oracle Coherence 3.4/405 (thread=main, member=n/a): Loaded
operational overrides from
resource
"jar:file:/D:/coherence/lib/coherence.jar!/tangosol-coherence-override-dev.xml"
2008-09-15 16:54:18.745 Oracle Coherence 3.4/405 (thread=main, member=n/a):
Optional configuration override
"/tangosol-coherence-override.xml" is not specified
2008-09-15 16:54:18.755 Oracle Coherence 3.4/405 (thread=main, member=n/a):
Optional configuration override
"/custom-mbeans.xml" is not specified

Oracle Coherence Version 3.4/405
Grid Edition: Development mode
Copyright (c) 2000-2008 Oracle. All rights reserved.

2008-09-15 16:54:18.945 Oracle Coherence GE 3.4/405(thread=main, member=n/a):
Loaded cache configuration from
file "D:\coherence\examples\config\explore-config.xml"
2008-09-15 16:54:19.716 Oracle Coherence GE 3.4/405(thread=Cluster, member=n/a):
Service Cluster joined the
cluster with senior service member n/a
2008-09-15 16:54:22.921 Oracle Coherence GE 3.4/405(thread=Cluster, member=n/a):
Created a new cluster
"cluster:0x19DB" with Member(Id=1, Timestamp=2008-09-15 16:54:19.396,
Address=xxx.xxx.xxx.xxx:8088, MachineId=6522,
Location=site:mydomain.com,machine:mycomputer,process:3500,
Role=TangosolSimpleCacheExplorer, Edition=Grid Edition, Mode=Development,
CpuCount=1, SocketCount=1) UID=0x0A8F9C7A0000011BE70BDE04197A1F98
2008-09-15 16:54:23.001 Oracle Coherence GE 3.4/405(thread=ReplicatedCache,
member=1): Service ReplicatedCache
joined the cluster with senior service member 1
```

```
Command:
```

Type `Help` to view the `SimpleCacheExplorer` command line options. You may need to press **Enter** to display the Command: prompt.

Example 1–7 Output from the help Command

```
Command: help

clear
get
keys
info
put
quit
remove

Command:
```

Type `info` to display configuration and member information (please note that in the following example there are 2 cluster members active):

Example 1–8 Output from the info Command

```
Command: info

>> VirtualCache cache is using a cache-scheme named 'default-replicated' defined
as:
<replicated-scheme>
  <scheme-name>default-replicated</scheme-name>
  <service-name>ReplicatedCache</service-name>
  <backing-map-scheme>
    <class-scheme>
      <scheme-ref>default-backing-map</scheme-ref>
    </class-scheme>
  </backing-map-scheme>
</replicated-scheme>

>> The following member nodes are currently active:
Member(Id=1, Timestamp=2008-09-15 16:54:19.396, Address=xxx.xxx.xxx.xxx:8088,
MachineId=6522, Location=site:mydomain.com,machine:
mycomputer,process:3500, Role=TangosolSimpleCacheExplorer) <-- this node
Member(Id=2, Timestamp=2008-09-15 17:19:56.096, Address=xxx.xxx.xxx.xxx:8089,
MachineId=6522, Location=site:mydomain.com,machine:
mycomputer,process:3892, Role=TangosolSimpleCacheExplorer)

Command:
```

You can also put a value into the cache:

Example 1–9 Putting a Value into the Cache

```
Command: put 1 One

>> Put Complete

Command:
```

And retrieve a value from the cache:

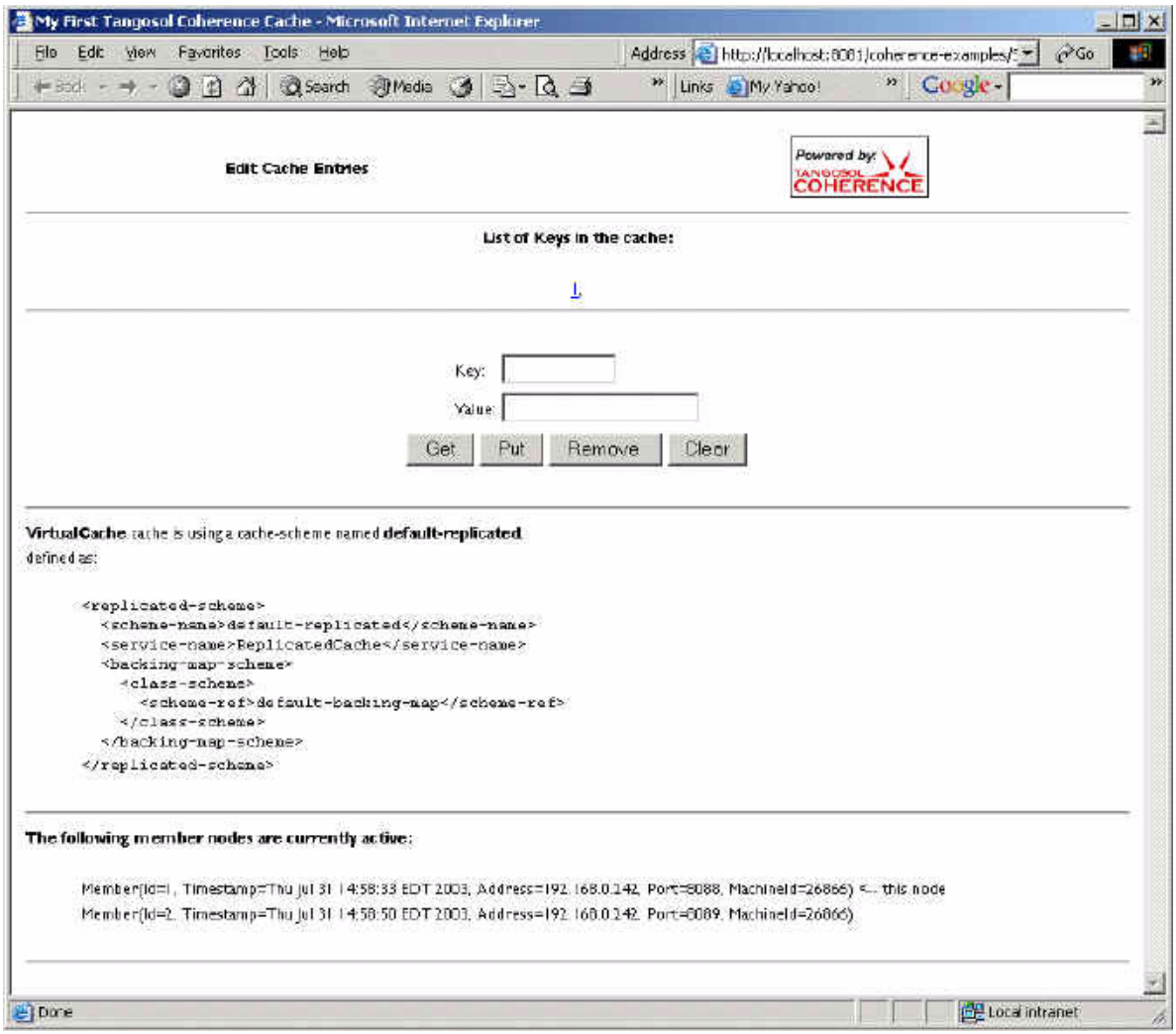
Example 1–10 Retrieving a Value from the Cache

```
Command: get 1

>> Value is One

Command:
```

Try these commands from multiple sessions and see the results. The `examples/jsp/explore/SimpleCacheExplorer.jsp` is the JSP file that can be used with your favorite application server:

- To deploy and run it, you will need to deploy the JSP to the default web applications directory of your application server (along with the contents of the `examples/jsp/images` directory), modify the server start-up script to make sure that the classpath includes `coherence.jar`, and specify the location of the cache configuration file on the Java command line using the `-Dtangosol.coherence.cacheconfig` option (for example, `-Dtangosol.coherence.cacheconfig=$COHERENCE_HOME/examples/config/explore-config.xml`).
- You can then start one or more instances of the application server (on different machines or different ports) and access the `SimpleCacheExplorer.jsp` from the browser. You should see something like the following when you bring it up:

Figure 1–1 Interacting with the Cache through a Browser



This figure is described in the text.

****

As with the command line application try adding, updating, and removing entries from multiple instances of the application server. Also please notice the information about the cache configuration and cluster membership at the bottom of the page. As cluster members are added and removed, this information will change.

Modifying the Cache Configuration

When you are comfortable with the test setup, let's change the cache configuration and test our changes, using this simple test harness. Please remember that after each cache configuration change all the cluster members need to be shut down and then restarted (whether you are using application server instances or just plain java JVMs). All our tests are configured to use `coherence/examples/config/explore-config.xml`, so this is the file that must be edited to make cache configuration changes. Let's make the first change we described previously, changing the `VirtualCache` to be a distributed cache by adding the following (**bolded**) sections:

Example 1–11 Specifying a Distributed Cache in the cache-config File

```
<?xml version="1.0"?> <!DOCTYPE cache-config SYSTEM "cache-config.dtd">
<cache-config> <caching-scheme-mapping> <!-- Caches with any name will be created
as default replicated. --> <cache-mapping> <cache-name>*</cache-name>
<scheme-name>default-replicated</scheme-name>
 </cache-mapping>
<cache-mapping><cache-name>VirtualCache</cache-name><scheme-name>default-distribut
ed</scheme-name></cache-mapping></caching-scheme-mapping>

<caching-schemes> <!-- Default Replicated caching scheme. --> <replicated-scheme>
<scheme-name>default-replicated</scheme-name>
<service-name>ReplicatedCache</service-name>

<backing-map-scheme> <class-scheme> <scheme-ref>default-backing-map</scheme-ref>
</class-scheme> </backing-map-scheme> </replicated-scheme>

<!--Default Distributed caching scheme.-->
<distributed-scheme>      <scheme-name>default-distributed</scheme-name>
<service-name>DistributedCache</service-name>

<backing-map-scheme><class-scheme><scheme-ref>default-backing-map</scheme-ref></cl
ass-scheme></backing-map-scheme></distributed-scheme>
<!-- Default backing map scheme definition used by all The caches that do not
require any eviction policies --> <class-scheme>
<scheme-name>default-backing-map</scheme-name>
<class-name>com.tangosol.util.SafeHashMap</class-name> </class-scheme>
</caching-schemes> </cache-config>
```

After the changes are saved, the test instances are restarted and you have had a chance to do some test data entry to see how the cache behaves, you should see the following in the cache configuration section of the tests:

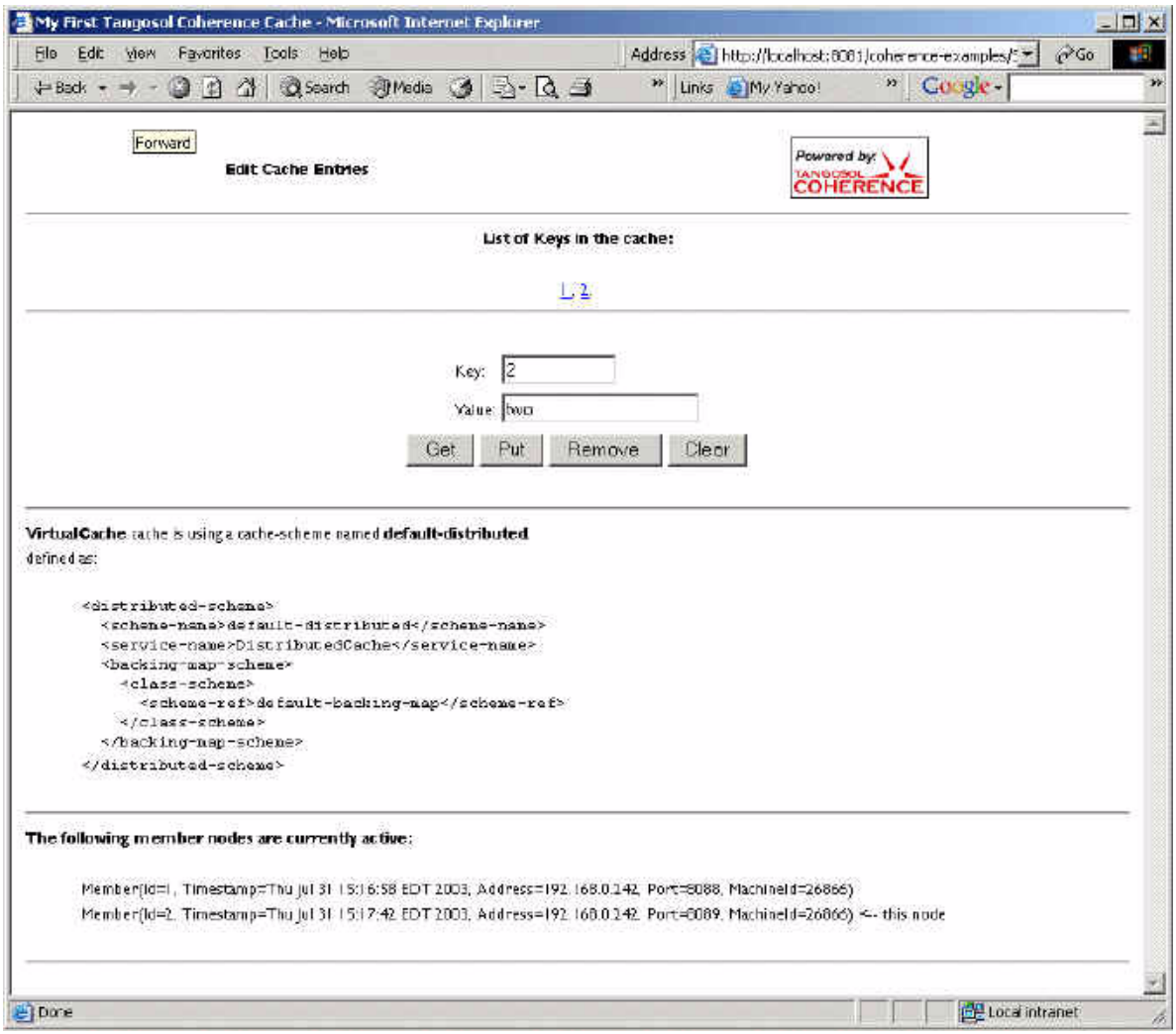
- `SimpleCacheExplorer.java`:

Example 1–12 Running SimpleCacheExplorer.java with a Distributed Cache

```
Command: info

>> VirtualCache cache is using a cache-scheme named 'default-distributed' defined
as:
<distributed-scheme>
  <scheme-name>default-distributed</scheme-name>
  <service-name>DistributedCache</service-name>
  <backing-map-scheme>
    <class-scheme>
      <scheme-ref>default-backing-map</scheme-ref>
    </class-scheme>
  </backing-map-scheme>
</distributed-scheme>

>> The following member nodes are currently active:
Member(Id=1, Timestamp=2008-09-15 17:53:22.701, Address=xxx.xxx.xxx.xxx:8088,
MachineId=6522, Location=site:mydomain.com,
machine:mycomputer,process:3156, Role=TangosolSimpleCacheExplorer) <-- this node
Member(Id=2, Timestamp=2008-09-15 17:54:37.619, Address=xxx.xxx.xxx.xxx:8089,
MachineId=6522, Location=site:mydomain.com,
machine:mycomputer,process:916, Role=TangosolSimpleCacheExplorer)

Command:
```

- `SimpleCacheExplorer.jsp`:

Figure 1–2 Running SimpleCacheExplorer.jsp with a Distributed Cache



This figure is described in the text.

****

As you can see, our VirtualCache cache is now distributed according to the cache configuration descriptor.

Now let's add an eviction policy for our default distributed cache, limiting its size to 5 entries (per node) and setting the entry expiry to 60 seconds with an LRU eviction policy. To do that we need to make the following (**bolded**) changes to our descriptor:

Example 1–13 Adding an Eviction Policy to a cache-config File

```
<?xml version="1.0"?> <!DOCTYPE cache-config SYSTEM "cache-config.dtd">
<cache-config> <caching-scheme-mapping> <!-- Caches with any name will be created
as default replicated. --> <cache-mapping> <cache-name>*</cache-name>
<scheme-name>default-replicated</scheme-name> </cache-mapping> <cache-mapping>
<cache-name>VirtualCache</cache-name>
<scheme-name>default-distributed</scheme-name> </cache-mapping>
</caching-scheme-mapping> <caching-schemes> <!-- Default Replicated caching
```

```
scheme. --> <replicated-scheme> <scheme-name>default-replicated</scheme-name>
<service-name>ReplicatedCache</service-name> <backing-map-scheme> <class-scheme>
<scheme-ref>default-backing-map</scheme-ref> </class-scheme> </backing-map-scheme>
</replicated-scheme> <!-- Default Distributed caching scheme. -->
<distributed-scheme> <scheme-name>default-distributed</scheme-name>
<service-name>DistributedCache</service-name>
<backing-map-scheme><local-scheme><scheme-ref>default-eviction</scheme-ref><evicti
on-policy>LRU</eviction-policy><high-units>5</high-units><expiry-delay>60</expiry-
delay></local-scheme></backing-map-scheme> </distributed-scheme> <!-- Default
backing map scheme definition used by all The caches that do not require any
eviction policies --> <class-scheme>
<scheme-name>default-backing-map</scheme-name>
<class-name>com.tangosol.util.SafeHashMap</class-name> </class-scheme><!--Default
eviction policy
scheme.--><local-scheme><scheme-name>default-eviction</scheme-name><eviction-polic
y>HYBRID</eviction-policy><high-units>0</high-units><expiry-delay>3600</expiry-del
ay></local-scheme></caching-schemes> </cache-config>
```

Note that we defined a general purpose local-scheme 'default-eviction' (with no size limit, 5 minute expiry and a HYBRID eviction policy) and then used it by reference (using scheme-ref) for our default-distributed scheme definition, overriding it's configuration settings to match our requirements.

After the changes are saved, the test instances are restarted and you have had a chance to do some test data entry to see how the cache behaves, you should see the following in the cache configuration section of the tests:

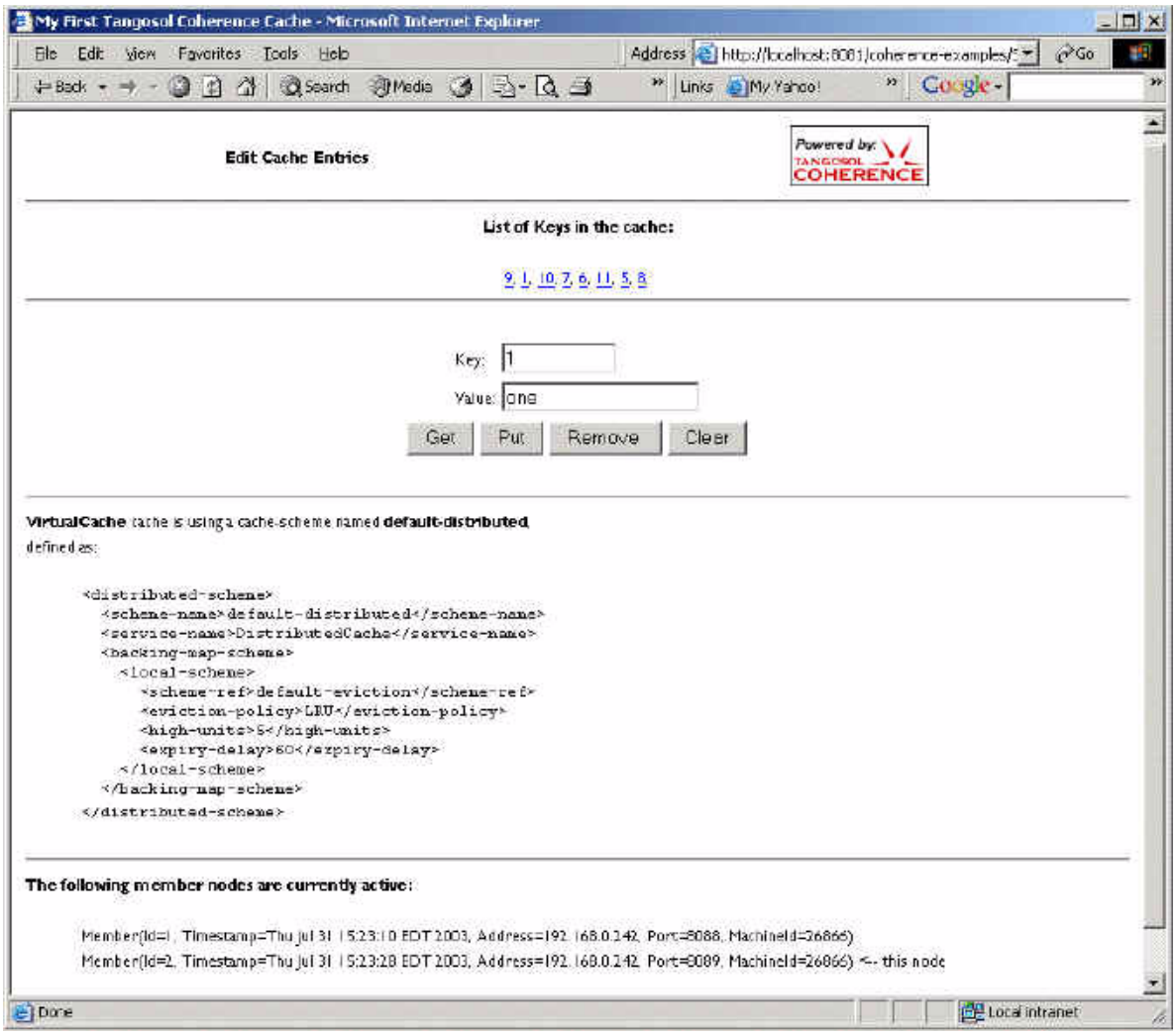
- `SimpleCacheExplorer.java`:

Example 1–14 Running SimpleCacheExplorer.java with an Eviction Policy

```
Command: info

>> VirtualCache cache is using a cache-scheme named 'default-distributed' defined
as:
<distributed-scheme>
  <scheme-name>default-distributed</scheme-name>
  <service-name>DistributedCache</service-name>
  <backing-map-scheme>
    <local-scheme>
      <scheme-ref>default-eviction</scheme-ref>
      <eviction-policy>LRU</eviction-policy>
      <high-units>5</high-units>
      <expiry-delay>60</expiry-delay>
    </local-scheme>
  </backing-map-scheme>
</distributed-scheme>

>> The following member nodes are currently active:
Member(Id=1, Timestamp=2008-09-15 18:10:23.148, Address=xxx.xxx.xxx.xxx:8088,
MachineId=6522, Location=site:mydomain.com,
machine:mycomputer,process:2960, Role=TangosolSimpleCacheExplorer) <-- this node
Member(Id=2, Timestamp=2008-09-15 18:10:35.957, Address=xxx.xxx.xxx.xxx:8089,
MachineId=6522, Location=site:mydomain.com,
machine:mycomputer,process:3348, Role=TangosolSimpleCacheExplorer)

Command:
```

- `SimpleCacheExplorer.jsp`:

Figure 1–3 Running SimpleCacheExplorer.jsp with an Eviction Policy



This figure is described in the text.

****

Try doing some puts and gets, carefully noting the time you last updated the specific entries. You should see that the number of entries does not exceed 5 entries per node (so if you have 2 nodes running the number of entries should not exceed 10, for 3 nodes - 15, and so on) and entries either expire after they have not been updated for 60 seconds, or when you add the 6th entry (with the least recently touched entries being 'evicted' from the cache first. (Hint: use the `keys` command in the `SimpleCacheExplorer.java` to see the list of keys in the cache.)

These examples show you the general approach to modifying the cache configurations without making any code changes (as you no doubt noticed we did not touch our test application's code). Please refer to the `cache-config.dtd`, which can be found in the coherence.jar for full details on the available cache configuration descriptor settings and the explanation of their meaning and possible settings.

2

Implement Transactions, Locks, and Concurrency

Coherence provides several different options to support locking, transactions, and concurrent access to data.

Concurrency Options

Coherence provides several options for managing concurrent access to data.

Table 2–1 Coherence Concurrent Access Options

Option Name	Description
Explicit Locking	The `ConcurrentMap` interface (part of the `NamedCache` interface) supports explicit locking operations. Many developers find this simple locking API to be the most natural approach.
Transactions	The `TransactionMap` API builds on top of the explicit locking operations to support ACID-style transactions.
Container Integration	For transaction management in a Java EE container, Coherence provides a JCA resource adaptor to allow transactions to be managed by using JTA. Although Coherence does not currently support XA transactions, it can participate in XA transactions as the last resource.
Entry Processors	Coherence also supports a lock-free programming model through the `EntryProcessor` API. For many transaction types, this minimizes contention and latency and improves system throughput, without compromising the fault-tolerance of data operations.
Data Source Integration	Guidelines on maintaining caches with local (non XA) data resources.

Explicit Locking

The standard `NamedCache` interface extends the `ConcurrentMap` interface which includes basic locking methods. Locking operations are applied at the entry level by requesting a lock against a specific key in a `NamedCache`:

Example 2–1 Applying Locking Operations on a Cache

```
...
NamedCache cache = CacheFactory.getCache("dist-cache");
Object key = "example_key";
cache.lock(key, -1);
try
```

```
    {
    Object value = cache.get(key);
    // application logic
    cache.put(key, value);
    }
finally
    {
    // Always unlock in a "finally" block
    // to ensure that uncaught exceptions
    // don't leave data locked
    cache.unlock(key);
    }
...
```

Coherence lock functionality is similar to the Java `synchronized` keyword and the C# `lock` keyword: locks only block locks. Threads must cooperatively coordinate access to data through appropriate use of locking. If a thread has locked the key to an item, another thread can read the item without locking.

Locks are unaffected by server failure (and will failover to a backup server.) Locks are immediately released when the lock owner (client) fails.

Locking behavior varies depending on the timeout requested and the type of cache. A timeout of -1 will block indefinitely until a lock can be obtained, 0 will return immediately, and a value greater than 0 will wait the specified number of milliseconds before timing out. The boolean return value should be examined to ensure the caller has actually obtained the lock. See ConcurrentMap.lock() for more details. Note that if a timeout value is not passed to `lock()` the default is 0. With replicated caches, the entire cache can be locked by using `ConcurrentMap.LOCK_ALL` as the key, although this is usually not recommended. This operation is not supported with partitioned caches.

In both replicated and partitioned caches, gets are permitted on keys that are locked. In a replicated cache, puts are blocked, but they are not blocked in a partitioned cache. When a lock is in place, it is the responsibility of the caller (either in the same thread or the same cluster node, depending on the `lease-granularity` configuration) to release the lock. This is why locks should always be released with a finally clause (or equivalent). If this is not done, unhandled exceptions may leave locks in place indefinitely. For more information on `lease-granularity` configuration, see "DistributedCache Service Parameters".

Transactions

A `TransactionMap` can be used to update multiple items in a Coherence cache in a transaction. To perform transactions with a `TransactionMap`, the Java Transaction API (JTA) libraries must be present in the classpath. `TransactionMaps` are created using the `CacheFactory`:

```
NamedCache     cache = CacheFactory.getCache("dist-cache");
TransactionMap tmap  = CacheFactory.getLocalTransaction(cache);
```

Before using a `TransactionMap`, concurrency and isolation levels should be set to the desired level. For most short lived, highly concurrent transactions, a concurrency level of `CONCUR_PESSIMISTIC` and an isolation level of `TRANSACTION_REPEATABLE_GET` is necessary. For most long lived transactions that don't occur as often, `CONCUR_OPTIMISTIC` and `TRANSACTION_REPEATABLE_GET` are good defaults. The combination of concurrency level `CONCUR_PESSIMISTIC` and isolation level `TRANSACTION_SERIALIZABLE` will lock the entire cache. As mentioned

previously, partitioned caches do not allow the entire cache to be locked, thus this mode will not work for partitioned caches. (In general, this level of isolation is not required for reliable transaction processing.) Queries against caches (calling `keySet(Filter)`) or `entrySet(Filter)`) are always performed with `READ_COMMITTED` isolation. For more information about concurrency and isolation levels, see the `TransactionMap` API.

Here is how to set the isolation and concurrency levels:

```
tmap.setTransactionIsolation(TransactionMap.TRANSACTION_REPEATABLE_GET);
tmap.setConcurrency(TransactionMap.CONCUR_PESSIMISTIC);
```

Now the `TransactionMap` can be used to update the cache in a transaction:

Example 2–2 Updating the Cache in a Transaction

```
import com.tangosol.net.CacheFactory;
import com.tangosol.net.NamedCache;

import com.tangosol.util.Base;
import com.tangosol.util.TransactionMap;

import java.util.Collection;
import java.util.Collections;


public class TxMapSample
        extends Base
    {
    public static void main(String[] args)
        {
        // populate the cache
        NamedCache cache = CacheFactory.getCache("dist-cache");

        String key1 = "key1";
        String key2 = "key2";

        cache.put(key1, new Integer(1));
        cache.put(key2, new Integer(1));

        out("Initial value for key 1: " + cache.get(key1));
        out("Initial value for key 2: " + cache.get(key2));

        // create one TransactionMap per NamedCache
        TransactionMap mapTx = CacheFactory.getLocalTransaction(cache);
        mapTx.setTransactionIsolation(TransactionMap.TRANSACTION_REPEATABLE_GET);
        mapTx.setConcurrency(TransactionMap.CONCUR_PESSIMISTIC);

        // gather the cache(s) into a Collection
        Collection txnCollection = Collections.singleton(mapTx);
        boolean    fTxSucceeded  = false;

        try
            {
            // start the transaction
            mapTx.begin();

            int i1 = ((Integer)mapTx.get(key1)).intValue();
            mapTx.put(key1, new Integer(++i1));

            int i2 = ((Integer)mapTx.get(key2)).intValue();
```

```
            mapTx.put(key2, new Integer(++i2));

            // commit the changes
            fTxSucceeded = CacheFactory.commitTransactionCollection(txnCollection,
1);
            }

        catch (Throwable t)
            {
            // rollback
            CacheFactory.rollbackTransactionCollection(txnCollection);
            }

        int v1 = ((Integer) cache.get(key1)).intValue();
        int v2 = ((Integer) cache.get(key2)).intValue();

        out("Transaction " + (fTxSucceeded ? "succeeded" : "did not succeed"));

        out("Updated value for key 1: " + v1);
        out("Updated value for key 2: " + v2);

        azzert(v1 == 2, "Expected value for key1 == 2");
        azzert(v2 == 2, "Expected value for key2 == 2");
        }
    }
```

The `CacheFactory` API provides helper methods for committing and rolling back a collection of `TransactionMap` instances. The `commit` method uses a conventional two-phase commit (2PC) protocol. However, as with any 2PC implementation, if one of the resources fails to commit during the second phase ("commit"), the transaction may end up partially committed. Unlike traditional 2PC implementations, Coherence can guarantee that a given server will not enter an "in doubt" state during the commit phase, but other failures are possible (for example, write-through caching can cause persistent failures). Additionally, as the transaction log is stored only on the client, a client-side failure during the "commit" phase might result in a partial commit. As the "commit" phase is nonblocking (any required locks are acquired before the start of the "commit" phase), the "commit" phase is much shorter (usually no more than a few milliseconds) than the "prepare" phase and thus the exposure, while nonzero, is minimal for typical workloads.

`MapListeners` registered with caches that participate in a transaction will receive a `MapEvent` as each item is committed. There are no guarantees that events will be fired in the order that they appear in the transaction. Additionally, if the transaction updates an item multiple times, only one event will be dispatched, reflecting the final state of the item.

Container Integration

JCA

Coherence ships with a JCA 1.0 compliant resource adaptor that can be used to manage transactions in a Java EE container. It is packaged in a resource adaptor archive (RAR) file that can be deployed to any Java EE container compatible with JCA 1.0. When deployed, JTA can be used to execute the transaction:

Example 2–3 Configuration for a JCA Container

```
String          key = "key";
Context         ctx = new InitialContext();
UserTransaction tx  = null;
try
    {
    // the transaction manager from container
    tx = (UserTransaction) ctx.lookup("java:comp/UserTransaction");
    tx.begin();

    // the try-catch-finally block below is the block of code
    // that could be on an EJB and therefore automatically within
    // a transactional context
    CacheAdapter adapter = null;
    try
        {
        adapter = new CacheAdapter(ctx, "tangosol.coherenceTx",
                CacheAdapter.CONCUR_OPTIMISTIC,
                CacheAdapter.TRANSACTION_GET_COMMITTED, 0);

        NamedCache cache = adapter.getNamedCache("dist-test",
                getClass().getClassLoader());

        int n = ((Integer)cache.get(key)).intValue();
        cache.put(key, new Integer(++n));
        }
    catch (Throwable t)
        {
        String sMsg = "Failed to connect: " + t;
        System.err.println(sMsg);
        t.printStackTrace(System.err);
        }
    finally
        {
        try
            {
            adapter.close();
            }
        catch (Throwable ex)
            {
            System.err.println("SHOULD NOT HAPPEN: " + ex);
            }
        }
    }
finally
    {
    try
        {
        tx.commit();
        }
    catch (Throwable t)
        {
        String sMsg = "Failed to commit: " + t;
        System.err.println(sMsg);
        }
    }
```

XA

Coherence can participate in an XA transaction as the last resource. This feature is supported by most transaction managers and is known by various names, such as "Last Resource Commit" or "Last Participant." In this scenario, the completion of a transaction would involve the following steps:

- prepare is called on all XA resources
- commit is called on the Coherence transaction
- if the commit is successful, commit is called on the other XA participants in the transaction.

Refer to your transaction manager's documentation on XA last resource configuration for further details on this technique.

Entry Processors

The `InvocableMap` superinterface of `NamedCache` allows for concurrent lock-free execution of processing code within a cache. This processing is performed by an `EntryProcessor`. In exchange for reduced flexibility compared to the more general purpose `TransactionMap` and `ConcurrentMap` explicit locking APIs, `EntryProcessors` provide the highest levels of efficiency without compromising data reliability.

Since `EntryProcessors` perform an implicit low level lock on the entries they are processing, the end user can place processing code in an `EntryProcessor` without having to worry about concurrency control. Note that this is **not** the same as the explicit `lock(key)` functionality provided by ConcurrentMap!

In a replicated cache or a partitioned cache running under Caching Edition, execution will happen locally on the initiating client. In partitioned caches running under Enterprise Edition or greater, the execution occurs on the node that is responsible for primary storage of the data.

`InvocableMap` provides three methods of starting EntryProcessors:

- Invoke an `EntryProcessor` on a specific key. Note that the key need not exist in the cache to invoke an EntryProcessor on it.
- Invoke an `EntryProcessor` on a collection of keys.
- Invoke an `EntryProcessor` on a `Filter`. In this case, the `Filter` will be executed against the cache entries. Each entry that matches the `Filter` criteria will have the `EntryProcessor` executed against it. For more information on Filters, see Chapter 6, "Query the Cache".

In partitioned caches running under Enterprise Edition or greater, `EntryProcessors` will be executed in parallel across the cluster (on the nodes that own the individual entries.) This provides a significant advantage over having a client lock all affected keys, pull all required data from the cache, process the data, place the data back in the cache, and unlock the keys. The processing occurs in parallel across multiple machines (as opposed to serially on one machine) and the network overhead of obtaining and releasing locks is eliminated.

Here is a sample of high level concurrency control. Code that will require network access is commented:

Example 2–4 Concurrency Control without Using EntryProcessors

```
final NamedCache cache = CacheFactory.getCache("dist-test");
```

```
final String key = "key";

cache.put(key, new Integer(1));


// begin processing

// *requires network access*
if (cache.lock(key, 0))
    {
    try
        {
        // *requires network access*
        Integer i = (Integer) cache.get(key);
        // *requires network access*
        cache.put(key, new Integer(i.intValue() + 1));
        }
    finally
        {
        // *requires network access*
        cache.unlock(key);
        }
    }

// end processing
```

The following is an equivalent technique using an Entry Processor. Again, network access is commented:

Example 2–5 Concurrency Control Using EntryProcessors

```
final NamedCache cache = CacheFactory.getCache("dist-test");
final String key = "key";

cache.put(key, new Integer(1));


// begin processing

// *requires network access*
cache.invoke(key, new MyCounterProcessor());

// end processing

...

public static class MyCounterProcessor
        extends AbstractProcessor
    {
    // this is executed on the node that owns the data,
    // no network access required
    public Object process(InvocableMap.Entry entry)
        {
        Integer i = (Integer) entry.getValue();
        entry.setValue(new Integer(i.intValue() + 1));
        return null;
        }
    }
```

`EntryProcessors` are individually executed atomically, however multiple `EntryProcessor` invocations by using `InvocableMap.invokeAll()` will not be executed as one atomic unit. As soon as an individual `EntryProcessor` has completed, any updates made to the cache will be immediately visible while the other `EntryProcessors` are executing. Furthermore, an uncaught exception in an `EntryProcessor` will not prevent the others from executing. Should the primary node for an entry fail while executing an `EntryProcessor`, the backup node will perform the execution instead. However if the node fails after the completion of an `EntryProcessor`, the `EntryProcessor` will not be invoked on the backup.

Note that in general, `EntryProcessors` should be short lived. Applications with longer running `EntryProcessors` should increase the size of the distributed service thread pool so that other operations performed by the distributed service are not blocked by the long running `EntryProcessor`. For more information on the distributed service thread pool, see "DistributedCache Service Parameters".

Coherence includes several `EntryProcessor` implementations for common use cases. Further details on these `EntryProcessors`, along with additional information on parallel data processing, can be found in *"Provide a Data Grid"*.

Data Source Integration

When using write-behind and write-through to a database in a Coherence cache, transactional behavior must be taken into account. With write-through enabled, the put will succeed if the item is successfully stored in the database. Otherwise, the exception that occurred in the CacheStore will be rethrown to the client. (Note: to enable this behavior, set `<rollback-cachestore-failures>` to `true`. See "read-write-backing-map-scheme" on page D-71 for more details.) This only applies when updating one cache item at a time; if two cache items are updated at a time then each `CacheStore` operation will be a distinct database transaction. This limitation will be addressed in a future release of Coherence.

Write-behind caching provides much higher throughput and performance. However, writes to the database are performed after the cache has been updated. Therefore care must be taken to ensure that writes to the database will not fail. Write-behind should only be used in applications where:

- data constraints will be managed by the application, not the database
- no other application will update the database

See *"Read-Through, Write-Through, Write-Behind Caching and Refresh-Ahead"* for more information on cache stores.

If multiple updates to cache entries must be persisted to the database in a transaction, it may be more suitable to implement a cache-aside pattern where the client is responsible for updating the database and the cache. Note that a CacheLoader may still be used to load cache misses from the data source.

3

Perform Continuous Query

While it is possible to obtain a point in time query result from a Coherence cache to, and it is possible to receive events that would change the result of that query, Coherence provides a feature that combines a query result with a continuous stream of related events to maintain an up-to-date query result in a real-time fashion. This capability is called *Continuous Query*, because it has the same effect as if the desired query had zero latency *and* the query were being executed several times every millisecond! For more information on point in time query results and events, see Chapter 6, "Query the Cache" and *"Deliver Events for Changes as they Occur"*.

Coherence implements the Continuous Query functionality by materializing the results of the query into a Continuous Query Cache, and then keeping that cache up-to-date in real-time using event listeners on the query. In other words, a Coherence Continuous Query is a cached query result that never gets out-of-date.

Uses of Continuous Query Caching

There are several different general use categories for Continuous Query Caching:

- It is an ideal building block for Complex Event Processing (CEP) systems and event correlation engines.
- It is ideal for situations in which an application repeats a particular query, and would benefit from always having instant access to the up-to-date result of that query.
- A Continuous Query Cache is analogous to a *materialized view*, and is useful for accessing and manipulating the results of a query using the standard NamedCache API, and receiving an ongoing stream of events related to that query.
- A Continuous Query Cache can be used in a manner similar to a Near Cache, because it maintains an up-to-date set of data locally *where it is being used*, for example on a particular server node or on a client desktop; note that a Near Cache is invalidation-based, but the Continuous Query Cache actually maintains its data in an up-to-date manner.

An example use case is a trading system desktop, in which a trader's open orders and all related information must be maintained in an up-to-date manner at all times. By combining the Coherence*Extend functionality with Continuous Query Caching, an application can support literally tens of thousands of concurrent users.

Note: Continuous Query Caches are useful in almost every type of application, including both client-based and server-based applications, because they provide the ability to very easily and efficiently maintain an up-to-date local copy of a specified sub-set of a much larger and potentially distributed cached data set.

The Coherence Continuous Query Cache

The Coherence implementation of Continuous Query is found in the `com.tangosol.net.cache.ContinuousQueryCache` class. This class, like all Coherence caches, implements the standard `NamedCache` interface, which includes the following capabilities:

- Cache access and manipulation using the `Map` interface: `NamedCache` extends the standard `Map` interface from the Java Collections Framework, which is the same interface implemented by the JDK's `HashMap` and `Hashtable` classes.
- Events for all objects modifications that occur within the cache: `NamedCache` extends the `ObservableMap` interface.
- Identity-based clusterwide locking of objects in the cache: `NamedCache` extends the `ConcurrentMap` interface.
- Querying the objects in the cache: `NamedCache` extends the QueryMap interface.
- Distributed Parallel Processing and Aggregation of objects in the cache: `NamedCache` extends the InvocableMap interface.

Since the `ContinuousQueryCache` implements the `NamedCache` interface, which is the same API provided by all Coherence caches, it is extremely simple to use, and it can be easily substituted for another cache when its functionality is called for.

Constructing a Continuous Query Cache

There are two items that define a Continuous Query Cache:

1. The underlying cache that it is based on;
2. A query of that underlying cache that produces the sub-set that the Continuous Query Cache will cache.

The underlying cache is any Coherence cache, including another Continuous Query Cache. A cache is usually obtained from a `CacheFactory`, which allows the developer to simply specify the name of the cache and have it automatically configured based on the application's cache configuration information; for example:

```
NamedCache cache = CacheFactory.getCache("orders");
```

See Appendix D, "Cache Configuration Elements" for more information on specifying cache configuration information.

The query is the same type of query that would be used to; for example:

Example 3–1 A Query for a Continuous Query Cache

```
Filter filter = new AndFilter(new EqualsFilter("getTrader", traderid),
                              new EqualsFilter("getStatus", Status.OPEN));
```

See Chapter 6, "Query the Cache" for more information on queries.

Normally, to query a cache, one of the methods from the `QueryMap` is used; for examples, to obtain a snap-shot of all open trades for this trader:

Example 3–2 Getting Data for the Continuous Query Cache

```
Set setOpenTrades = cache.entrySet(filter);
```

Similarly, the Continuous Query Cache is constructed from those same two pieces:

Example 3–3 Constructing the Continuous Query Cache

```
ContinuousQueryCache cacheOpenTrades = new ContinuousQueryCache(cache, filter);
```

Cleaning up the resources associated with a ContinuousQueryCache

A Continuous Query Cache places one or more event listeners on its underlying cache. If the Continuous Query Cache is used for the duration of the application, then the resources will be cleaned up when the node is shut down or otherwise stops. However, if the Continuous Query Cache is only used for a period, then when the application is done using it, the application must call the `release()` method on the `ContinuousQueryCache`.

Caching only keys, or caching both keys and values

When constructing a Continuous Query Cache, it is possible to specify that the cache should only keep track of the keys that result from the query, and obtain the values from the underlying cache only when they are asked for. This feature may be useful for creating a Continuous Query Cache that represents a very large query result set, or if the values are never or rarely requested. To specify that only the keys should be cached, use the constructor that allows the `CacheValues` property to be configured; for example:

Example 3–4 A Constructor that Allows the CacheValues Property

```
ContinuousQueryCache cacheOpenTrades = new ContinuousQueryCache(cache, filter,
false);
```

If necessary, the `CacheValues` property can also be modified after the cache has been instantiated; for example:

Example 3–5 Setting the CacheValues Property

```
cacheOpenTrades.setCacheValues(true);
```

CacheValues Property and Event Listeners

If the Continuous Query Cache has any standard (non-lite) event listeners, or if any of the event listeners are filtered, then the `CacheValues` property will automatically be set to true, because the Continuous Query Cache uses the locally cached values to filter events and to supply the old and new values for the events that it raises.

Listening to the ContinuousQueryCache

Since the Continuous Query Cache is itself observable, it is possible for the client to place one or more event listeners onto it. For example:

Example 3–6 Adding a Listener to a Continuous Query Cache

```
ContinuousQueryCache cacheOpenTrades = new ContinuousQueryCache(cache, filter);
cacheOpenTrades.addMapListener(listener);
```

Assuming some processing has to occur against every item that is already in the cache and every item added to the cache, there are two approaches. First, the processing could occur then a listener could be added to handle any later additions:

Example 3–7 Processing Continuous Query Cache Entries and Adding a Listener

```
ContinuousQueryCache cacheOpenTrades = new ContinuousQueryCache(cache, filter);
for (Iterator iter = cacheOpenTrades.entrySet().iterator(); iter.hasNext(); )
    {
    Map.Entry entry = (Map.Entry) iter.next();
    // .. process the cache entry
    }
cacheOpenTrades.addMapListener(listener);
```

However, **that code is incorrect** because it allows events that occur in the split second after the iteration and before the listener is added to be missed! The alternative is to add a listener first, so no events are missed, and then do the processing:

Example 3–8 Adding a Listener Before Processing Continuous Query Cache Entries

```
ContinuousQueryCache cacheOpenTrades = new ContinuousQueryCache(cache, filter);
cacheOpenTrades.addMapListener(listener);
for (Iterator iter = cacheOpenTrades.entrySet().iterator(); iter.hasNext(); )
    {
    Map.Entry entry = (Map.Entry) iter.next();
    // .. process the cache entry
    }
```

However, it is possible that the same entry will show up in both an event an in the `Iterator`, and the events can be asynchronous, so the sequence of operations cannot be guaranteed.

The solution is to provide the listener during construction, and it will receive one event for each item that is in the Continuous Query Cache, whether it was there to begin with (because it was in the query) or if it got added during or after the construction of the cache:

Example 3–9 Providing a Listener When Constructing the Continuous Query Cache

```
ContinuousQueryCache cacheOpenTrades = new ContinuousQueryCache(cache, filter,
listener);
```

Achieving a Stable Materialized View

The `ContinuousQueryCache` implementation faced the same challenge: How to assemble an exact point-in-time snapshot of an underlying cache *while receiving a stream of modification events from that same cache*. The solution has several parts. First, Coherence supports an option for synchronous events, which provides a set of ordering guarantees. See *"Deliver Events for Changes as they Occur"* for more information on this option.

Secondly, the `ContinuousQueryCache` has a two-phase implementation of its initial population that allows it to first query the underlying cache and then subsequently resolve all of the events that came in during the first phase. Since achieving these guarantees of data visibility without any missing or repeated events is fairly complex,

the `ContinuousQueryCache` allows a developer to pass a listener during construction, thus avoiding exposing these same complexities to the application developer.

Support for Synchronous and Asynchronous Listeners

By default, listeners to the `ContinuousQueryCache` will have their events delivered asynchronously. However, the `ContinuousQueryCache` does respect the option for synchronous events as provided by the `SynchronousListener` interface. See *"Deliver Events for Changes as they Occur"* for more information on this option.

Making the ContinuousQueryCache Read-Only

The `ContinuousQueryCache` can be made into a read-only cache; for example:

Example 3–10 Making the Continuous Query Cache Read-Only

```
cacheOpenTrades.setReadOnly(true);
```

A read-only `ContinuousQueryCache` will not allow objects to be added to, changed in, removed from or locked in the cache.

When a `ContinuousQueryCache` has been set to read-only, it cannot be changed back to read/write.

4

Managing Map Operations with Triggers

Map triggers supplement the standard capabilities of Oracle Coherence to provide a highly customized cache management system. For example, map triggers can be used to prevent invalid transactions, enforce complex security authorizations or complex business rules, provide transparent event logging and auditing, and gather statistics on data modifications. Other possible use for triggers include restricting operations against a cache to those issued during application re-deployment time.

For example, assume that you have code that is working with a `NamedCache`, and you want to change an entry's behavior or contents before the entry is inserted into the map. The addition of a map trigger will allow you to make this change, without having to modify all the exiting code.

Map triggers could also be used as part of an upgrade process. The addition of a map trigger could prompt inserts to be diverted from one cache into another.

A map trigger in the Oracle Coherence cache is somewhat similar to a trigger that might be applied to a database. It is a functional agent represented by the `MapTrigger` interface that will be run in response to a pending change (or removal) of the corresponding map entry. The pending change is represented by the `MapTrigger.Entry` interface. This interface inherits from the `InvocableMap.Entry` interface, so it provides methods to retrieve, update, and remove values in the underlying map.

The `MapTrigger` interface contains the process method that is used to validate, reject, or modify the pending change in the map. This method is called before an operation that intends to change the underlying map content is committed. An implementation of this method can evaluate the pending change by analyzing the original and the new value and produce any of the following results:

- override the requested change with a different value
- undo the pending change by resetting the original value
- remove the entry from the underlying map
- reject the pending change by throwing a RuntimeException
- do nothing, and allow the pending change to be committed

`MapTrigger` functionality is typically added as part of an application start-up process. It can be added programmatically as described in the `MapTrigger` API, or it can be configured using the `class-factory` mechanism in the `coherence-cache-config.xml` configuration file. In this case, a `MapTrigger` will be registered during the very first `CacheFactory.getCache(...)` call for the corresponding cache. Example 4–1 assumes that the `createMapTrigger` method would return a new `MapTriggerListener(new MyCustomTrigger());`:

Example 4–1 Creating a MapTriggerListener in the coherence-cache-config.xml File

```
<cache-config>
  ...
    <distributed-scheme>
      ...
      <listener>
        <class-scheme>
          <class-factory-name>package.MyFactory</class-factory-name>
          <method-name>createTriggerListener</method-name>
          <init-params>
            <init-param>
              <param-type>string</param-type>
              <param-value>{cache-name}</param-value>
            </init-param>
          </init-params>
        </class-scheme>
      </listener>
    <distributed-scheme>
  ...
</cache-config>
```

In addition to the `MapTrigger.Entry` and `MapTrigger` interfaces, Oracle Coherence provides the `FilterTrigger` and `MapTriggerListener` classes. The `FilterTrigger` is a generic `MapTrigger` implementation that will perform a predefined action if a pending change is rejected by the associated `Filter`. The `FilterTrigger` can either reject the pending operation, ignore the change and restore the entry's original value, or remove the entry itself from the underlying map.

The `MapTriggerListener` is a special purpose `MapListener` implementation that is used to register a `MapTrigger` with a corresponding `NamedCache`. In Example 4–2, `MapTriggerListener` is used to register the `PersonMapTrigger` with the `People` named cache.

Example 4–2 A MapTriggerListener Registering a MapTrigger with a Named Cache

```
NamedCache person = CacheFactory.getCache("People");
MapTrigger trigger = new PersonMapTrigger();
person.addMapListener(new MapTriggerListener(trigger));
```

These API reside in the `com.tangosol.util` package. For more information on these API, see the Javadoc pages for `MapTrigger`, `MapTrigger.Entry`, `FilterTrigger`, and `MapTriggerListener`.

A Map Trigger Example

The code in Example 4–3 illustrates a map trigger and how it can be called. In the `PersonMapTrigger` class in Example 4–3, the `process` method is implemented to modify an entry before it is placed in the map. In this case, the last name attribute of a `Person` object is converted to upper case characters. The object is then returned to the entry.

Example 4–3 A MapTrigger Class

```
...

public class PersonMapTrigger implements MapTrigger
    {
    public PersonMapTrigger()
```

```
        {
        }

    public void process(MapTrigger.Entry entry)
        {
        Person person  = (Person) entry.getValue();
        String sName   = person.getLastName();
        String sNameUC = sName.toUpperCase();

        if (!sNameUC.equals(sName))
           {
           person.setLastName(sNameUC);

           System.out.println("Changed last name of [" + sName + "] to [" +
person.getLastName() + "]");

           entry.setValue(person);
           }
        }

    // ---- hashCode() and equals() must be implemented

    public boolean equals(Object o)
        {
        return o != null && o.getClass() == this.getClass();
        }
    public int hashCode()
        {
        return getClass().getName().hashCode();
        }
    }
```

The `MapTrigger` in Example 4–4, calls the `PersonMapTrigger`. The new `MapTriggerListener` passes the `PersonMapTrigger` to the `People NamedCache`.

Example 4–4 Calling a MapTrigger and Passing it to a Named Cache

```
...

public class MyFactory
    {
    /**
    * Instantiate a MapTriggerListener for a given NamedCache
    */
    public static MapTriggerListener createTriggerListener(String sCacheName)
        {
        MapTrigger trigger;
        if ("People".equals(sCacheName))
            {
            trigger = new PersonMapTrigger();
            }
        else
            {
            throw IllegalArgumentException("Unknown cache name " + sCacheName);
            }

        System.out.println("Creating MapTrigger for cache " + sCacheName);

        return new MapTriggerListener(trigger);
```

```
    }

public static void main(String[] args)
    {
    NamedCache cache = CacheFactory.getCache("People");
    cache.addMapListener(createTriggerListener("People"));

    System.out.println("Installed MapTrigger into cache People");
    }
}
```

5

Data Affinity

Data affinity describes the concept of ensuring that a group of related cache entries is contained within a single cache partition. This ensures that all relevant data is managed on a single primary cache node (without compromising fault-tolerance).

Affinity may span multiple caches (if they are managed by the same cache service, which will generally be the case). For example, in a master-detail pattern such as an "Order-LineItem", the Order object may be co-located with the entire collection of LineItem objects that are associated with it.

The benefit is two-fold. First, only a single cache node is required to manage queries and transactions against a set of related items. Second, all concurrency operations can be managed locally, avoiding the need for clustered synchronization.

Several standard Coherence operations can benefit from affinity, including cache queries, `InvocableMap` operations and the `getAll`, `putAll`, and `removeAll` methods.

> **Note:** Data affinity is specified in terms of entry keys (not values). As a result, the association information must be present in the key class. Similarly, the association logic applies to the key class, not the value class.

Specifying Affinity

Affinity is specified in terms of a relationship to a partitioned key. In the `Order-LineItem` example above, the `Order` objects would be partitioned normally, and the `LineItem` objects would be associated with the appropriate `Order` object.

The association does not need to be directly tied to the actual parent key - it only must be a functional mapping of the parent key. It could be a single field of the parent key (even if it is non-unique), or an integer hash of the parent key. All that matters is that all child keys return the same associated key; it does not matter whether the associated key is an actual key (it is simply a "group id"). This fact may help minimize the size impact on the child key classes that don't already contain the parent key information (as it is derived data, the size of the data may be decided explicitly, and it also will not affect the behavior of the key). Note that making the association too general (having too many keys associated with the same "group id") can cause a "lumpy" distribution (if all child keys return the same association key regardless of what the parent key is, the child keys will all be assigned to a single partition, and will not be spread across the cluster).

There are two ways to ensure that a set of cache entries are co-located. Note that association is based on the cache key, not the value (otherwise updating a cache entry

could cause it to change partitions). Also, note that while the `Order` will be co-located with the child `LineItems`, Coherence does not currently support composite operations that span multiple caches (for example, updating the `Order` and the collection of `LineItems` within a single invocation request `com.tangosol.util.InvocableMap.EntryProcessor`).

Specifying Data Affinity with a KeyAssociation

For application-defined keys, the class (of the cache key) may implement `com.tangosol.net.cache.KeyAssociation` as follows:

Example 5–1 Creating a Key Association

```
import com.tangosol.net.cache.KeyAssociation;

public class LineItemId implements KeyAssociation
   {
   // {...}

   public Object getAssociatedKey()
       {
       return getOrderId();
       }

   // {...}
   }
```

Specifying Data Affinity with a KeyAssociator

Applications may also provide a custom `KeyAssociator`:

Example 5–2 A Custom KeyAssociator

```
import com.tangosol.net.partition.KeyAssociator;

public class LineItemAssociator implements KeyAssociator
    {
    public Object getAssociatedKey(Object oKey)
        {
        if (oKey instanceof LineItemId)
            {
            return ((LineItemId) oKey).getOrderId();
            }
        else if (oKey instanceof OrderId)
            {
            return oKey;
            }
        else
            {
            return null;
            }
        }

    public void init(PartitionedService service)
        {
        }
    }
```

The key associator may be configured for a `NamedCache` in the associated <distributed-scheme> element:

Example 5–3 Configuring a Key Associator

```
<distributed-scheme>
    <!-- ... -->
    <key-associator>
        <class-name>LineItemAssociator</class-name>
    </key-associator>
</distributed-scheme>
```

Example of Using Affinity

Example 5–4 illustrates how to use affinity to create a more efficient query (`NamedCache.entrySet(Filter)`) and cache access (`NamedCache.getAll(Collection)`).

Example 5–4 Using Affinity for a More Efficient Query

```
OrderId orderId = new OrderId(1234);

// this Filter will be applied to all LineItem objects to fetch those
// for which getOrderId() returns the specified order identifier
// "select * from LineItem where OrderId = :orderId"Filter filterEq = new
EqualsFilter("getOrderId", orderId);

// this Filter will direct the query to the cluster node that currently owns
// the Order object with the given identifier
Filter filterAsc = new KeyAssociatedFilter(filterEq, orderId);

// run the optimized query to get the ChildKey objects
Set setLineItemKeys = cacheLineItems.keySet(filterAsc);

// get all the Child objects immediately
Set setLineItems = cacheLineItems.getAll(setLineItemKeys);

// Or remove all immediately
cacheLineItems.keySet().removeAll(setLineItemKeys);
```

6

Query the Cache

Coherence can perform queries and indexes against currently cached data that meets a given set of criteria. Queries and indexes can be simple, employing filters packaged with Coherence, or they can be run against multi-value attributes such as collections and arrays.

Query Functionality

Coherence provides the ability to search for cache entries that meet a given set of criteria. The result set may be sorted if desired. Queries are evaluated with Read Committed isolation.

It should be noted that queries apply only to currently cached data (and will not use the `CacheLoader` interface to retrieve additional data that may satisfy the query). Thus, the dataset should be loaded entirely into cache before queries are performed. In cases where the dataset is too large to fit into available memory, it may be possible to restrict the cache contents along a specific dimension (for example, "date") and manually switch between cache queries and database queries based on the structure of the query. For maintainability, this is usually best implemented inside a cache-aware data access object (DAO).

Indexing requires the ability to extract attributes on each Partitioned cache node; in the case of dedicated `CacheServer` instances, this implies (usually) that application classes must be installed in the `CacheServer` classpath.

For Local and Replicated caches, queries are evaluated locally against unindexed data. For Partitioned caches, queries are performed in parallel across the cluster, using indexes if available. Coherence includes a Cost-Based Optimizer (CBO). Access to unindexed attributes requires object deserialization (though indexing on other attributes can reduce the number of objects that must be evaluated).

Simple Queries

Querying cache content is very simple:

Example 6–1 Querying the Cache with a Filter

```
Filter filter = new GreaterEqualsFilter("getAge", 18);

for (Iterator iter = cache.entrySet(filter).iterator(); iter.hasNext(); )
    {
    Map.Entry entry = (Map.Entry) iter.next();
    Integer key = (Integer) entry.getKey();
    Person person = (Person) entry.getValue();
    System.out.println("key=" + key + " person=" + person);
```

```
    }
```

Coherence provides a wide range of filters in the `com.tangosol.util.filter` package.

A `LimitFilter` may be used to limit the amount of data sent to the client, and also to provide "paging" for users. This is illustrated in Example 6–2:

Example 6–2 Using LimitFilter Class to Limit the Amount of Data Sent to the Client

```
int pageSize = 25;
Filter filter = new GreaterEqualsFilter("getAge", 18);

// get entries 1-25
Filter limitFilter = new LimitFilter(filter, pageSize);
Set entries = cache.entrySet(limitFilter);

// get entries 26-50
limitFilter.nextPage();
entries = cache.entrySet(limitFilter);
```

Any queryable attribute may be indexed with the `addIndex` method of the `QueryMap` class. This is illustrated in Example 6–3:

Example 6–3 Indexing a Queryable Attribute

```
// addIndex(ValueExtractor extractor, boolean fOrdered, Comparator comparator)
cache.addIndex(extractor, true, null);
```

The `fOrdered` argument specifies whether the index structure is sorted. Sorted indexes are useful for range queries, including "select all entries that fall between two dates" and "select all employees whose family name begins with 'S'". For "equality" queries, an unordered index may be used, which may have better efficiency in terms of space and time.

The comparator argument can be used to provide a custom `java.util.Comparator` for ordering the index.

This method is only intended as a hint to the cache implementation, and as such it may be ignored by the cache if indexes are not supported or if the desired index (or a similar index) already exists. It is expected that an application will call this method to suggest an index even if the index may already exist, just so that the application is certain that index has been suggested. For example in a distributed environment, each server will likely suggest the same set of indexes when it starts, and there is no downside to the application blindly requesting those indexes regardless of whether another server has already requested the same indexes.

Indexes are a feature of Coherence Enterprise Edition or higher. This method will have no effect when using Coherence Standard Edition.

Note that queries can be combined by Coherence if necessary, and also that Coherence includes a cost-based optimizer (CBO) to prioritize the usage of indexes. To take advantage of an index, queries must use extractors that are equal (`(Object.equals()`) to the one used in the query.

A list of applied indexes can be retrieved from the `StorageManagerMBean` by using JMX. For more information, see Chapter 22, "How to Manage Coherence Using JMX".

Querying Partitioned Caches

The Partitioned Cache implements this method using the Parallel Query feature, which is only available in Coherence Enterprise Edition or higher. When working with a Partitioned Cache in Coherence Standard Edition, this method will retrieve the data set to the client for processing.

Querying Near Caches

Although queries can be executed through a near cache, the query will not use the front portion of a near cache. If using a near cache with queries, the best approach is to use the sequence in Example 6–4:

Example 6–4 Querying the Near Cache

```
Set setKeys = cache.keySet(filter);
Map mapResult = cache.getAll(setKeys);
```

Query Concepts

This section goes into more detail on the design of the query interface, building up from the core components.

The concept of querying is based on the `ValueExtractor` interface. A value extractor is used to extract an attribute from a given object for querying (and similarly, indexing). Most developers will need only the `ReflectionExtractor` implementation of this interface. The ReflectionExtractor uses reflection to extract an attribute from a value object by referring to a method name, typically a "getter" method like `getName()`.

```
ValueExtractor extractor = new ReflectionExtractor("getName");
```

Any "void argument" method can be used, including `Object` methods like `toString()` (useful for prototyping/debugging). Indexes may be either traditional "field indexes" (indexing fields of objects) or "functional indexes" (indexing "virtual" object attributes). For example, if a class has field accessors `getFirstName` and `getLastName`, the class may define a function `getFullName` which concatenates those names, and this function may be indexed.

To query a cache that contains objects with `getName` attributes, a `Filter` must be used. A filter has a single method which determines whether a given object meets a criterion.

Example 6–5 Equality Filter

```
Filter filter = new EqualsFilter(extractor, "Bob Smith");
```

Note that the filters also have convenience constructors that accept a method name and internally construct a `ReflectionExtractor`:

Example 6–6 Filter that Constructs a ReflectionExtractor

```
Filter filter = new EqualsFilter("getName", "Bob Smith");
```

Example 6–7 illustrates a routine to select the entries of a cache that satisfy a particular filter:

Example 6–7 Selecting Cache Entries that Satisfy a Filter

```
for (Iterator iter = cache.entrySet(filter).iterator(); iter.hasNext(); )
    {
    Map.Entry entry = (Map.Entry)iter.next();
    Integer key = (Integer)entry.getKey();
    Person person = (Person)entry.getValue();
    System.out.println("key=" + key + " person=" + person);
    }
```

Example 6–8 illustrates using a filter to select and sort cache entries:

Example 6–8 Selecting and Sorting Cache Entries that Satisfy a Filter

```
// entrySet(Filter filter, Comparator comparator)
Iterator iter = cache.entrySet(filter, null).iterator();
```

The additional null argument specifies that the result set should be sorted using the "natural ordering" of `Comparable` objects within the cache. The client may explicitly specify the ordering of the result set by providing an implementation of `Comparator`. Note that sorting places significant restrictions on the optimizations that Coherence can apply, as sorting requires that the entire result set be available before sorting.

Example 6–9 illustrates using the `keySet` form of the queries, combined with `getAll()`. This technique may provide more control over memory usage:

Example 6–9 Using a keySet Query Format

```
// keySet(Filter filter)
Set setKeys = cache.keySet(filter);
Set setPageKeys = new HashSet();
int PAGE_SIZE = 100;
for (Iterator iter = setKeys.iterator(); iter.hasNext();)
    {
    setPageKeys.add(iter.next());
    if (setKeyPage.size() == PAGE_SIZE || !iter.hasNext())
        {
        // get a block of values
        Map mapResult = cache.getAll(setPageKeys);

        // process the block
        // ...

        setPageKeys.clear();
        }
    }
```

Queries Involving Multi-Value Attributes

Coherence supports indexing and querying of multi-value attributes including collections and arrays. When an object is indexed, Coherence will verify if it is a multi-value type, and will then index it as a collection rather than a singleton. The `ContainsAllFilter`, `ContainsAnyFilter` and `ContainsFilter` are used to query against these collections.

Example 6–10 Querying on Multi-Value Attributes

```
Set searchTerms = new HashSet();
searchTerms.add("java");
searchTerms.add("clustering");
```

```
searchTerms.add("books");

// The cache contains instances of a class "Document" which has a method
// "getWords" which returns a Collection<String> containing the set of
// words that appear in the document.
Filter filter = new ContainsAllFilter("getWords", searchTerms);

Set entrySet = cache.entrySet(filter);

// iterate through the search results
// ...
```

ChainedExtractor

The `ChainedExtractor` implementation allows chained invocation of zero-argument (accessor) methods. In Example 6–11, the extractor will first use reflection to call `getName()` on each cached `Person` object, and then use reflection to call `length()` on the returned `String`.

Example 6–11 Chaining Invocation Methods

```
ValueExtractor extractor = new ChainedExtractor("getName.length");
```

This extractor could be passed into a query, allowing queries (for example) to select all people with names not exceeding 10 letters. Method invocations may be chained indefinitely, for example `getName.trim.length`.

7

Security Framework

This chapter describes the following security features:

- Transport Layer Security
- Access Controller
- Proof of Identity
- Proof of Trustworthiness
- Default Access Controller implementation
- Working in applications with installed security manager

Transport Layer Security

For information on transport layer security, see "Encryption Filters" on page 8-1.

Access Controller

Security Framework in Coherence is based on a concept of Clustered Access Controller, which can be turned on (activated) by a configurable parameter or command line attribute.

The Access Controller manages access to the "clustered resources", such as clustered services and caches and controls operations that include (but not limited to) the following:

- creating a new clustered cache or service;
- joining an existing clustered cache or service;
- destroying an existing clustered cache.

The Access Controller serves three purposes:

- grant or deny access to a protected clustered resource based on the caller's permissions
- encrypt outgoing communications based on the caller's private credentials
- decrypt incoming communications based on the caller's public credentials

Coherence uses a local *LoginModule* (see JAAS Reference Guide for details) to authenticate the caller and an Access Controller on one or more cluster nodes to verify the caller's access rights.

The Access Controller is a pluggable component that could be declared in the Coherence deployment descriptior, `tangosol-coherence.xml`. The specified class should implement the com.tangosol.net.security.AccessController interface.

Coherence provides a default Access Controller implementation that is based on the Key Management infrastructure that is shipped as a standard part of Sun's JDK.

Each clustered service in Coherence maintains a concept of a "senior" service member (cluster node), which serves as a controlling agent for a particular service. While the senior member does not have to consult anyone when accessing a clustered resource, any junior node willing to join that service has to request and receive a confirmation from the senior member, which in turn notifies all other cluster nodes about the joining node.

Since Coherence is a system providing distributed data management and computing, the security subsystem is designed to operate in a partially hostile environment. We assume that when there is data shared between two cluster nodes either node could be a malicious one - lacking sufficient credentials to join a clustered service or obtain access to a clustered resource.

Let's call a cluster node that may try to gain unauthorized access to clustered resources by using nonstandard means as a "malicious" node. The means of such an access could vary. They could range from attempts to get protected or private class data using reflection, replacing classes in the distribution (coherence.jar or other application binaries), modifying classes on-the-fly using custom *ClassLoader(s)* and so on. Alternatively, a cluster node that never attempts to gain unauthorized access to clustered resources by using nonstandard means will be called a "trusted" node. It's important to note that even a trusted node may attempt to gain access to resources without having sufficient rights, but it does so in a standard way by using the exposed standard API.

File system mechanisms (the same that is used to protect the integrity of the Java runtime libraries) and standard Java security policy could be used to resolve an issue of guarantying the trustworthiness of a given single node. In a case of inter-node communications there are two dangers that we have to consider:

- A malicious node surpasses the local access check and attempts to join a clustered service or gain access to a clustered resource controlled by a trusted node;
- A malicious node creates a clustered service or clustered resource becoming its controller.

To prevent either of these two scenarios from occurring Coherence uses two-ways encryption algorithm: all client requests must be accompanied by the proof of identity and all service responses must be accompanied by the proof of trustworthiness.

Proof of Identity

In a case of an active Access Controller the client code could use the following construct to authenticate the caller and perform necessary actions:

```
import com.tangosol.net.security.Security;
import java.security.PrivilegedAction;
import javax.security.auth.Subject;

...

Subject subject = Security.login(sName, acPassword);
PrivilegedAction action = new PrivilegedAction()
    {
```

```
    public Object run()
        {
        // all processing here is taking place with access
        // rights assigned to the corresponding Subject
        ...
        }
    };
Security.runAs(subject, action);
```

During the "login" call Coherence uses JAAS that runs on the caller's node to authenticate the caller. In a case of successful authentication, it uses the local Access Controller to:

1. Determine whether the local caller has sufficient rights to access the protected clustered resource (local access check);
2. Encrypt the outgoing communications regarding the access to the resource with the caller's private credentials retrieved during the authentication phase;
3. Decrypt the result of the remote check using the requester's public credentials;
4. In the case that access is granted verify whether the responder had sufficient rights to do so.

Step 2 (above) serves a role of the proof of identity for the responder preventing a malicious node pretending to pass the local access check phase.

There are two alternative ways to provide the client authentication information. First, a reference to a *CallbackHandler* could be passed instead of the user name and password. Second, a previously authenticated Subject could be used, which could become handy when Coherence is used by a Java EE application that could retrieve an authenticated Subject from the application container.

If a caller's request comes without any authentication context, Coherence will instantiate and call a *CallbackHandler* implementation declared in the Coherence operational descriptor to retrieve the appropriate credentials. However that "lazy" approach is much less efficient, since without externally defined call scope, every access to a protected clustered resource will force repetitive authentication calls.

Proof of Trustworthiness

Every clustered resource in Coherence is created by an explicit API call. A senior service member retains the private credentials that are presented during that call as a proof of trustworthiness. When the senior service member receives an access request to a protected clustered resource, it use the local Access Controller to:

1. Decrypt the incoming communication using the remote caller's public credentials;
2. Determine whether the remote caller has sufficient rights to access the protected clustered resource (remote access check);
3. Encrypt the response of access check using the private credentials of the service.

Since the requester will accept the response as valid only after decrypting it, step 3) in this cycle serves a role of the proof of trustworthiness for the requester preventing a malicious node pretending to be a valid service senior.

Default Access Controller implementation

Coherence ships with an Access Controller implementation that uses a standard Java KeyStore. The implementation class is com.tangosol.net.security.DefaultController and

the corresponding part of the Coherence operational descriptor used to configure the default implementation is:

```
<security-config>
  <enabled system-property="tangosol.coherence.security">true</enabled>
  <login-module-name>Coherence</login-module-name>
  <access-controller>
    <class-name>com.tangosol.net.security.DefaultContoller</class-name>
    <init-params>
      <init-param id="1">
        <param-type>java.io.File</param-type>
        <param-value>./keystore.jks</param-value>
      </init-param>
      <init-param id="2">
        <param-type>java.io.File</param-type>
        <param-value>./permissions.xml</param-value>
      </init-param>
    </init-params>
  </access-controller>
  <callback-handler>
    <class-name/>
  </callback-handler>
</security-config>
```

The **login-module-name** element serves as the application name in a login configuration file (see JAAS Reference Guide1 for complete details). Coherence is shipped with a Java keystore (JKS) based login module that is contained in the coherence-login.jar, which depends only on standard Java runtime classes and could be placed in the JRE's lib/ext (standard extension) directory. The corresponding login module declaration would look like:

```
// LoginModule Configuration for Oracle Coherence(TM)
Coherence {
    com.tangosol.security.KeystoreLogin required
      keyStorePath="${user.dir}${/}keystore.jks";
};
```

The **access-controller** element defines the *AccessController* implementation that takes two parameters to instantiate.

- The first parameter is a path to the same keystore that will be used by both controller and login module.
- The second parameter is a path to the access permission file (see discussion below).

The **callback-handler** is an optional element that defines a custom implementation of the *javax.security.auth.callback.CallbackHandler* interface that would be instantiated and used by Coherence to authenticate the client when all other means are exhausted.

Two more steps have to be performed, To make the default Access Controller implementation usable in your application, you must perform two additional steps:

1. Create a keystore with necessary principals.
2. Create the `permissions` file that would declare the access right for the corresponding principals.

Consider the following example that creates three principals: `admin` to be used by the Java Security framework; `manager` and `worker` to be used by Coherence:

```
keytool -genkey -v -keystore ./keystore.jks -storepass password -alias admin
-keypass password -dname CN=Administrator,O=MyCompany,L=MyCity,ST=MyState
```

```
keytool -genkey -v -keystore ./keystore.jks -storepass password -alias manager
-keypass password -dname CN=Manager,OU=MyUnit

keytool -genkey -v -keystore ./keystore.jks -storepass password -alias worker
-keypass password -dname CN=Worker,OU=MyUnit
```

Consider the following example that assigns all rights to the `Manager` principal, only `join` rights to the `Worker` principal for caches that have names prefixed by `common` and all rights to the `Worker` principal for the invocation service named `invocation`:

```
<?xml version='1.0'?>
<permissions>
  <grant>
    <principal>
      <class>javax.security.auth.x500.X500Principal</class>
      <name>CN=Manager,OU=MyUnit</name>
    </principal>

    <permission>
      <target>*</target>
      <action>all</action>
    </permission>
  </grant>

  <grant>
    <principal>
      <class>javax.security.auth.x500.X500Principal</class>
      <name>CN=Worker,OU=MyUnit</name>
    </principal>

    <permission>
      <target>cache=common*</target>
      <action>join</action>
    </permission>
    <permission>
      <target>service=invocation</target>
      <action>all</action>
    </permission>
  </grant>
</permissions>
```

Working in applications with installed security manager

1. The policy file format is fully described in Java SE Security Guide. Example:

   ```
   grant codeBase "file:${coherence.home}/lib/coherence.jar"
       {
               permission java.security.AllPermission;
       };
   ```

   The minimum set of privileges required for Coherence to function are specified in the security.policy file which is included as part of the Coherence installation. This file can be found in coherence/lib/security/security.policy.

2. The binaries could be signed using the JDK jarsigner tool, for example:

   ```
   jarsigner -keystore ./keystore.jks -storepass password coherence.jar admin
   ```

   and then additionally protected in the policy file:

```
grant SignedBy "admin" codeBase "file:${coherence.home}/lib/coherence.jar"
    {
            permission java.security.AllPermission;
    };
```

3. All relevant files such as policy format, coherence binaries, and permissions should be protected by operating system mechanisms to prevent malicious modifications.

8

Network Filters

A filter is a mechanism for plugging into the low-level TCMP stream protocol. Every message that is sent across the network by Coherence is streamed through this protocol. Coherence supports custom filters. By writing a filter, the contents of the network traffic can be modified. The most common examples of modification are encryption and compression.

Compression Filters

The compression filter is based on the `java.util.zip` package and compresses message contents thus reducing the network load. This is useful when there is ample CPU available but insufficient network bandwidth. See "Configuring Filters" on page 8-4 for information on enabling this filter.

Encryption Filters

Coherence ships with two JCA based encryption filters which can be used to protect the clustered communications for privacy and authenticity.

Symmetric Encryption Filter

This filter uses symmetric encryption to protect cluster communications. The encryption key is generated from a shared password known to all cluster members. This filter is suitable for small deployments or where the maintenance and protection of a shared password is feasible.

To enable this filter, specify which services will have their traffic encrypted by using this filter, or to enable it for all cluster traffic you may simply specify it as a filter for the <`outgoing-message-handler`> element.

Example 8–1 Enabling a Filter for all Network Traffic

```
<outgoing-message-handler>
  <use-filters>
    <filter-name>symmetric-encryption</filter-name>
  </use-filters>
</outgoing-message-handler>
```

The shared password may either be specified in the <`filters`> section of the operational configuration file, or by using the `tangosol.coherence.security.password` system property. See "Symmetric Encryption Filter Parameters" on page 8-2 for additional configuration options.

Symmetric Encryption Filter Parameters

The symmetric encryption filter supports the parameters listed in Table 8–1. See the `com.tangosol.net.security.PasswordBasedEncryptionFilter` Javadoc for additional configuration details.

Table 8–1 Symmetric Encryption Filter Parameters

Parameter Name	Value Description
algorithm	Specifies the mechanism to use in deriving a secret key from the above material. Default value is `PBEWithMD5AndDES`.
iterations	Specifies the iteration count to use in deriving the key. Default value is 32.
password	Specifies the raw material used to generate the secret key. Preconfigured is `tangosol.coherence.security.password`. See "Preconfigured Override Values" on page L-2
salt	Specifies the salt to use in deriving the key. Default value is `nosecret`.

PKCS Encryption Filter

This filter uses public key cryptography (asymmetric encryption) to protect the cluster join protocol, and then switches over to much faster symmetric encryption for service level data transfers. Unlike the symmetric encryption filter, there is no persisted shared secret. The symmetric encryption key is randomly generated by the cluster's senior member, and is securely transfer to authenticated cluster members as part of the cluster join protocol. This encryption filter is suitable for deployments where maintenance of a shared secret is not feasible.

> **Note:** This filter requires the JVM be configured with a JCA public key cryptography provider implementation such as Bouncy Castle, which supports asymmetric block ciphers. See the JCA documentation for details on installing and configuring JCA providers.

In the default setup each cluster node must be configured with a Java Keystore from which it may retrieve its identity Certificate and associated private key, and a set of trusted Certificates for other cluster members. You can construct this keystore as follows:

Create a Java Keystore and the local cluster member's password protected certificate and private key.

```
keytool -genkey -alias local -keypass secret -keyalg rsa -storepass secret
-keystore ./keystore.jks
```

Export this public certificate for inclusion in all cluster members keystores.

```
keytool -export -alias local -keypass secret -storepass secret -keystore
./keystore.jks -rfc -file local.cert
```

Import the Certificates of other trusted cluster members. Each certificate must be stored under a unique but otherwise unimportant alias.

```
keytool -import -alias remote_1 -storepass secret -keystore ./keystore.jks -file
local_1.cert
keytool -import -alias remote_2 -storepass secret -keystore ./keystore.jks -file
```

```
local_2.cert
keytool -import -alias remote_3 -storepass secret -keystore ./keystore.jks -file
local_3.cert
```

At this point you will have one keystore per cluster node, each containing a single private key plus a full set of trusted public certificates. If new nodes are to be added to the cluster the keystores of all existing nodes must be updated with the new node's certificate.

> **Note:** You may also choose to supply custom key and trust management logic to eliminate the need for a full keystore per node. See the implementation's documentation for details on customization.

Then configure the cluster to encrypt all traffic using this filter by specifying it in the <outgoing-message-handler>.

```
<outgoing-message-handler>
  <use-filters>
    <filter-name>pkcs-encryption</filter-name>
  </use-filters>
</outgoing-message-handler>
```

The keystore and alias password can be specified either in the <filters> section of the operational configuration file, or by using the `tangosol.coherence.security.password` system property. See "PKCS Encryption Filter Parameters" for additional configuration options.

Note unlike the Symmetric Encryption Filter, this filter is not currently supported by Coherence*Extend, or on a service by service level.

PKCS Encryption Filter Parameters

The PKCS encryption filter supports the following parameters, see "Encryption Filters" on page 8-1 section for examples, or the `com.tangosol.net.security.ClusterEncryptionFilter` Javadoc for additional configuration details.

Table 8–2 PKCS Encryption Filter Parameters

Parameter Name	Description
asymmetricFilterClassName	Specifies the asymmetric filter implementation. Default value is `com.tangosol.net.security.AsymmetricEncryptionFilter`.
keyAlias	Specifies the alias to use in reading the key from the keystore.
keyPassword	Specifies the password to use in reading the key. Preconfigured value is `tangosol.coherence.security.password`. See "Preconfigured Override Values" on page L-2.
store	Specifies the path to the `KeyStore` Default value is `.keystore`.
sharedKeySize	Specifies the size of shared key. Default value is `112`.
sharedKeyType	Specifies the type of shared key. Default value is `DESede`.
storePassword	Specifies the password to use to access the store If unspecified value of `keyPassword` parameter will be used.
storeType	Specifies the type of `KeyStore`. Default value is `JKS`.

Table 8–2 (Cont.) PKCS Encryption Filter Parameters

Parameter Name	Description
transformation	Specifies the transformation to use. Default value is `RSA/NONE/PKCS1Padding`.

Configuring Filters

There are two steps to configuring a filter.

1. Declare the filter in the <filters> XML element of the `tangosol-coherence.xml` file:

Example 8–2 Declaring a Filter in the tangosol-coherence.xml File

```
<filter>
  <filter-name>gzip</filter-name>
  <filter-class>com.tangosol.net.CompressionFilter</filter-class>
  <init-params>
    <init-param>
      <param-name>strategy</param-name>
      <param-value>gzip</param-value>
    </init-param>
  </init-params>
</filter>
```

   For more information on the structure of the <filters> XML element of the `tangosol-coherence.xml` file, see the documentation in the `coherence.dtd` file, which is also located inside `coherence.jar`.

2. The second step is to attach the filter to one or more specific services, or to make the filter global (for all services). To specify the filter for a specific service, for example the `ReplicatedCache` service, add a `<filter-name>` element to the `<use-filters>` element of the service declaration in the `tangosol-coherence.xml` file:

Example 8–3 Attaching the Filter to a Service

```
<service>
  <service-type>ReplicatedCache</service-type>
  <service-component>ReplicatedCache</service-component>
  <use-filters>
    <filter-name>gzip</filter-name>
  </use-filters>
  <init-params>
    ...
  </init-params>
</service>
```

To add the filter to all services, do the same under the <outgoing-message-handler> XML element instead of under a `<service>` XML element:

Example 8–4 Adding the Filter to All Services

```
<outgoing-message-handler>
  <use-daemon>false</use-daemon>
  <use-filters>
    <filter-name>gzip</filter-name>
  </use-filters>
</outgoing-message-handler>
```

Note: Filters should be used in an all-or-nothing manner: If one cluster member is using a filter and other is not, the messaging protocol will fail. You should stop the entire cluster before configuring filters.

Creating a Custom Filter

To create a new filter, create a Java class that implements the `com.tangosol.io.WrapperStreamFactory` interface and optionally implements the `com.tangosol.run.xml.XmlConfigurable` interface. The `WrapperStreamFactory` interface provides the stream to be wrapped ("filtered") on input (received message) or output (sending message) and expects a stream back that wraps the original stream. These methods are called for each incoming and outgoing message.

If the filter class implements the `XmlConfigurable` interface, then Coherence will configure the filter after instantiating it. Example 8–5 illustrates a filter declaration in the `tangosol-coherence.xml` file. If the filter is associated with a service type, every time a new service is started of that type, Coherence will instantiate the `CompressionFilter` class and will hold it with the service until the service stops. If the filter is associated with all outgoing messages, Coherence will instantiate the filter on startup and will hold it until the cluster stops.

Example 8–5 Configuration for a Custom Filter

```
<filter>
  <filter-name>my-gzip-filter</filter-name>
  <filter-class>com.tangosol.net.CompressionFilter</filter-class>
  <init-params>
    <init-param>
      <param-name>strategy</param-name>
      <param-value>gzip</param-value>
    </init-param>
    <init-param>
      <param-name>buffer-length</param-name>
      <param-value>1024</param-value>
    </init-param>
  </init-params>
</filter>
```

After instantiating the filter, Coherence will call the `setConfig` method (if the filter implements `XmlConfigurable`) with the following XML element:

Example 8–6 Configuring a setConfig Call for a Filter

```
<config>
  <strategy>gzip</strategy>
  <buffer-length>1024</buffer-length>
</config>
```

9

Priority Tasks

Coherence Priority Tasks provide applications that have critical response time requirements better control of the execution of processes within Coherence. Execution and request timeouts can be configured to limit wait time for long running threads. In addition, a custom task API allows applications to control queue processing. Note that these features should be used with extreme caution because they can dramatically effect performance and throughput of the data grid.

Priority Tasks — Timeouts

Care should be taken when configuring Coherence Task Execution timeouts, especially for Coherence applications that pre-date this feature and thus do not handle timeout exceptions. If a write-through in a `CacheStore` is blocked (for example, if a database query is hung) and exceeds the configured timeout value, the Coherence Task Manager will attempt to interrupt the execution of the thread and an exception will be thrown. In a similar fashion, queries or aggregations that exceed configured timeouts will be interrupted and an exception will be thrown. Applications that use this feature should make sure that they handle these exceptions correctly to ensure system integrity. Since this configuration is performed on a service by service basis, changing these settings on existing caches/services not designed with this feature in mind should be done with great care.

Configuring Execution Timeouts

When configuring Execution Timeouts these values need to be considered: request-timeout, task-timeout, and the task-hung-threshold (see "Execution Timeout Parameters"). The request-timeout is the amount of time the client will wait a request to return. The task-timeout is the amount of time that the server will allow the thread to execute before interrupting execution. The task-hung-threshold is the amount of time that a thread can execute before the server reports the thread as "hung." "Hung" threads are for reporting purposes only. These timeout settings are in milliseconds and are configured in the `coherence-cache-config.xml` or by using command line parameters.

Execution Timeout Parameters

Table 9–1 describes the execution timeout parameters.

Table 9–1 Execution Timeout Parameters

Parameter Name	Description
`<task-hung-threshold>`	Specifies the amount of time in milliseconds that a task can execute before it is considered "hung". Note: A posted task that has not yet started is never considered as hung. This attribute is applied only if the Thread pool is used (the `thread-count` value is positive).
`<task-timeout>`	Specifies the default timeout value for tasks that can be timed-out (for example, implement the `PriorityTask` interface), but don't explicitly specify the task execution timeout value. The task execution time is measured on the server side and does not include the time spent waiting in a service backlog queue before being started. This attribute is applied only if the thread pool is used (the `thread-count` value is positive)
`<request-timeout>`	Specifies the default timeout value for requests that can time-out (for example, implement the `PriorityTask` interface), but don't explicitly specify the request timeout value. The request time is measured on the client side as the time elapsed from the moment a request is sent for execution to the corresponding server node(s) and includes the following: **1.** The time it takes to deliver the request to an executing node (server). **2.** The interval between the time the task is received and placed into a service queue until the execution starts. **3.** The task execution time. **4.** The time it takes to deliver a result back to the client.

To set the distributed cache thread count to 7 with a task time out of 5000 milliseconds and a task hung threshold of 10000 milliseconds, the following would need to be added to the `coherence-cache-config.xml` for the node.

Example 9–1 Sample Task Time and Task Hung Configuration

```
<caching-schemes>
    <distributed-scheme>
      <scheme-name>example-distributed</scheme-name>
      <service-name>DistributedCache</service-name>
      <thread-count>7</thread-count>
      <task-timeout>5000ms</task-timeout>
      <task-hung-threshold>10000ms</task-hung-threshold>
    </distributed-scheme>
</caching-schemes>
```

Setting the client request timeout to 15 milliseconds

Example 9–2 Sample Client Request Timeout Configuration

```
<distributed-scheme>
      <scheme-name>example-distributed</scheme-name>
      <service-name>DistributedCache</service-name>
      <request-timeout>15000ms</request-timeout>
    </distributed-scheme>
```

Note: The request-timeout should always be longer than the `thread-hung-threshold` or the `task-timeout`.

Command Line Options

The command line options can be used to set the service type default (such as distributed cache, invocation, proxy, and so on) for the node. Table 9–2 describes the options.

Table 9–2 Command Line Options for Setting Service Type

Option	Description
`tangosol.coherence.replicated.request.timeout`	The default client request timeout for the Replicated cache service
`tangosol.coherence.optimistic.request.timeout`	The default client request timeout for the Optimistic cache service
`tangosol.coherence.distributed.request.timeout`	The default client request timeout for distributed cache services
`tangosol.coherence.distributed.task.timeout`	The default server execution timeout for distributed cache services
`tangosol.coherence.distributed.task.hung`	The default time before a thread is reported as hung by distributed cache services
`tangosol.coherence.invocation.request.timeout`	The default client request timeout for invocation services
`tangosol.coherence.invocation.task.hung`	The default time before a thread is reported as hung by invocation services
`tangosol.coherence.invocation.task.timeout`	The default server execution timeout invocation services
`tangosol.coherence.proxy.request.timeout`	The default client request timeout for proxy services
`tangosol.coherence.proxy.task.timeout`	The default server execution timeout proxy services
`tangosol.coherence.proxy.task.hung`	The default time before a thread is reported as hung by proxy services

Priority Task Execution — Custom Objects

The `PriorityTask` interface enables you to control the ordering in which a service schedules tasks for execution using a thread pool and hold their execution time to a specified limit. Instances of `PriorityTask` typically also implement either the `Invocable` or `Runnable` interface. Priority Task Execution is only relevant when a task back log exists.

The API defines the following ways to schedule tasks for execution

- `SCHEDULE_STANDARD`—a task will be scheduled for execution in a natural (based on the request arrival time) order
- `SCHEDULE_FIRST`—a task will be scheduled in front of any equal or lower scheduling priority tasks and executed as soon as any of worker threads become available
- `SCHEDULE_IMMEDIATE`—a task will be immediately executed by any idle worker thread; if all of them are active, a new thread will be created to execute this task

APIs for Creating Priority Task Objects

Coherence provides the following classes to help create priority task objects:

- `PriorityProcessor` can be extended to create a custom entry processor.
- `PriorityFilter` can be extended to create a custom priority filter.
- `PriorityAggregator` can be extended to create a custom aggregation.
- `PriorityTask` can be extended to create an priority invocation class.

After extending each of these classes the developer will need to implement several methods. The return values for `getRequestTimeoutMillis`, `getExecutionTimeoutMillis`, and `getSchedulingPriority` should be stored on a class-by-class basis in your application configuration parameters. These methods are described in Table 9–3.

Table 9–3 Methods to Support Task Timeout

Method	Description
`public long getRequestTimeoutMillis()`	Obtains the maximum amount of time a calling thread is willing to wait for a result of the request execution. The request time is measured on the client side as the time elapsed from the moment a request is sent for execution to the corresponding server node(s) and includes: the time it takes to deliver the request to the executing node(s); the interval between the time the task is received and placed into a service queue until the execution starts; the task execution time; the time it takes to deliver a result back to the client. The value of `TIMEOUT_DEFAULT` indicates a default timeout value configured for the corresponding service; the value of `TIMEOUT_NONE` indicates that the client thread is willing to wait indefinitely until the task execution completes or is canceled by the service due to a task execution timeout specified by the `getExecutionTimeoutMillis()` value.
`public long getExecutionTimeoutMillis()`	Obtains the maximum amount of time this task is allowed to run before the corresponding service will attempt to stop it. The value of `TIMEOUT_DEFAULT` indicates a default timeout value configured for the corresponding service; the value of `TIMEOUT_NONE` indicates that this task can execute indefinitely. If, by the time the specified amount of time passed, the task has not finished, the service will attempt to stop the execution by using the `Thread.interrupt()` method. In the case that interrupting the thread does not result in the task's termination, the `runCanceled` method will be called.
`public int getSchedulingPriority()`	Obtains this task's scheduling priority. Valid values are `SCHEDULE_STANDARD`, `SCHEDULE_FIRST`, `SCHEDULE_IMMEDIATE`
`public void runCanceled(boolean fAbandoned)`	This method will be called if and only if all attempts to interrupt this task were unsuccessful in stopping the execution or if the execution was canceled before it had a chance to run at all. Since this method is usually called on a service thread, implementors must exercise extreme caution since any delay introduced by the implementation will cause a delay of the corresponding service.

Errors Thrown by Task Timeouts

When a task timeout occurs the node will get a `RequestTimeoutException`. Example 9–3 illustrates an exception that may be thrown.

Example 9–3 Exception Thrown by a TaskTimeout

```
com.tangosol.net.RequestTimeoutException: Request timed out after 4015 millis
        at com.tangosol.coherence.component.util.daemon.queueProcessor.Service.
checkRequestTimeout(Service.CDB:8)
        at com.tangosol.coherence.component.util.daemon.queueProcessor.Service.
poll(Service.CDB:52)
        at com.tangosol.coherence.component.util.daemon.queueProcessor.Service.
poll(Service.CDB:18)
        at com.tangosol.coherence.component.util.daemon.queueProcessor.service.
InvocationService.query(InvocationService.CDB:17)
        at com.tangosol.coherence.component.util.safeService.
SafeInvocationService.query(SafeInvocationService.CDB:1)
```

10

Integrate CacheFactory with Spring

The CacheFactory static factory methods allow you to access all Coherence caches and services. These methods, (such as getCache), delegate to a ConfigurableCacheFactory interface, which is pluggable by using CacheFactory.setConfigurableCacheFactory or the operational override file (tangosol-coherence-override.xml).

In the Coherence cache configuration file, (coherence-cache-config.xml) hooks are provided for end users to provide their own implementations of Coherence interfaces, such as CacheStore and MapListener. This is configured by using the class-scheme element. Coherence can instantiate these classes in two ways: it can create a new instance by using the new operator, or it can invoke a user-provided factory method.

For some applications, it may be useful for Coherence to retrieve objects configured in a class-scheme from a Spring BeanFactory instead of creating its own instance. This is especially true for cache servers configured with CacheStore objects running in a standalone JVM, as these CacheStore objects typically need to be configured with data sources, connection pools, and so on. Spring is well known for its ability to provide easy configuration of data sources for plain Java objects.

SpringAwareCacheFactory is a custom ConfigurableCacheFactory which has the ability to delegate class-scheme bean instantiations to a Spring BeanFactory. It has two modes of operation:

- It can instantiate its own ApplicationContext with a provided configuration file. This is useful for cache servers that require beans from a Spring container.
- A BeanFactory can be provided to it at runtime. This is useful for Coherence applications running in a container that already has an existing BeanFactory.

To configure Coherence to use the SpringAwareCacheFactory, the following XML code should be placed in the operational override file (tangosol-coherence-override.xml):

Example 10–1 Configuring the Cache to use SpringAwareCacheFactory in the Override File

```
<configurable-cache-factory-config>
  <class-name system-property="tangosol.coherence.cachefactory">
    com.tangosol.coherence.spring.SpringAwareCacheFactory
  </class-name>
  <init-params>
    <init-param>
      <param-type>java.lang.String</param-type>
      <param-value system-property="tangosol.coherence.cacheconfig">
        coherence-cache-config.xml
```

```
      </param-value>
    </init-param>
    <init-param id="1">
      <param-type>java.lang.String</param-type>
      <param-value system-property="tangosol.coherence.springconfig">
        application-context.xml
      </param-value>
    </init-param>
  </init-params>
</configurable-cache-factory-config>
```

This will, by default, use `coherence-cache-config.xml` as the cache configuration file and `application-context.xml` as the Spring configuration file.

As an alternative to using the configuration file, the `SpringAwareCacheFactory` can be configured programmatically as illustrated inExample 10–2:

Example 10–2 Configuring a SpringAwareCacheFactory Programmatically

```
BeanFactory             bf  = ...
SpringAwareCacheFactory scf = new SpringAwareCacheFactory();

scf.setBeanFactory(bf);
CacheFactory.setConfigurableCacheFactory(scf);
```

Since the `SpringAwareCacheFactory` is `BeanFactoryAware`, it can also be defined in an application context:

Example 10–3 Defining a SpringAwareCacheFactory in an Application Context

```
<bean id="cacheFactory"
      class="com.tangosol.coherence.spring.SpringAwareCacheFactory">
</bean>
```

Taking this a step further, the Coherence `CacheFactory` can be configured inside of the application context:

Example 10–4 Configuring a CacheFactory in an Application Context

```
<bean class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">
  <property name="targetClass" value="com.tangosol.net.CacheFactory"/>
  <property name="targetMethod" value="setConfigurableCacheFactory"/>
  <property name="arguments" ref="cacheFactory"/>
</bean>
```

The application context may have a `CacheStore` configured as in Example 10–5. Note that the `EntityCacheStore` is scoped as **prototype**. This is done because Coherence will manage the lifecycle of the bean when it is retrieved from Spring, just as if Coherence had instantiated the object using `new`.

Example 10–5 Configuring a CacheStore in an Application Context

```
<bean id="dataSource" class="...">
...
</bean>

<bean id="sessionFactory" class="...">
  <property name="dataSource" ref="dataSource"/>
  ...
</bean>
```

```
<bean id="entityCacheStore"
      class="com.company.app.EntityCacheStore"
      scope="prototype">
  <property name="sessionFactory" ref="sessionFactory" />
</bean>
```

Coherence can use the `entityCacheStore` bean as illustrated in Example 10–6. By using the `init-param` element, setter injection can be used to set properties on the bean retrieved from Spring. In the above example, the bean will have the method `setEntityName` invoked with the cache name before it is used by Coherence.

Example 10–6 Configuring Setter Injection to Set Properties on the Bean

```
<?xml version="1.0"?>

<!DOCTYPE cache-config SYSTEM "cache-config.dtd">

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>com.company.app.domain.*</cache-name>
      <scheme-name>distributed-domain</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <distributed-scheme>
      <scheme-name>distributed-domain</scheme-name>
      <backing-map-scheme>
        <read-write-backing-map-scheme>
          <internal-cache-scheme>
            <local-scheme />
          </internal-cache-scheme>
          <cachestore-scheme>
            <class-scheme>
              <class-name>spring-bean:entityCacheStore</class-name>
              <init-params>
                <init-param>
                  <param-name>setEntityName</param-name>
                  <param-value>{cache-name}</param-value>
                </init-param>
              </init-params>
            </class-scheme>
          </cachestore-scheme>
          <write-delay>5s</write-delay>
        </read-write-backing-map-scheme>
      </backing-map-scheme>
      <autostart>true</autostart>
    </distributed-scheme>
  </caching-schemes>
</cache-config>
```

Example 10–7 lists the source for `SpringAwareCacheFactory`. It requires Coherence 3.4.x and Spring 2.x.

Example 10–7 SpringAwareCacheFactory.java

```
/*
* SpringAwareCacheFactory.java
```

```
*
* Copyright 2001-2007 by Oracle. All rights reserved.
*
* Oracle is a registered trademarks of Oracle Corporation and/or its affiliates.
*
* This software is the confidential and proprietary information of
* Oracle Corporation. You shall not disclose such confidential and
* proprietary information and shall use it only in accordance with the
* terms of the license agreement you entered into with Oracle.
*
* This notice may not be removed or altered.
*/
package com.tangosol.coherence.spring;

import com.tangosol.net.BackingMapManagerContext;
import com.tangosol.net.DefaultConfigurableCacheFactory;
import com.tangosol.run.xml.SimpleElement;
import com.tangosol.run.xml.XmlElement;
import com.tangosol.run.xml.XmlHelper;

import com.tangosol.util.ClassHelper;

import java.util.Iterator;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.BeanFactoryAware;

import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;

/**
* SpringAwareCacheFactory provides a facility to access caches declared
* in a "cache-config.dtd" compliant configuration file, similar to its super
* class {@link DefaultConfigurableCacheFactory}.  In addition, this factory
* provides the ability to reference beans in a Spring application context
* by using the use of a class-scheme element.
*
* <p>This factory can be configured to start its own Spring application
* context from which to retrieve these beans.  This can be useful for standalone
* JVMs such as cache servers.  It can also be configured at runtime with a
* preconfigured Spring bean factory.  This can be useful for Coherence
* applications running in an environment that is itself responsible for starting
* the Spring bean factory, such as a web container.
*
* @see #instantiateAny(CacheInfo, XmlElement,
        BackingMapManagerContext, ClassLoader)
*
* @author pperalta Jun 14, 2007
*/
public class SpringAwareCacheFactory
        extends DefaultConfigurableCacheFactory
        implements BeanFactoryAware
    {
    // ----- constructors ---------------------------------------------------

    /**
    * Construct a default DefaultConfigurableCacheFactory using the
    * default configuration file name.
    */
```

```
public SpringAwareCacheFactory()
    {
    super();
    }

/**
* Construct a SpringAwareCacheFactory using the specified path to
* a "cache-config.dtd" compliant configuration file or resource.  This
* will also create a Spring ApplicationContext based on the supplied
* path to a Spring compliant configuration file or resource.
*
* @param sCacheConfig location of a cache configuration
* @param sAppContext  location of a Spring application context
*/
public SpringAwareCacheFactory(String sCacheConfig, String sAppContext)
    {
    super(sCacheConfig);

    azzert(sAppContext != null && sAppContext.length() > 0,
            "Application context location required");

    m_beanFactory = sCacheConfig.startsWith("file:") ? (BeanFactory)
        new FileSystemXmlApplicationContext(sCacheConfig) :
        new ClassPathXmlApplicationContext(sAppContext);

    // register a shutdown hook so the bean factory cleans up
    // upon JVM exit
    ((AbstractApplicationContext) m_beanFactory).registerShutdownHook();
    }

/**
* Construct a SpringAwareCacheFactory using the specified path to
* a "cache-config.dtd" compliant configuration file or resource and
* the supplied Spring BeanFactory.
*
* @param sPath       the configuration resource name or file path
* @param beanFactory Spring BeanFactory used to load Spring beans
*/
public SpringAwareCacheFactory(String sPath, BeanFactory beanFactory)
    {
    super(sPath);

    m_beanFactory = beanFactory;
    }

// ----- extended methods ---------------------------------------------

/**
* Create an Object using the "class-scheme" element.
* <p/>
* In addition to the functionality provided by the super class,
* this will retreive an object from the configured Spring BeanFactory
* for class names that use the following format:
* <pre>
* &lt;class-name&gt;spring-bean:sampleCacheStore&lt;/class-name&gt;
* </pre>
*
* Parameters may be passed to these beans by using setter injection as well:
* <pre>
*   &lt;init-params&gt;
```

```
     *     &lt;init-param&gt;
     *       &lt;param-name&gt;setEntityName&lt;/param-name&gt;
     *       &lt;param-value&gt;{cache-name}&lt;/param-value&gt;
     *     &lt;/init-param&gt;
     *   &lt;/init-params&gt;
     * </pre>
     *
     * Note that Coherence will manage the lifecycle of the instantiated Spring
     * bean, therefore any beans that are retreived using this method should be
     * scoped as a prototype in the Spring configuration file, for example:
     * <pre>
     *   &lt;bean id="sampleCacheStore"
     *         class="com.company.SampleCacheStore"
     *         scope="prototype"/&gt;
     * </pre>
     *
     * @param info      the cache info
     * @param xmlClass  "class-scheme" element.
     * @param context   BackingMapManagerContext to be used
     * @param loader    the ClassLoader to instantiate necessary classes
     *
     * @return a newly instantiated Object
     *
     * @see DefaultConfigurableCacheFactory#instantiateAny(
     *        CacheInfo, XmlElement, BackingMapManagerContext, ClassLoader)
     */
    protected Object instantiateAny(CacheInfo info, XmlElement xmlClass,
            BackingMapManagerContext context, ClassLoader loader)
        {
        if (translateSchemeType(xmlClass.getName()) != SCHEME_CLASS)
            {
            throw new IllegalArgumentException(
                    "Invalid class definition: " + xmlClass);
            }

        String sClass = xmlClass.getSafeElement("class-name").getString();

        if (sClass.startsWith(SPRING_BEAN_PREFIX))
            {
            String sBeanName = sClass.substring(SPRING_BEAN_PREFIX.length());

            azzert(sBeanName != null && sBeanName.length() > 0,
                    "Bean name required");

            XmlElement xmlParams = xmlClass.getElement("init-params");
            XmlElement xmlConfig = null;
            if (xmlParams != null)
                {
                xmlConfig = new SimpleElement("config");
                XmlHelper.transformInitParams(xmlConfig, xmlParams);
                }

            Object oBean = getBeanFactory().getBean(sBeanName);
            if (xmlConfig != null)
                {
                for (Iterator iter = xmlConfig.getElementList().iterator();
iter.hasNext();)
                    {
                    XmlElement xmlElement = (XmlElement) iter.next();
```

```
                String sMethod = xmlElement.getName();
                String sParam  = xmlElement.getString();
                try
                    {
                    ClassHelper.invoke(oBean, sMethod, new Object[]{sParam});
                    }
                catch (Exception e)
                    {
                    ensureRuntimeException(e,"Could not invoke " + sMethod +
                            "(" + sParam + ") on bean " + oBean);
                    }
                }
            }
        return oBean;
        }
    else
        {
        return super.instantiateAny(info, xmlClass, context, loader);
        }
    }

/**
* Get the Spring BeanFactory used by this CacheFactory
* @return the Spring {@link BeanFactory} used by this CacheFactory
*/
public BeanFactory getBeanFactory()
    {
    azzert(m_beanFactory != null, "Spring BeanFactory == null");
    return m_beanFactory;
    }

/**
* Set the Spring BeanFactory used by this CacheFactory
* @param beanFactory the Spring {@link BeanFactory} used by this CacheFactory
*/
public void setBeanFactory(BeanFactory beanFactory)
    {
    m_beanFactory = beanFactory;
    }

// ----- data fields ----------------------------------------------------

/**
* Spring BeanFactory used by this CacheFactory
*/
private BeanFactory m_beanFactory;

/**
* Prefix used in cache configuration "class-name" element to indicate
* this bean is in Spring
*/
private static final String SPRING_BEAN_PREFIX = "spring-bean:";
}
```

11

Specifying a Custom Eviction Policy

The `LocalCache` class is used for size-limited caches. It is used both for caching on-heap objects (as in a local cache or the front portion of a near cache) and as the backing map for a partitioned cache. Applications can provide custom eviction policies for use with a `LocalCache`.

Note that Coherence's default eviction policy is very effective for most workloads; the majority of applications will not need to provide a custom policy. Generally, it is best to restrict the use of eviction policies to scenarios where the evicted data is present in a backing system (that is, the back portion of a near cache or a database). Eviction should be treated as a physical operation (freeing memory) and not a logical operation (deleting an entity).

Example 11–1 shows the implementation of a simple custom eviction policy:

Example 11–1 Implementing a Custom Eviction Policy

```
import com.tangosol.net.cache.CacheEvent;
import com.tangosol.net.cache.LocalCache;
import com.tangosol.net.cache.OldCache;
import com.tangosol.util.AbstractMapListener;
import com.tangosol.util.MapEvent;

import java.util.Iterator;

public class MyEvictionPolicy extends AbstractMapListener implements
OldCache.EvictionPolicy
    {
    LocalCache m_cache = null;

    public void entryInserted(MapEvent evt)
        {
        System.out.println("entryInserted:" + isSynthetic(evt) + evt);
        if (m_cache == null)
            {
            m_cache = (LocalCache) evt.getMap();
            }
        }

    public void entryUpdated(MapEvent evt)
        {
        System.out.println("entryUpdated:" + isSynthetic(evt) + evt);
        }

    public void entryDeleted(MapEvent evt)
        {
        System.out.println("entryDeleted:" + isSynthetic(evt) + evt);
```

```
        }

    String isSynthetic(MapEvent evt)
        {
        // synthetic events are caused by internal processing - eviction or
loading
        return ((CacheEvent) evt).isSynthetic() ? " SYNTHETIC " : " ";
        }

    public void entryTouched(OldCache.Entry entry)
        {
        System.out.println("entryTouched:" + entry.getKey());
        }

    public void requestEviction(int cMaximum)
        {
        int cCurrent = m_cache.getUnits();
        System.out.println("requestEviction: current:" + cCurrent + " to:" +
cMaximum);

        //
        // ... eviction policy calculations ...
        //
        for (Iterator iter = m_cache.entrySet().iterator(); iter.hasNext();)
            {
            OldCache.Entry entry = (OldCache.Entry) iter.next();
            if (m_cache.getUnits() > cMaximum)
                {
                m_cache.evict(entry.getKey());
                }
            else
                {
                break;
                }
            }
        }

    public MyEvictionPolicy()
        {
        }

    }
```

Example 11–2 illustrates a Coherence cache configuration file (`coherence-cache-config.xml`) with an eviction policy:

Example 11–2 Custom Eviction Policy in a coherence-cache-config.xml File

```
<?xml version="1.0"?>

<!DOCTYPE cache-config SYSTEM "cache-config.dtd">

<cache-config>
    <caching-scheme-mapping>
        <cache-mapping>
            <cache-name>test</cache-name>
            <scheme-name>example-near</scheme-name>
        </cache-mapping>
    </caching-scheme-mapping>
```

```
    <caching-schemes>
        <distributed-scheme>
            <scheme-name>example-distributed</scheme-name>
            <service-name>DistributedCache</service-name>

            <backing-map-scheme>
                <local-scheme>
                    <scheme-ref>example-backing-map</scheme-ref>
                </local-scheme>
            </backing-map-scheme>

            <autostart>true</autostart>
        </distributed-scheme>

        <near-scheme>
            <scheme-name>example-near</scheme-name>

            <front-scheme>
                <local-scheme>
                    <eviction-policy>
                        <class-scheme>
                            <class-name>MyEvictionPolicy</class-name>
                        </class-scheme>
                    </eviction-policy>
                    <high-units>10</high-units>
                </local-scheme>
            </front-scheme>

            <back-scheme>
                <distributed-scheme>
                    <scheme-ref>example-distributed</scheme-ref>
                </distributed-scheme>
            </back-scheme>

            <invalidation-strategy>all</invalidation-strategy>
            <autostart>true</autostart>
        </near-scheme>

        <local-scheme>
            <scheme-name>example-backing-map</scheme-name>
            <eviction-policy>HYBRID</eviction-policy>
            <high-units>{back-size-limit 0}</high-units>
            <expiry-delay>{back-expiry 1h}</expiry-delay>
            <flush-delay>1m</flush-delay>
            <cachestore-scheme></cachestore-scheme>
        </local-scheme>
    </caching-schemes>
</cache-config>
```

12

Serialization Paged Cache

Oracle Coherence provides explicit support for efficient caching of huge amounts of automatically-expiring data using potentially high-latency storage mechanisms such as disk files. The benefits include supporting much larger data sets than can be managed in memory, while retaining an efficient expiry mechanism for timing out the management (and automatically freeing the resources related to the management) of that data. Optimal usage scenarios include the ability to store many large objects, XML documents or content that will be rarely accessed, or whose accesses will tolerate a higher latency if the cached data has been paged to disk.

Understanding Serialization Paged Cache

This feature is known as a Serialization Paged Cache:

- *Serialization* implies that objects stored in the cache are serialized and stored in a *Binary Store*; refer to the existing features *Serialization Map* and *Serialization Cache*.
- *Paged* implies that the objects stored in the cache are segmented for efficiency of management; in this case, the pages represent periods of time such that the cache could be divided into pages of one hour each.
- *Cache* implies that there can be limits specified to the size of the cache; in this case, the limit is the maximum number of concurrent pages that the cache will manage before expiring pages, starting with the oldest page.

The result is a feature that organizes data in the cache based on the time that the data was placed in the cache, and then is capable of efficiently expiring that data from the cache, an entire page at a time, and typically without having to reload any data from disk.

Configuring Serialization Paged Cache

The primary configuration for the Serialization Paged Cache is composed of two parameters: The number of pages that the cache will manage, and the length of time represented by each page. For example, to cache data for one day, the cache can be configured as 24 pages of one hour each, or 96 pages of 15 minutes each, and so on.

Each page of data in the cache is managed by a separate Binary Store. The cache requires a *Binary Store Manager*, which provides the means to create and destroy these Binary Stores. Coherence provides Binary Store Managers for all of the built-in Binary Store implementations, including disk (referred to as "LH") and the various NIO implementations.

Optimizing a Partitioned Cache Service

Coherence provides an optimization for the partitioned cache service, since - when it is used to back a partitioned cache—the data being stored in any of the Serialization Maps and Caches is entirely binary in form. This is called the Binary Map optimization, and when it is enabled, it gives the Serialization Map, the Serialization Cache and the Serialization Paged Cache permission to assume that all data being stored in the cache is binary. The result of this optimization is a lower CPU and memory utilization, and also slightly higher performance.

Configuring for High Availability

Explicit support is also provided in the Serialization Paged Cache for the high-availability features of the partitioned cache service, by providing a configuration that can be used for the primary storage of the data and a configuration that is optimized for the backup storage of the data. The configuration for the backup storage is known as a passive model, because it does not actively expire data from its storage, but rather reflects the expiration that is occurring on the primary cache storage. When using the high-availability data feature (a backup count of one or greater; the default is one) for a partitioned cache service, and using the Serialization Paged Cache as the backing storage for the service, we strongly suggest that you also use the Serialization Paged Cache as the backup store, and configure the backup with the passive option.

Configuring Load Balancing and Failover

When used with the distributed cache service, special considerations should be made for load balancing and failover purposes. The partition-count parameter of the distributed cache service should be set higher than normal if the amount of cache data is very large or huge; that will break up the overall cache into smaller chunks for load-balancing and recovery processing due to failover. For example, if the cache is expected to be one terabyte in size, twenty thousand partitions will break the cache up into units averaging about 50MB in size. If a unit (the size of a partition) is too large, it will cause an out-of-memory condition when load-balancing the cache. (Remember to make sure that the partition count is a prime number; see http://primes.utm.edu/lists/small/ for lists of prime numbers that you can use.)

Supporting Huge Caches

To support huge caches (for example, terabytes) of expiring data, the expiration processing is performed concurrently on a daemon thread with no interruption to the cache processing. The result is that many thousands or millions of objects can exist in a single cache page, and they can be expired asynchronously, thus avoiding any interruption of service. The daemon thread is an option that is enabled by default, but it can be disabled.

When the cache is used for large amounts of data, the pages will typically be disk-backed. Since the cache eventually expires each page, thus releasing the disk resources, the cache uses a virtual erase optimization by default. This means that data that is explicitly removed or expired from the cache is not actually removed from the underlying Binary Store, but when a page (a Binary Store) is completely emptied, it will be erased in its entirety. This reduces I/O by a considerable margin, particularly during expiry processing and during operations such as load-balancing that have to redistribute large amounts of data within the cluster. The cost of this optimization is that the disk files (if a disk-based Binary Store option is used) will tend to be larger than the data that they are managing would otherwise imply; since disk space is

considered to be inexpensive compared to other factors such as response times, the virtual erase optimization is enabled by default, but it can be disabled. Note that the disk space is typically allocated locally to each server, and thus a terabyte cache partitioned over one hundred servers would only use about 20GB of disk space per server (10GB for the primary store and 10GB for the backup store, assuming one level of backup.)

13

Pre-Loading the Cache

This section describes different patterns you can use to pre-load the cache. The patterns include bulk loading and distributed loading.

Performing Bulk Loading and Processing

Example 13–5, `PagedQuery.java`, demonstrates techniques for efficiently bulk loading and processing items in a Coherence Cache.

Bulk Writing to a Cache

A common scenario when using Coherence is to pre-populate a cache before the application uses it. A simple way to do this is illustrated by the Java code in Example 13–1:

Example 13–1 Pre-Loading a Cache

```
public static void bulkLoad(NamedCache cache, Connection conn)
    {
    Statement s;
    ResultSet rs;

    try
        {
        s = conn.createStatement();
        rs = s.executeQuery("select key, value from table");
        while (rs.next())
            {
            Integer key   = new Integer(rs.getInt(1));
            String  value = rs.getString(2);
            cache.put(key, value);
            }
        ...
        }
    catch (SQLException e)
        {...}
    }
```

This technique works, but each call to put may result in network traffic, especially for partitioned and replicated caches. Additionally, each call to put will return the object it just replaced in the cache (per the `java.util.Map interface`) which will add more unnecessary overhead. Loading the cache can be made much more efficient by using the `ConcurrentMap.putAll` method instead. This is illustrated in Example 13–2:

Example 13–2 Pre-Loading a Cache Using ConcurrentMap.putAll

```
public static void bulkLoad(NamedCache cache, Connection conn)
    {
    Statement s;
    ResultSet rs;
    Map       buffer = new HashMap();

    try
        {
        int count = 0;
        s = conn.createStatement();
        rs = s.executeQuery("select key, value from table");
        while (rs.next())
            {
            Integer key   = new Integer(rs.getInt(1));
            String  value = rs.getString(2);
            buffer.put(key, value);

            // this loads 1000 items at a time into the cache
            if ((count++ % 1000) == 0)
                {
                cache.putAll(buffer);
                buffer.clear();
                }
            }
        if (!buffer.isEmpty())
            {
            cache.putAll(buffer);
            }
        ...
        }
    catch (SQLException e)
        {...}
    }
```

Efficient processing of filter results

Coherence provides the ability to query caches based on criteria by using the `Filter` API. Here is an example (given entries with integers as keys and strings as values):

Example 13–3 Using a Filter to Query a Cache

```
NamedCache c = CacheFactory.getCache("test");

// Search for entries that start with 'c'
Filter query = new LikeFilter(IdentityExtractor.INSTANCE, "c%", '\\', true);

// Perform query, return all entries that match
Set results = c.entrySet(query);
for (Iterator i = results.iterator(); i.hasNext();)
    {
    Map.Entry e = (Map.Entry) i.next();
    out("key: "+e.getKey() + ", value: "+e.getValue());
    }
```

This example works for small data sets, but it may encounter problems, such as running out of heap space, if the data set is too large. Example 13–4 illustrates a pattern to process query results in batches to avoid this problem:

Example 13–4 Processing Query Results in Batches

```
public static void performQuery()
    {
    NamedCache c = CacheFactory.getCache("test");

    // Search for entries that start with 'c'
    Filter query = new LikeFilter(IdentityExtractor.INSTANCE, "c%", '\\', true);

    // Perform query, return keys of entries that match
    Set keys = c.keySet(query);

    // The amount of objects to process at a time
    final int BUFFER_SIZE = 100;

    // Object buffer
    Set buffer = new HashSet(BUFFER_SIZE);

    for (Iterator i = keys.iterator(); i.hasNext();)
        {
        buffer.add(i.next());

        if (buffer.size() >= BUFFER_SIZE)
            {
            // Bulk load BUFFER_SIZE number of objects from cache
            Map entries = c.getAll(buffer);

            // Process each entry
            process(entries);

            // Done processing these keys, clear buffer
            buffer.clear();
            }
        }
        // Handle the last partial chunk (if any)
        if (!buffer.isEmpty())
            {
            process(c.getAll(buffer));
            }
    }

public static void process(Map map)
    {
    for (Iterator ie = map.entrySet().iterator(); ie.hasNext();)
        {

        Map.Entry e = (Map.Entry) ie.next();
        out("key: "+e.getKey() + ", value: "+e.getValue());
        }
    }
```

In this example, all keys for entries that match the filter are returned, but only `BUFFER_SIZE` (in this case, 100) entries are retrieved from the cache at a time.

Note that `LimitFilter` can be used to process results in parts, similar to the example above. However `LimitFilter` is meant for scenarios where the results will be paged, such as in a user interface. It is not an efficient means to process all data in a query result.

A Bulk Loading and Processing Example

Example 13–5 illustrates `PagedQuery.java`, a sample program that demonstrates the concepts described in the previous section.

To run the example, follow these steps:

1. Save the following Java file as `com/tangosol/examples/PagedQuery.java`
2. Point the classpath to the Coherence libraries and the current directory
3. Compile and run the example

Example 13–5 A Sample Bulk Loading Program

```
package com.tangosol.examples;

import com.tangosol.net.CacheFactory;
import com.tangosol.net.NamedCache;
import com.tangosol.net.cache.NearCache;
import com.tangosol.util.Base;
import com.tangosol.util.Filter;
import com.tangosol.util.filter.LikeFilter;

import java.io.Serializable;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Random;
import java.util.Set;
import java.util.HashSet;


/**
* This sample application demonstrates the following:
* <ul>
* <li>
* <b>Obtaining a back cache from a near cache for populating a cache.</b>
* Since the near cache holds a limited subset of the data in a cache it is
* more efficient to bulk load data directly into the back cache instead of
* the near cache.
* </li>
* <li>
* <b>Populating a cache in bulk using <tt>putAll</tt>.</b>
* This is more efficient than <tt>put</tt> for a large amount of entries.
* </li>
* <li>
* <b>Executing a filter against a cache and processing the results in bulk.</b>
* This sample issues a query against the cache using a filter.  The result is
* a set of keys that represent the query results.  Instead of iterating
* through the keys and loading each item individually with a <tt>get</tt>,
* this sample loads entries from the cache in bulk using <tt>getAll</tt> which
* is more efficient.
* </li>
*
* @author cp
*/
public class PagedQuery
        extends Base
    {
    /**
```

```
* Command line execution entry point.
*/
public static void main(String[] asArg)
    {
    NamedCache cacheContacts = CacheFactory.getCache("contacts",
            Contact.class.getClassLoader());

    populateCache(cacheContacts);

    executeFilter(cacheContacts);

    CacheFactory.shutdown();
    }

// ----- populate the cache ---------------------------------------------

/**
* Populate the cache with test data.  This example shows how to populate
* the cache a chunk at a time using {@link NamedCache#putAll} which is more
* efficient than {@link NamedCache#put}.
*
* @param cacheDirect  the cache to populate. Note that this should <b>not</b>
*                     be a near cache since that will thrash the cache
*                     if the load size exceeds the near cache max size.
*/
public static void populateCache(NamedCache cacheDirect)
    {
    if (cacheDirect.isEmpty())
        {
        Map mapBuffer = new HashMap();
        for (int i = 0; i < 100000; ++i)
            {
            // make up some fake data
            Contact contact = new Contact();
            contact.setName(getRandomName() + ' ' + getRandomName());
            contact.setPhone(getRandomPhone());
            mapBuffer.put(new Integer(i), contact);

            // this loads 1000 items at a time into the cache
            if ((i % 1000) == 0)
                {
                out("Adding "+mapBuffer.size()+" entries to cache");
                cacheDirect.putAll(mapBuffer);
                mapBuffer.clear();
                }
            }
        if (!mapBuffer.isEmpty())
            {
            cacheDirect.putAll(mapBuffer);
            }
        }
    }

/**
* Creates a random name.
*
* @return  a random string between 4 to 11 chars long
*/
public static String getRandomName()
    {
```

```
    Random rnd = getRandom();
    int    cch = 4 + rnd.nextInt(7);
    char[] ach = new char[cch];
    ach[0] = (char) ('A' + rnd.nextInt(26));
    for (int of = 1; of < cch; ++of)
        {
        ach[of] = (char) ('a' + rnd.nextInt(26));
        }
    return new String(ach);
    }

/**
* Creates a random phone number
*
* @return  a random string of integers 10 chars long
*/
public static String getRandomPhone()
    {
    Random rnd = getRandom();
    return "("
        + toDecString(100 + rnd.nextInt(900), 3)
        + ") "
        + toDecString(100 + rnd.nextInt(900), 3)
        + "-"
        + toDecString(10000, 4);
    }

// ----- process the cache ----------------------------------------------

/**
* Query the cache and process the results in batches.  This example
* shows how to load a chunk at a time using {@link NamedCache#getAll}
* which is more efficient than {@link NamedCache#get}.
*
* @param cacheDirect  the cache to issue the query against
*/
private static void executeFilter(NamedCache cacheDirect)
    {
    Filter query = new LikeFilter("getName", "C%");

    // Let's say we want to process 100 entries at a time
    final int CHUNK_COUNT = 100;

    // Start by querying for all the keys that match
    Set setKeys = cacheDirect.keySet(query);

    // Create a collection to hold the "current" chunk of keys
    Set setBuffer = new HashSet();

    // Iterate through the keys
    for (Iterator iter = setKeys.iterator(); iter.hasNext(); )
        {
        // Collect the keys into the current chunk
        setBuffer.add(iter.next());

        // handle the current chunk when it gets big enough
        if (setBuffer.size() >= CHUNK_COUNT)
            {
            // Instead of retrieving each object with a get,
            // retrieve a chunk of objects at a time with a getAll.
```

```
            processContacts(cacheDirect.getAll(setBuffer));
            setBuffer.clear();
            }
        }

    // Handle the last partial chunk (if any)
    if (!setBuffer.isEmpty())
        {
        processContacts(cacheDirect.getAll(setBuffer));
        }
    }

/**
* Process the map of contacts. In a real application some sort of
* processing for each map entry would occur. In this example each
* entry is logged to output.
*
* @param map  the map of contacts to be processed
*/
public static void processContacts(Map map)
    {
    out("processing chunk of " + map.size() + " contacts:");
    for (Iterator iter = map.entrySet().iterator(); iter.hasNext(); )
        {
        Map.Entry entry = (Map.Entry) iter.next();
        out("  [" + entry.getKey() + "]=" + entry.getValue());
        }
    }

// ----- inner classes ------------------------------------------------

/**
* Sample object used to populate cache
*/
public static class Contact
        extends Base
        implements Serializable
    {
    public Contact() {}

    public String getName()
        {
        return m_sName;
        }
    public void setName(String sName)
        {
        m_sName = sName;
        }

    public String getPhone()
        {
        return m_sPhone;
        }
    public void setPhone(String sPhone)
        {
        m_sPhone = sPhone;
        }

    public String toString()
        {
```

```
            return "Contact{"
                    + "Name=" + getName()
                    + ", Phone=" + getPhone()
                    + "}";
            }

        public boolean equals(Object o)
            {
            if (o instanceof Contact)
                {
                Contact that = (Contact) o;
                return equals(this.getName(), that.getName())
                    && equals(this.getPhone(), that.getPhone());
                }
            return false;
            }

        public int hashCode()
            {
            int result;
            result = (m_sName != null ? m_sName.hashCode() : 0);
            result = 31 * result + (m_sPhone != null ? m_sPhone.hashCode() : 0);
            return result;
            }

        private String m_sName;
        private String m_sPhone;
        }
    }
```

Example 13–6 illustrates the terminal output from Coherence when you compile and run the example:

Example 13–6 Terminal Output from the Bulk Loading Program

```
$ export COHERENCE_HOME=[**Coherence install directory**]

$ export CLASSPATH=$COHERENCE_HOME/lib/coherence.jar:.

$ javac com/tangosol/examples/PagedQuery.java

$ java com.tangosol.examples.PagedQuery

2008-09-15 12:19:44.156 Oracle Coherence 3.4/405 <Info> (thread=main, member=n/a):
Loaded operational configuration from
 resource "jar:file:/C:/coherence/lib/coherence.jar!/tangosol-coherence.xml"
2008-09-15 12:19:44.171 Oracle Coherence 3.4/405 <Info> (thread=main, member=n/a):
Loaded operational overrides from
resource
"jar:file:/C:/coherence/lib/coherence.jar!/tangosol-coherence-override-dev.xml"
2008-09-15 12:19:44.171 Oracle Coherence 3.4/405 <D5> (thread=main, member=n/a):
Optional configuration override
"/tangosol-coherence-override.xml" is not specified

Oracle Coherence Version 3.4/405
 Grid Edition: Development mode
Copyright (c) 2000-2008 Oracle. All rights reserved.

2008-09-15 12:19:44.812 Oracle Coherence GE 3.4/405 <D5> (thread=Cluster,
member=n/a): Service Cluster joined the cluster
```

```
with senior service member n/a
2008-09-15 12:19:48.062 Oracle Coherence GE 3.4/405 <Info> (thread=Cluster,
member=n/a): Created a new cluster with
Member(Id=1, Timestamp=2008-09-15 12:19:44.609, Address=xxx.xxx.x.xxx:8088,
MachineId=26828, Edition=Grid Edition,
Mode=Development, CpuCount=2, SocketCount=1)
UID=0xC0A800CC00000112B9BC9B6168CC1F98
Adding 1024 entries to cache
Adding 1024 entries to cache

...repeated many times...

Adding 1024 entries to cache
Adding 1024 entries to cache
Adding 1024 entries to cache
processing chunk of 100 contacts:
  [25827]=Contact{Name=Cgkyleass Kmknztk, Phone=(285) 452-0000}
  [4847]=Contact{Name=Cyedlujlc Ruexrtgla, Phone=(255) 296-0000}
...repeated many times
  [33516]=Contact{Name=Cjfwlxa Wsfhrj, Phone=(683) 968-0000}
  [71832]=Contact{Name=Clfsyk Dwncpr, Phone=(551) 957-0000}
processing chunk of 100 contacts:
  [38789]=Contact{Name=Cezmcxaokf Kwztt, Phone=(725) 575-0000}
  [87654]=Contact{Name=Cuxcwtkl Tqxmw, Phone=(244) 521-0000}
...repeated many times
  [96164]=Contact{Name=Cfpmbvq Qaxty, Phone=(596) 381-0000}
  [29502]=Contact{Name=Cofcdfgzp Nczpdg, Phone=(563) 983-0000}
...
processing chunk of 80 contacts:
  [49179]=Contact{Name=Czbjokh Nrinuphmsv, Phone=(140) 353-0000}
  [84463]=Contact{Name=Cyidbd Rnria, Phone=(571) 681-0000}
...
  [2530]=Contact{Name=Ciazkpbos Awndvrvcd, Phone=(676) 700-0000}
  [9371]=Contact{Name=Cpqo Rmdw, Phone=(977) 729-0000}
```

Performing Distributed Bulk Loading

When pre-populating a Coherence partitioned cache with a large data set, it may be more efficient to distribute the work to Coherence cluster members. Distributed loading will allow for higher data throughput rates to the cache by leveraging the aggregate network bandwidth and CPU power of the cluster. When performing a distributed load, the application will need to decide on the following:

- which cluster members will perform the load
- how to divide the data set among the members

The application should consider the load that will be placed on the underlying data source (such as a database or file system) when selecting members and dividing work. For example, a single database can easily be overwhelmed if too many members execute queries concurrently.

A Distributed Bulk Loading Example

This section outlines the general steps to perform a simple distributed load. The example assumes that the data is stored in files and will be distributed to all storage-enabled members of a cluster.

1. Retrieve the set of storage-enabled members. For example, the following method uses the `getStorageEnabledMembers` method to retrieve the storage-enabled members of a distributed cache.

Example 13–7 Retrieving Storage-Enabled Members of the Cache

```
protected Set getStorageMembers(NamedCache cache)
        {
        return ((DistributedCacheService) cache.getCacheService())
                .getStorageEnabledMembers();
        }
```

2. Divide the work among the storage enabled cluster members. For example, the following routine returns a map, keyed by member, containing a list of files assigned to that member.

Example 13–8 Routine to Get a List of Files Assigned to a Cache Member

```
protected Map<Member, List<String>> divideWork(Set members, List<String>
fileNames)
        {
        Iterator i = members.iterator();
        Map<Member, List<String>> mapWork = new HashMap(members.size());
        for (String sFileName : fileNames)
            {
            Member member = (Member) i.next();
            List<String> memberFileNames = mapWork.get(member);
            if (memberFileNames == null)
                {
                memberFileNames = new ArrayList();
                mapWork.put(member, memberFileNames);
                }
            memberFileNames.add(sFileName);

            // recycle through the members
            if (!i.hasNext())
                {
                i = members.iterator();
                }
            }
        return mapWork;
        }
```

3. Launch a task that will perform the load on each member. For example, use Coherence's `InvocationService` to launch the task. In this case, the implementation of `LoaderInvocable` will need to iterate through `memberFileNames` and process each file, loading its contents into the cache. The cache operations normally performed on the client will need to be executed through the `LoaderInvocable`.

Example 13–9 Class to Load Each Member of the Cache

```
public void load()
        {
        NamedCache cache = getCache();

        Set members = getStorageMembers(cache);

        List<String> fileNames = getFileNames();
```

```
        Map<Member, List<String>> mapWork = divideWork(members, fileNames);

        InvocationService service = (InvocationService)
                CacheFactory.getService("InvocationService");

        for (Map.Entry<Member, List<String>> entry : mapWork.entrySet())
            {
            Member member = entry.getKey();
            List<String> memberFileNames = entry.getValue();

            LoaderInvocable task = new LoaderInvocable(memberFileNames,
cache.getCacheName());
            service.execute(task, Collections.singleton(member), this);
            }
        }
```

Running a Distributed Bulk Loading Example

The examples in the previous section are taken from `DistributedLoader.java`, which is included in the attached zip file, `coherence-example-distributedload.zip`. This sample application uses the `InvocationService` to distribute the task of loading data into a cache. Each storage-enabled member of the cluster will be responsible for loading a portion of the data. The data in this example will come from several CSV files and the unit of work is one file. All storage-enabled nodes must have access to all of the data files.

To build and run the example, you must have the following software installed:

- J2SE SDK 1.4 or later
- Apache Ant
- Oracle Coherence

Building the Sample Application

1. Extract the contents of `coherence-example-distributedload.zip` into your file system.
2. Update the `bin\set-env.cmd` file to reflect your system environment.
3. Open a command prompt and execute the following command in the bin directory to build the samples:

   ```
   C:\distributedLoad\bin\ant.cmd build
   ```

   After running the samples, you can completely remove all build artifacts from your file system by running the clean command:

   ```
   C:\distributedLoad\bin\ant.cmd clean
   ```

Running the Sample Application

1. Start multiple cache servers (from the `bin` directory):

   ```
   C:\distributedLoad\bin\server.cmd
   ```

2. Run the client loader (from the `bin` directory):

   ```
   C:\distributedLoad\bin\load.cmd
   ```

After entering `load.cmd` on the command line, you will messages indicating that the various members are joining the services. Then you will see messages that indicate that the date is being distributed among the members. In this example, four cache servers were started.

Example 13–10 Server Response from the Sample Distributed Loading Application

```
...
  Member(Id=1, Timestamp=2008-09-15 16:49:04.359, Address=ip_address:8088, Mac
hineId=49690, Location=site:us.oracle.com,machine:machine_name,process:21952, Rol
e=CoherenceServer)
  Member(Id=2, Timestamp=2008-09-15 16:49:50.25, Address=ip_address:8089, Mach
ineId=49690, Location=site:us.oracle.com,machine:machine_name,process:16604, Role
=CoherenceServer)
  Member(Id=3, Timestamp=2008-09-15 16:49:54.937, Address=ip_address:8090, Mac
hineId=49690, Location=site:us.oracle.com,machine:machine_name,process:7344, Role
=CoherenceServer)
  Member(Id=4, Timestamp=2008-09-15 16:49:58.734, Address=ip_address:8091, Mac
hineId=49690, Location=site:us.oracle.com,machine:machine_name,process:19052, Role
=CoherenceServer)
  )
2008-09-15 16:51:00.593/4.890 Oracle Coherence GE 3.4/405 <D5>
(thread=main, member=5): Loading stock file names from '..\data'
2008-09-15 16:51:00.593/4.890 Oracle Coherence GE 3.4/405 <D5>
(thread=main, member=5): Files to load: [..\data\AAPL.CSV, ..\data\BT.CSV, ..
\data\DELL.CSV, ..\data\GOOG.CSV, ..\data\HPQ.CSV, ..\data\JAVA.CSV, ..\data\MSF
T.CSV, ..\data\ORCL.CSV, ..\data\YHOO.CSV]
2008-09-15 16:51:00.593/4.890 Oracle Coherence GE 3.4/405 <D5>
(thread=main, member=5): Invoking load on member(Id=2) for files [..\data\BT.
CSV, ..\data\JAVA.CSV]
2008-09-15 16:51:00.640/4.937 Oracle Coherence GE 3.4/405 <D5>
(thread=main, member=5): Invoking load on member(Id=3) for files [..\data\DEL
L.CSV, ..\data\MSFT.CSV]
2008-09-15 16:51:00.750/5.047 Oracle Coherence GE 3.4/405 <D5>
(thread=main, member=5): Invoking load on member(Id=4) for files [..\data\GOO
G.CSV, ..\data\ORCL.CSV]
2008-09-15 16:51:00.781/5.078 Oracle Coherence GE 3.4/405 <D5>
(thread=main, member=5): Invoking load on member(Id=1) for files [..\data\AAP
L.CSV, ..\data\HPQ.CSV, ..\data\YHOO.CSV]
2008-09-15 16:51:27.500/31.797 Oracle Coherence GE 3.4/405 <D5>
(thread=Invocation:InvocationService, member=5): Finished loading on member:
 Member(Id=4, Timestamp=2008-09-15 16:49:58.734, Address=ip_address:8091, Mach
ineId=49690, Location=site:us.oracle.com,machine:machine_name,process:19052, Role
=CoherenceServer)
2008-09-15 16:51:31.640/35.937 Oracle Coherence GE 3.4/405 <D5>
(thread=Invocation:InvocationService, member=5): Finished loading on member:
 Member(Id=2, Timestamp=2008-09-15 16:49:50.25, Address=ip_address:8089, Machi
neId=49690, Location=site:us.oracle.com,machine:machine_name,process:16604, Role=
CoherenceServer)
2008-09-15 16:51:32.812/37.109 Oracle Coherence GE 3.4/405 <D5>
(thread=Invocation:InvocationService, member=5): Finished loading on member:
 Member(Id=3, Timestamp=2008-09-15 16:49:54.937, Address=ip_address:8090, Mach
ineId=49690, Location=site:us.oracle.com,machine:machine_name,process:7344, Role=
CoherenceServer)
2008-09-15 16:51:37.750/42.047 Oracle Coherence GE 3.4/405 <D5>
(thread=Invocation:InvocationService, member=5): Finished loading on member:
 Member(Id=1, Timestamp=2008-09-15 16:49:04.359, Address=ip_address:8088, Mach
ineId=49690, Location=site:us.oracle.com,machine:machine_name,process:21952, Role
=CoherenceServer)
2008-09-15 16:51:37.796/42.093 Oracle Coherence GE 3.4/405 <D5>
```

```
(thread=main, member=5): Load finished in 37.20 secs
2008-09-15 16:51:37.812/42.109 Oracle Coherence GE 3.4/405 <D5> (thread=main,
member=5): Final cache size: 47131

C:\distributedload\bin>
```

14

Constraints on Re-entrant Calls

Coherence is architected as a collection of services. Each Coherence service consists of the Coherence code that implements the service, along with an associated configuration. The service runs on an allocated pool of threads with associated queues that receive requests and return responses.

Coherence does not support re-entrant calls. A "re-entrant service call" occurs when a service thread, in the act of processing a request, makes a request to that same service. As all requests to a service are delivered by using the inbound queue, and Coherence uses a thread-per-request model, this means that each reentrant request would consume an additional thread (the calling thread would block while awaiting a response). Note that this is distinct from the similar-sounding concept of recursion.

Re-entrancy, Services, and Service Threads

A service is defined as a unique combination of a service name and a service type (such as Invocation, Replicated, or Distributed). For example, you can call from a distributed service `Dist-Customers` into a distributed service named `Dist-Inventory`, or from a distributed service named `Dist-Customers` into a replicated service named `Repl-Catalog`. Service names are configured in the cache configuration file using the `<service-name>` element.

Parent-Child Object Relationships

In the current implementation of Coherence, it is irrelevant whether the "call" is local or remote. This complicates the use of key association to support the efficient assembly of parent-child relationships. If you use key association to co-locate a Parent object with all of its Child objects, then you cannot send an EntryProcessor to the parent object and have that `EntryProcessor` "grab" the (local) Child objects. This is true even though the Child objects are already in-process.

To access both a parent object and its child objects, you can do any of the following:

- Embed the child objects within the parent object (using an "aggregate" pattern) or,
- Use direct access to the server-side backing map (which requires advanced knowledge to do safely), or
- Run the logic on another service (for example, Invocation targeted by using `PartitionedService.getKeyOwner`), and have that service access the data by using `NamedCache` interfaces, or
- Place the child objects on another service which would allow reentrant calls (but incur network access since there is no affinity between partitions in different cache services).

Using the aggregate pattern is probably the best solution for most use cases. However, if this is impractical (due to size restrictions, for example), and there is a need to access both the parent and child objects without using a client/server model, the Invocation service approach is probably the best compromise for most use cases.

Avoiding Deadlock

Even when re-entrancy is allowed, one should be very careful to avoid saturating the thread pool and causing catastrophic deadlock. For example, if service A calls service B, and service B calls service A, there is a possibility that a sufficient number of concurrent calls could fill one of the thread pools, which would cause a form of deadlock. As with traditional locking, using ordered access (for example, service A can call service B, but not vice versa) can help.

So:

- Service A calling into service A is never allowed
- Service A calling into service B, and service B calling back into service A is technically allowed but is deadlock-prone and should be avoided if at all possible.
    - Service A calling into service B, and service B calling into service C, and service C calling back into service A is similarly restricted
- Service A calling into service B is allowed
    - Service A calling into service B, and service B calling into service C, and service A calling into service C is similarly allowed

A service thread is defined as any thread involved in *fulfilling* a Coherence API request. Service threads may invoke any of the following entities:

- Map Listeners
- Membership Listeners
- Network Filters
- Custom Serialization/Deserialization such as `ExternalizableLite` implementations
- Backing Map Listeners
- `CacheLoader/CacheStore` Modules
- Query logic such as `Aggregators`, `Filters`, `ValueExtractors` and `Comparators`
- Entry Processors
- Triggers
- `InvocationService` Invocables

These entities should never make re-entrant calls back into their own services.

Re-entrancy and Listeners

Membership listeners can be used to observe the active set of members participating in the cluster or a specific service. Membership listener threading can be complex; thus, re-entrant calls from a member listener to any Coherence service should be avoided.

Part II

Testing and Tuning

This section contains the following chapters:

15

Evaluating Performance and Scalability

The Coherence distributed caches will often be evaluated with respect to pre-existing local caches. The local caches generally take the form of in-processes hash maps. While Coherence does include facilities for in-process non-clustered caches, direct performance comparison between local caches and a distributed cache not realistic. By the very nature of being out of process, the distributed cache must perform serialization and network transfers. For this cost you gain cluster wide coherency of the cache data, and data and query scalability beyond what a single JVM or machine is capable of providing. This does not mean that you cannot achieve impressive performance using a distributed cache, but it must be evaluated in the correct context.

Measuring Latency and Throughput

When evaluating performance you try to establish two things, latency, and throughput. A simple performance analysis test may simply try performing a series of timed cache accesses in a tight loop. While these tests may accurately measure latency, to measure maximum throughput on a distributed cache a test must make use of multiple threads concurrently accessing the cache, and potentially multiple test clients. In a single threaded test the client thread will naturally spend the majority of the time simply waiting on the network. By running multiple clients/threads, you can more efficiently make use of your available processing power by issuing several requests in parallel. The use of batching operations can also be used to increase the data density of each operation. As you add threads, you should see that the throughput continues to increase until you've maxxed-out the CPU or network, while the overall latency remains constant for the same period.

Demonstrating Scalability

To show true linear scalability as you increase cluster size, you need to be prepared to be add hardware, and not simply JVMs to the cluster. Adding JVMs to a single machine will scale only up to the point where the CPU or network are fully used.

Plan on testing with clusters with more then just two cache servers (storage enabled nodes). The jump from one to two cache servers will not show the same scalability as from two to four. The reason for this is because by default Coherence will maintain one backup copy of each piece of data written into the cache. The process of maintaining backups only begins when there are two storage-enabled nodes in the cluster (there must be a place to put the backup). Thus when you move from a one to two, the amount of work involved in a mutating operation such as cache.put actually doubles, but beyond that the amount of work stays fixed, and will be evenly distributed across the nodes.

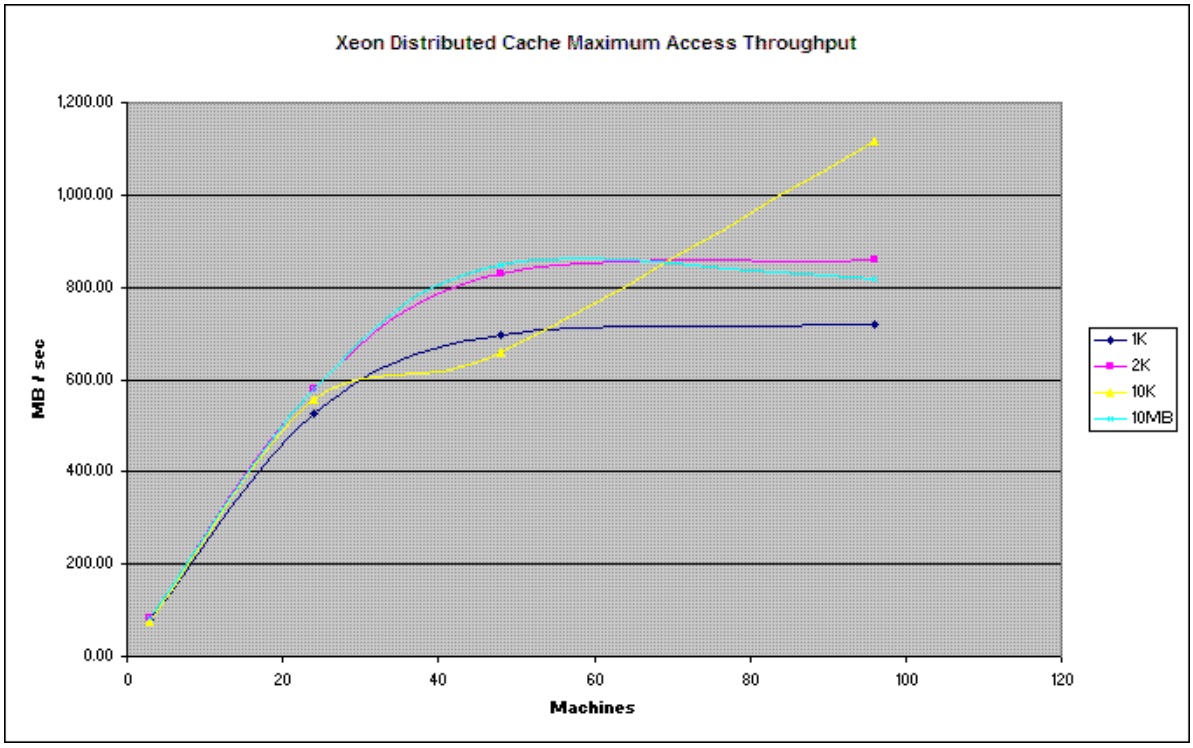
Tuning Your Environment

To get the most out of your cluster it is important that you've tuned of your environment and JVMs. Chapter 20, "Performance Tuning,", provides good start to getting the most out of your environment. For example, Coherence includes a registry script for Windows (`optimize.reg`), which will adjust a few critical settings and allow Windows to achieve much higher data rates.

Measurements on a Large Cluster

The following graphs show the results of scaling out a cluster in an environment of 100 machines. In this particular environment, Coherence was able to scale to the limits of the network's switching infrastructure. Namely, there were 8 sets of ~12 machines, each set having a 4Gbs link to a central switch. Thus for this test Coherence's network throughput scales up to ~32 machines at which point it has maxxed-out the available bandwidth, beyond that it continues to scale in total data capacity.

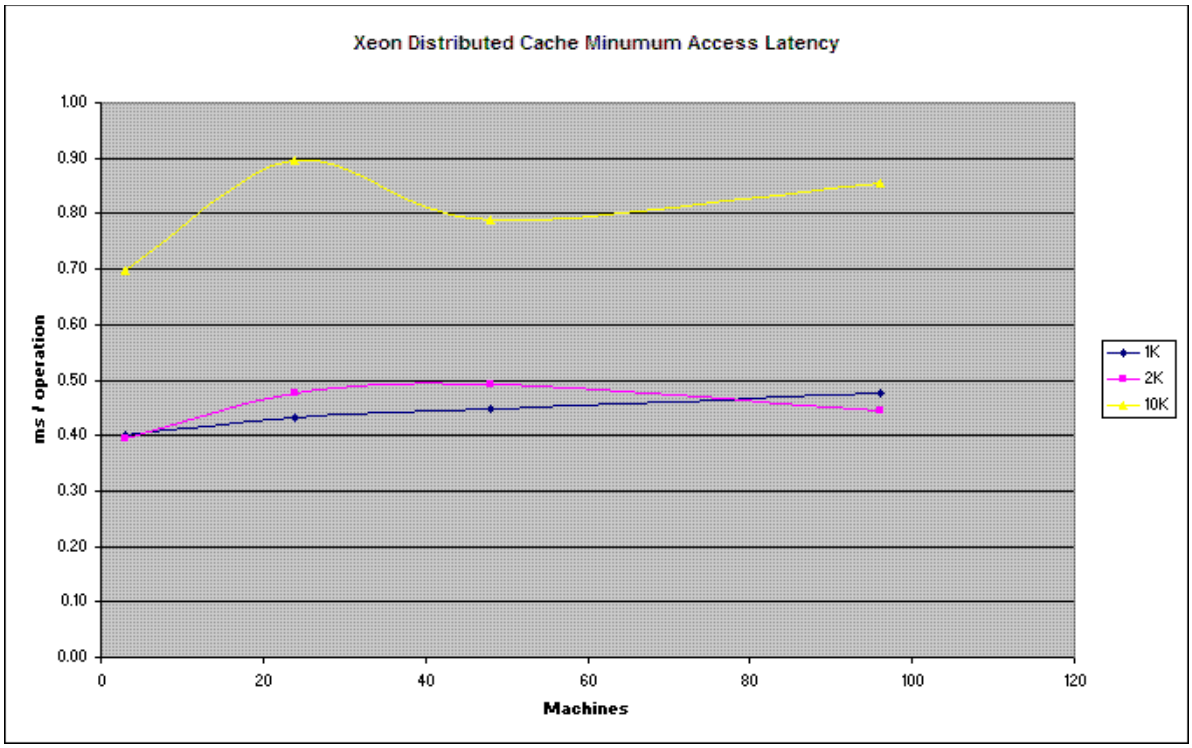
Figure 15–1 Coherence Throughput versus Number of Machines



This figure is described in he text.

****

Figure 15–2 Coherence Latency versus Number of Machines



This figure is described in the text.

****

Latency for 10MB operations (~300ms) is not included in the graph for display reasons, as the payload is 1000x the next payload size.

16

Performing a Multicast Connectivity Test

Included with Coherence is a Multicast Test utility, which helps you determine if multicast is enabled between two or more computers. This is a connectivity test, not a load test, each instance will by default only transmit a single multicast packet once every two seconds. For network load testing, see Chapter 17, "Performing a Datagram Test for Network Performance."

Running the Multicast Test Utility

The Multicast Test utility supports a large number of configuration options, though only a few are required for basic operation. To run the Multicast Test utility use the following syntax from the command line:

```
java com.tangosol.net.MulticastTest <command value> <command value> ...
```

Table 16–1 describes the available command line options for the Multicast Test utility.

Table 16–1 Command Line Options for the Multicast Test Utility

Command	Required/ Optional	Description	Default
-local	Optional	The address of the NIC to transmit on, specified as an IP address	localhost
-group	Optional	The multicast address to use, specified as IP:port.	237.0.0.1:9000
-ttl	Optional	The time to live for multicast packets.	4
-delay	Optional	The delay between transmitting packets, specified in seconds.	2
-display	Optional	The number of bytes to display from unexpected packets.	0

Sample Commands

```
java com.tangosol.net.MulticastTest -group 237.0.0.1:9000
```

For ease of use, `multicast-test.sh` and `multicast-test.cmd` scripts are provided in the Coherence `bin` directory, and can be used to execute this test.

Note: before Coherence 3.1 the following syntax was used, and scripts were not provided:

```
java com.tangosol.net.MulticastTest <ip-addr> <multicast-addr> <port> <ttl>
<delay-secs>
```

Multicast Test Example

Presume that you want to test if you can use multicast address 237.0.0.1, port 9000 (the test's defaults) to send messages between two servers: `Server A` with IP address 195.0.0.1 and `Server B` with IP address 195.0.0.2.

Starting with `Server A`, let's determine if it has multicast address 237.0.0.1 port 9000 available for 195.0.0.1 by first checking the machine or interface by itself as follows:

From a command prompt, enter the following command:

Example 16–1 Command to Determine a Multicast Address

```
multicast-test.sh -ttl 0
```

After pressing ENTER, you should see the Multicast Test utility display how it is sending sequential multicast packets and receiving them. Example 16–2 illustrates sample output.

Example 16–2 Sequential Multicast Packets Sent by the Multicast Test Utility

```
Starting test on ip=servera/195.0.0.1, group=/237.0.0.1:9000,ttl=0
Configuring multicast socket...
Starting listener...
Tue Mar 17 15:59:51 EST 2008: Sent packet 1.
Tue Mar 17 15:59:51 EST 2008: Received test packet 1 from self.
Tue Mar 17 15:59:53 EST 2008: Sent packet 2.
Tue Mar 17 15:59:53 EST 2008: Received test packet 2 from self.
...
```

When you have seen several these packets sent and received successfully, you can press CTRL-C to stop further testing.

If you do not see something similar to the above, then multicast is not working. Also, please note that we specified a TTL of 0 to prevent the multicast packets from leaving `Server A`.

You can repeat the same test on `Server B` to assure that it too has the multicast enabled for it's port combination.

Now to test multicast communications between `Server A` and `Server B`. For this test we will use a nonzero TTL which will allow the packets to leave their respective servers. By default the test will use a TTL of 4, if you believe that there may be more network hops required to route packets between `Server A` and `Server B`, you may specify a higher TTL value.

Start the test on `Server A` and `Server B` by entering the following command into the command windows and pressing ENTER:

```
multicast-test.sh
```

You should see something like the following on `Server A`:

Example 16–3 Sample Multicast Test Results from Server A

```
Starting test on ip=servera/195.0.0.1, group=/237.0.0.1:9000, ttl=4
Configuring multicast socket...
Starting listener...
Tue Mar 17 16:11:03 EST 2008: Sent packet 1.
Tue Mar 17 16:11:03 EST 2008: Received test packet 1 from self.
Tue Mar 17 16:11:05 EST 2008: Sent packet 2.
Tue Mar 17 16:11:05 EST 2008: Received test packet 2 from self.
```

```
Tue Mar 17 16:11:07 EST 2008: Sent packet 3.
Tue Mar 17 16:11:07 EST 2008: Received test packet 3 from self.
Tue Mar 17 16:11:09 EST 2008: Sent packet 4.
Tue Mar 17 16:11:09 EST 2008: Received test packet 4 from self.
Tue Mar 17 16:11:10 EST 2008: Received test packet 1 from ip=serverb/195.0.0.2,
group=/237.0.0.1:9000, ttl=4.
Tue Mar 17 16:11:11 EST 2008: Sent packet 5.
Tue Mar 17 16:11:11 EST 2008: Received test packet 5 from self.
Tue Mar 17 16:11:12 EST 2008: Received test packet 2 from ip=serverb/195.0.0.2,
group=/237.0.0.1:9000, ttl=4.
Tue Mar 17 16:11:13 EST 2008: Sent packet 6.
Tue Mar 17 16:11:13 EST 2008: Received test packet 6 from self.
Tue Mar 17 16:11:14 EST 2008: Received test packet 3 from ip=serverb/195.0.0.2,
group=/237.0.0.1:9000, ttl=4.
Tue Mar 17 16:11:15 EST 2008: Sent packet 7.
Tue Mar 17 16:11:15 EST 2008: Received test packet 7 from self.
...
```

and something like the following on `Server B`:

Example 16–4 Sample Multicast Test Results on Server B

```
Starting test on ip=serverb/195.0.0.2, group=/237.0.0.1:9000, ttl=4
Configuring multicast socket...
Starting listener...
Tue Mar 17 16:11:10 EST 2008: Sent packet 1.
Tue Mar 17 16:11:10 EST 2008: Received test packet 1 from self.
Tue Mar 17 16:11:11 EST 2008: Received test packet 5 from ip=servera/195.0.0.1,
group=/237.0.0.1:9000, ttl=4.
Tue Mar 17 16:11:12 EST 2008: Sent packet 2.
Tue Mar 17 16:11:12 EST 2008: Received test packet 2 from self.
Tue Mar 17 16:11:13 EST 2008: Received test packet 6 from ip=servera/195.0.0.1,
group=/237.0.0.1:9000, ttl=4.
Tue Mar 17 16:11:14 EST 2008: Sent packet 3.
Tue Mar 17 16:11:14 EST 2008: Received test packet 3 from self.
Tue Mar 17 16:11:15 EST 2008: Received test packet 7 from ip=falco/192.168.0.204,
group=/237.0.0.1:9000, ttl=4.
...
```

You can see that both `Server A` and `Server B` are issuing multicast packets and seeing their own and each other's packets. This indicates that multicast is functioning properly between these servers using the default multicast address and port.

Note: `Server A` sees only its own packets 1-4 until we start `Server B` and it receives packet 1 from `Server B`.

Troubleshooting Multicast Communications

If you are unable to establish bidirectional multicast communication please try the following:

- Firewalls—If any of the machines running the multicast test employ firewalls, the firewall may be blocking the traffic. Consult your OS/firewall documentation for details on allowing multicast traffic.
- Switches—Ensure that your switches are configured to forward multicast traffic.
- IPv6—On OSs which support IPv6 Java may be attempting to route the Multicast traffic over IPv6 rather then IPv4. Try specifying the following Java system property to force IPv4 networking `java.net.preferIPv4Stack=true`.

- Received ???—If the test reports receiving "???" this is an indication that it is receiving multicast packets which did not originate from an instance of the Multicast test. This will occur if you run the test with the same multicast address as a running Coherence cluster, or any other multicast application.
- Multiple NICs—If your machines have multiple network interfaces you may try specifying an explicit interface by using the `-local test` parameter. For instance if `Server A` has two interfaces with IP addresses 195.0.0.1 and 195.0.100.1, including `-local 195.0.0.1` on the test command line would ensure that the multicast packets used the first interface. You may also need to explicitly set your machines routing table to forward multicast traffic through the desired network interface. This can be done by issuing the command in Example 16–5:

Example 16–5 Command to Set Machine Routing Table

```
route add -net 224.0.0.0 netmask 240.0.0.0 dev eth1
```

  Where `eth1` is the device which will be designated to transmit multicast traffic.
- AIX—On AIX systems you may run into the following multicast issues:
  - IPv6—In addition to specifying `java.net.preferIPv4Stack=true` you may need to configure the OS to perform IPv4 name resolution by adding `hosts=local,bind4` to your `/etc/netsvc.conf` file.
  - Virtual IP (VIPA)—AIX does not support multicast with VIPA. If using VIPA either bind multicast to a non-VIPA device, or run Coherence with multicast disabled. See "well-known-addresses" on page H-51 for details.
  - MTU—Configure the MTU for the multicast device to 1500 bytes.
- Cisco Switches—See "Deploying to Cisco Switches" on page M-2 for the list of known issues.
- Foundry Switches—See "Deploying to Foundry Switches" on page M-4 for the list of known issues.

If multicast is not functioning properly, you will need to consult with your network administrator or sysadmin to determine the cause and to correct the situation.

17

Performing a Datagram Test for Network Performance

Included with Coherence is a Datagram Test utility which can be used to test and tune network performance between two or more machines. The Datagram test operates in one of three modes, either as a packet publisher, a packet listener, or both. When run a publisher will transmit UDP packets to the listener who will measure the throughput, success rate, and other statistics.

To achieve maximum performance it is suggested that you tune your environment based on the results of these tests. See Chapter 20, "Performance Tuning" for more information.

Running the Datagram Test Utility

The Datagram test supports a large number of configuration options, though only a few are required for basic operation. To run the Datagram Test utility use the following syntax from the command line:

```
java com.tangosol.net.DatagramTest <command value ...> <addr:port ...>
```

Table 17–1 describes the available command line options for the Datagram Test utility.

Table 17–1 Command Line Options for the Datagram Test Utility

Command	Required/ Optional	Applicability	Description	Default
-local	Optional	Both	The local address to bind to, specified as `addr:port`	localhost:9999
-packetSize	Optional	Both	The size of packet to work with, specified in bytes.	1468
-processBytes	Optional	Both	The number of bytes (in multiples of 4) of each packet to process.	4
-rxBufferSize	Optional	Listener	The size of the receive buffer, specified in packets.	1428
-txBufferSize	Optional	Publisher	The size of the transmit buffer, specified in packets.	16
-txRate	Optional	Publisher	The rate at which to transmit data, specified in megabytes.	*unlimited*
-txIterations	Optional	Publisher	Specifies the number of packets to publish before exiting.	*unlimited*
-txDurationMs	Optional	Publisher	Specifies how long to publish before exiting.	*unlimited*

Table 17–1 (Cont.) Command Line Options for the Datagram Test Utility

Command	Required/ Optional	Applicability	Description	Default
-reportInterval	Optional	Both	The interval at which to output a report, specified in packets.	100000
-tickInterval	Optional	Both	The interval at which to output tick marks.	1000
-log	Optional	Listener	The name of a file to save a tabular report of measured performance.	*none*
-logInterval	Optional	Listener	The interval at which to output a measurement to the log.	100000
-polite	Optional	Publisher	Switch indicating if the publisher should wait for the listener to be contacted before publishing.	*off*
arguments	Optional	Publisher	Space separated list of addresses to publish to, specified as `addr:port`.	*none*

Sample Commands for a Listener and a Publisher

The following command line is for a listener:

```
java -server com.tangosol.net.DatagramTest -local box1:9999 -packetSize 1468
```

The following command line is for a publisher:

```
java -server com.tangosol.net.DatagramTest -local box2:9999 -packetSize 1468
box1:9999
```

For ease of use, `datagram-test.sh` and `datagram-test.cmd` scripts are provided in the Coherence `bin` directory, and can be used to execute this test.

Datagram Test Example

Presume that you want to test network performance between two servers— `Server A` with IP address `1{{95.0.0.1}}` and `Server B` with IP address `195.0.0.2`. One server will act as a packet publisher and the other as a packet listener, the publisher will transmit packets as fast as possible and the listener will measure and report performance statistics. First start the listener on `Server A`.

Example 17–1 Command to Start a Listener

```
datagram-test.sh
```

After pressing ENTER, you should see the Datagram Test utility showing you that it is ready to receive packets.

Example 17–2 Output from Starting a Listener

```
starting listener: at /195.0.0.1:9999
packet size: 1468 bytes
buffer size: 1428 packets
  report on: 100000 packets, 139 MBs
    process: 4 bytes/packet
        log: null
     log on: 139 MBs
```

As you can see by default the test will try to allocate a network receive buffer large enough to hold 1428 packets, or about 2 MB. If it is unable to allocate this buffer it will

report an error and exit. You can either decrease the requested buffer size using the `-rxBufferSize` parameter or increase your operating system network buffer settings. For best performance it is recommended that you increase the operating system buffers. See the following forum post for details on tuning your operating system for Coherence.

When the listener process is running you may start the publisher on `Server B`, directing it to publish to `Server A`.

Example 17–3 Command to Start a Publisher

```
datagram-test.sh servera
```

After pressing ENTER, you should see the new Datagram test instance on `Server B` start both a listener and a publisher. Note in this configuration `Server B` listener will not be used. The output illustrates in Example 17–4 should appear in the `Server B` command window.

Example 17–4 Datagram Test—Starting a Listener and a Publisher on a Server

```
starting listener: at /195.0.0.2:9999
packet size: 1468 bytes
buffer size: 1428 packets
  report on: 100000 packets, 139 MBs
    process: 4 bytes/packet
        log: null
     log on: 139 MBs

starting publisher: at /195.0.0.2:9999 sending to servera/195.0.0.1:9999
packet size: 1468 bytes
buffer size: 16 packets
  report on: 100000 packets, 139 MBs
    process: 4 bytes/packet
      peers: 1
       rate: no limit

no packet burst limit
oooooooooOoooooooooOoooooooooOoooooooooOoooooooooOoooooooooOoooooooooOoooooooooO
```

The series of `o` and `O` tick marks appear as data is (O)utput on the network. Each `o` represents 1000 packets, with `O` indicators at every 10,000 packets.

On `Server A` you should see a corresponding set of `i` and `I` tick marks, representing network (I)nput. This indicates that the two test instances are communicating.

Reporting

Periodically, each side of the test (publisher and listener) will report performance statistics.

Publisher Statistics

The publisher simply reports the rate at which it is publishing data on the network. A typical report is as follows:

Example 17–5 Sample Publisher Report

```
Tx summary 1 peers:
   life: 97 MB/sec, 69642 packets/sec
```

```
    now: 98 MB/sec, 69735 packets/sec
```

The report includes both the current transmit rate (since last report) and the lifetime transmit rate.

Listener Statistics

Table 17–2 describes the statistics that can be reported by the listener.

Table 17–2 Listener Statistics

Element	Description
Elapsed	The time interval that the report covers.
Packet size	The received packet size.
Throughput	The rate at which packets are being received.
Received	The number of packets received.
Missing	The number of packets which were detected as lost.
Success rate	The percentage of received packets out of the total packets sent.
Out of order	The number of packets which arrived out of order.
Average offset	An indicator of how out of order packets are.

As with the publisher both current and lifetime statistics are report. Example 17–6 displays a typical report:

Example 17–6 Sample Lifetime Statistics

```
Lifetime:
Rx from publisher: /195.0.0.2:9999
             elapsed: 8770ms
         packet size: 1468
          throughput: 96 MB/sec
                      68415 packets/sec
            received: 600000 of 611400
             missing: 11400
        success rate: 0.9813543
        out of order: 2
          avg offset: 1


Now:
Rx from publisher: /195.0.0.2:9999
             elapsed: 1431ms
         packet size: 1468
          throughput: 98 MB/sec
                      69881 packets/sec
            received: 100000 of 100000
             missing: 0
        success rate: 1.0
        out of order: 0
          avg offset: 0
```

The primary items of interest are the throughput and success rate. The goal is to find the highest throughput while maintaining a success rate as close to 1.0 as possible. On a 100 Mb network setup you should be able to achieve rates of around 10 MB/sec. On

a 1 Gb network you should be able to achieve rates of around 100 MB/sec. Achieving these rates will likely require some tuning (see below).

Throttling

The publishing side of the test may be throttled to a specific data rate expressed in megabytes per second, by including the `-txRate M` parameter when `M` represents the maximum MB/sec the test should put on the network.

Bidirectional Testing

You may also run the test in a bidirectional mode where both servers act as publishers and listeners. To do this simply restart test instances, supplying the instance on `Server A` with `Server B's` address, by running the command in Example 17–7 on `Server A`.

Example 17–7 Running Datagram Test in Bi-Directional Mode

```
datagram-test.sh -polite serverb
```

And then run the same command as before on `Server B`. The `-polite` parameter instructs this test instance to not start publishing until it is starts to receive data.

Distributed Testing

You may also use more then two machines in testing, for instance you can setup two publishers to target a single listener. This style testing is far more realistic then simple one-to-one testing, and may identify bottlenecks in your network which you were not otherwise aware of.

Assuming you intend to construct a cluster consisting of four machines, you can run the datagram test among all of them as follows:

On `Servera`:

```
datagramtest.sh -txRate 100 -polite serverb serverc serverd
```

On `Serverb`:

```
datagramtest.sh -txRate 100 -polite servera serverc serverd
```

On `Serverc`:

```
datagramtest.sh -txRate 100 -polite servera serverb serverd
```

On `Serverd`:

```
datagramtest.sh -txRate 100 servera serverb serverc
```

This test sequence will cause all nodes to send a total of 100MB per second to all other nodes (that is, 33MB/node/sec). On a fully switched network 1GbE network this should be achievable without packet loss.

To simplify the execution of the test all nodes can be started with an identical target list, they will obviously transmit to themselves as well, but this loopback data can easily be factored out. It is important to start all but the last node using the `-polite` switch, as this will cause all other nodes to delay testing until the final node is started.

18

Configuring and Using Coherence*Extend

Coherence*Extend extends the reach of the core Coherence TCMP cluster to a wider range of consumers, including desktops, remote servers and machines located across WAN connections. Typical uses of Coherence*Extend include providing desktop applications with access to Coherence caches (including support for Near Cache and continuous query) and Coherence cluster "bridges" that link together multiple Coherence clusters connected by using a high-latency, unreliable WAN.

Coherence*Extend consists of two basic components: a client and a Coherence*Extend clustered service hosted by one or more `DefaultCacheServer` processes. The adapter library includes implementations of both the `CacheService` and `InvocationService` interfaces that route all requests to a Coherence*Extend clustered service instance running within the Coherence cluster. The Coherence*Extend clustered service in turn responds to client requests by delegating to an actual Coherence clustered service (for example, a Partitioned or Replicated cache service). The client adapter library and Coherence*Extend clustered service use a low-level messaging protocol to communicate with each other. Coherence*Extend includes the following transport bindings for this protocol:

- **Extend-JMS**—uses your existing JMS infrastructure as the means to connect to the cluster
- **Extend-TCP**—uses a high performance, scalable TCP/IP-based communication layer to connect to the cluster

The choice of a transport binding is configuration-driven and is completely transparent to the client application that uses Coherence*Extend. A Coherence*Extend service is retrieved like a Coherence clustered service: by using the `CacheFactory` class. When obtained, a client uses the Coherence*Extend service in the same way as it would if it were part of the Coherence cluster. The fact that operations are being sent to a remote cluster node (over either JMS or TCP) is transparent to the client application.

General Instructions

Configuring and using Coherence*Extend requires four basic steps:

1. Create a client-side Coherence cache configuration descriptor that includes one or more <remote-cache-scheme> and <remote-invocation-scheme> configuration elements
2. Create a cluster-side Coherence cache configuration descriptor that includes one or more <proxy-scheme> configuration elements
3. Launch one or more `DefaultCacheServer` processes

4. Create a client application that uses one or more Coherence*Extend services. See "Sample Coherence*Extend Client Application" on page 18-12.
5. Launch the client application

The following sections describe each of these steps in detail for the Extend-JMS and Extend-TCP transport bindings.

- Configuring and Using Coherence*Extend-JMS
- Configuring and Using Coherence*Extend-TCP

Configuring and Using Coherence*Extend-JMS

Client-side Cache Configuration Descriptor

A Coherence*Extend client that uses the Extend-JMS transport binding must define a Coherence cache configuration descriptor which includes a `<remote-cache-scheme>` and/or `<remote-invocation-scheme>` element with a child `<jms-initiator>` element containing various JMS-specific configuration information. Example 18–1 illustrates a sample descriptor.

Example 18–1 Client-Side Cache Configuration Descriptor for Extend-JMS

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>dist-extend</cache-name>
      <scheme-name>extend-dist</scheme-name>
    </cache-mapping>

    <cache-mapping>
      <cache-name>dist-extend-near</cache-name>
      <scheme-name>extend-near</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <near-scheme>
      <scheme-name>extend-near</scheme-name>
      <front-scheme>
        <local-scheme>
          <high-units>1000</high-units>
        </local-scheme>
      </front-scheme>
      <back-scheme>
        <remote-cache-scheme>
          <scheme-ref>extend-dist</scheme-ref>
        </remote-cache-scheme>
      </back-scheme>
      <invalidation-strategy>all</invalidation-strategy>
    </near-scheme>

    <remote-cache-scheme>
      <scheme-name>extend-dist</scheme-name>
```

```
      <service-name>ExtendJmsCacheService</service-name>
      <initiator-config>
        <jms-initiator>

<queue-connection-factory-name>jms/coherence/ConnectionFactory</queue-connection-f
actory-name>
          <queue-name>jms/coherence/Queue</queue-name>
          <connect-timeout>10s</connect-timeout>
        </jms-initiator>
        <outgoing-message-handler>
          <request-timeout>5s</request-timeout>
        </outgoing-message-handler>
      </initiator-config>
    </remote-cache-scheme>

    <remote-cache-scheme>
      <scheme-name>extend-invocation</scheme-name>
      <service-name>ExtendJmsInvocationService</service-name>
      <initiator-config>
        <jms-initiator>

<queue-connection-factory-name>jms/coherence/ConnectionFactory</queue-connection-f
actory-name>
          <queue-name>jms/coherence/Queue</queue-name>
          <connect-timeout>10s</connect-timeout>
        </jms-initiator>
        <outgoing-message-handler>
          <request-timeout>5s</request-timeout>
        </outgoing-message-handler>
      </initiator-config>
    </remote-cache-scheme>
  </caching-schemes>
</cache-config>
```

This cache configuration descriptor defines two caching schemes, one that uses Extend-JMS to connect to a remote Coherence cluster (`<remote-cache-scheme>`) and one that maintains an in-process size-limited near cache of remote Coherence caches (again, accessed by Extend-JMS). Additionally, the cache configuration descriptor defines a `<remote-invocation-scheme>` that allows the client application to execute tasks within the remote Coherence cluster. Both the `<remote-cache-scheme>` and `<remote-invocation-scheme>` elements have a `<jms-initiator>` child element which includes all JMS-specific information needed to connect the client with the Coherence*Extend clustered service running within the remote Coherence cluster.

When the client application retrieves a `NamedCache` by using the `CacheFactory` using, for example, the name `dist-extend`, the Coherence*Extend adapter library will connect to the Coherence cluster by using a JMS Queue (retrieved by JNDI using the name `jms/coherence/Queue`") and return a `NamedCache` implementation that routes requests to the `NamedCache` with the *same name* running within the remote cluster. Likewise, when the client application retrieves a InvocationService by calling `CacheFactory.getConfigurableCacheFactory().ensureService("Extend JmsInvocationService")`, the Coherence*Extend adapter library will connect to the Coherence cluster by using the same JMS Queue and return an InvocationService implementation that executes synchronous Invocable tasks within the remote clustered JVM to which the client is connected.

Cluster-side Cache Configuration Descriptor

For a **Coherence*Extend-JMS** client to connect to a Coherence cluster, one or more `DefaultCacheServer` processes must be running that use a Coherence cache configuration descriptor. This desciptor must include a `<proxy-scheme>` element with a child `<jms-acceptor>` element containing various JMS-specific configuration information. Example 18–2 illustrates a sample descriptor.

Example 18–2 Cluster-Side Cache Configuration Descriptor for Extend-JMS

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>dist-*</cache-name>
      <scheme-name>dist-default</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <distributed-scheme>
      <scheme-name>dist-default</scheme-name>
      <lease-granularity>member</lease-granularity>
      <backing-map-scheme>
        <local-scheme/>
      </backing-map-scheme>
      <autostart>true</autostart>
    </distributed-scheme>

    <proxy-scheme>
      <service-name>ExtendJmsProxyService</service-name>
      <acceptor-config>
        <jms-acceptor>

<queue-connection-factory-name>jms/coherence/ConnectionFactory</queue-connection-f
actory-name>
          <queue-name>jms/coherence/Queue</queue-name>
        </jms-acceptor>
      </acceptor-config>
      <autostart>true</autostart>
    </proxy-scheme>
  </caching-schemes>
</cache-config>
```

This cache configuration descriptor defines two clustered services: one that uses Extend-JMS to allow remote Extend-JMS clients to connect to the Coherence cluster and a standard Partitioned cache service. Since this descriptor is used by a `DefaultCacheServer` it is important that the `<autostart>` configuration element for each service is set to `true` so that clustered services are automatically restarted upon termination. The `<proxy-scheme>` element has a `<jms-acceptor>` child element which includes all JMS-specific information needed to accept client connection requests over JMS.

The Coherence*Extend clustered service will listen to a JMS Queue (retrieved by JNDI using the name `jms/coherence/Queue`) for connection requests. When, for example, a client attempts to connect to a Coherence NamedCache called `dist-extend`, the Coherence*Extend clustered service will proxy subsequent requests to the NamedCache with the *same name* which, in this case, will be a

Partitioned cache. Note that Extend-JMS client connection requests will be load balanced across all `DefaultCacheServer` processes that run a Coherence*Extend clustered service with the same configuration.

Configuring your JMS Provider

Coherence*Extend-JMS uses JNDI to obtain references to all JMS resources. To specify the JNDI properties that Coherence*Extend-JMS uses to create a JNDI `InitialContext`, create a file called jndi.properties that contains your JMS provider's configuration properties and add the *directory* that contains the file to both the client application and `DefaultCacheServer` classpaths.

For example, if you are using WebLogic Server as your JMS provider, your `jndi.properties` file would look like Example 18–3:

Example 18–3 jndi.properties Values for a WebLogic Server Acting as a JMS Provider

```
java.naming.factory.initial=weblogic.jndi.WLInitialContextFactory
java.naming.provider.url=t3://localhost:7001
java.naming.security.principal=system
java.naming.security.credentials=weblogic
```

Additionally, Coherence*Extend-JMS uses a JMS Queue to connect Extend-JMS clients to a Coherence*Extend clustered service instance. Therefore, you must deploy an appropriately configured JMS `QueueConnectionFactory` and `Queue` and register them under the JNDI names specified in the `<jms-initiator>` and `<jms-acceptor>` configuration elements.

For example, if you are using WebLogic Server, you can use the Ant script in Example 18–4 to create and deploy these JMS resources:

Example 18–4 Ant Script to Create JMS Resources and Deploy on a WebLogic Server

```
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- Ant build script for configuring a WebLogic Server domain with the    -->
<!-- necessary JMS resources required by Coherence*Extend-JMS              -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!--                                                                       -->
<!-- Usage:                                                                -->
<!--                                                                       -->
<!--   1) Create the WLS domain:                                           -->
<!--      prompt> ant create.domain                                        -->
<!--                                                                       -->
<!--   2) Start the WLS instance:                                          -->
<!--      prompt> domain/startmydomain.cmd|sh                              -->
<!--                                                                       -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<project name="extend-jms-wls" default="create.domain" basedir=".">

  <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
  <!-- Project properties                                                  -->
  <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->

  <property name="weblogic.home"   value="c:/opt/bea/weblogic8.1.5"/>
  <property name="weblogic.jar"
value="${weblogic.home}/server/lib/weblogic.jar"/>
  <property name="server.user"     value="system"/>
  <property name="server.password" value="weblogic"/>
  <property name="domain.dir"      value="domain"/>
  <property name="domain.name"     value="mydomain"/>
```

```
<property name="server.name"     value="myserver"/>
<property name="realm.name"      value="myrealm"/>
<property name="server.host"     value="localhost"/>
<property name="server.port"     value="7001"/>
<property name="admin.url"       value="t3://${server.host}:${server.port}"/>

<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- Project paths                                                         -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->

<path id="project.classpath">
  <pathelement location="${weblogic.jar}"/>
</path>

<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- Project task definitions                                              -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->

<taskdef name="wlserver"
  classname="weblogic.ant.taskdefs.management.WLServer"
  classpathref="project.classpath"/>
<taskdef name="wlconfig"
  classname="weblogic.ant.taskdefs.management.WLConfig"
  classpathref="project.classpath"/>

<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- Project targets                                                       -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->

<target name="clean" description="Remove all build artifacts.">
  <delete dir="${domain.dir}"/>
</target>

<target name="create.domain"
  description="Create a WLS domain for use with Coherence*Extend-JMS.">
  <delete dir="${domain.dir}"/>
  <mkdir dir="${domain.dir}"/>

  <wlserver weblogicHome="${weblogic.home}"
    dir="${domain.dir}"
    classpathref="project.classpath"
    host="${server.host}"
    port="${server.port}"
    servername="${server.name}"
    domainname="${domain.name}"
    generateConfig="true"
    username="${server.user}"
    password="${server.password}"
    action="start"/>

  <antcall target="config.domain"/>
</target>

<target name="config.domain"
  description="Configure a WLS domain for use with Coherence*Extend-JMS.">
    <wlconfig url="${admin.url}"
      username="${server.user}"
      password="${server.password}">
      <query domain="${domain.name}"
        type="Server"
```

```
            name="${server.name}"
            property="server"/>

          <!-- Create a JMS template -->
          <create type="JMSTemplate" name="CoherenceTemplate" property="template"/>

          <!-- Add a JMS server and queue for the application -->
          <create type="JMSServer" name="MyJMSServer">
              <set attribute="Targets" value="${server}"/>
              <create type="JMSQueue" name="CoherenceQueue">
                  <set attribute="JNDIName" value="jms/coherence/Queue"/>
              </create>
              <set attribute="TemporaryTemplate" value="${template}"/>
          </create>

          <!-- Create a JMS connection factory -->
          <create type="JMSConnectionFactory" name="CoherenceConnectionFactory">
              <set attribute="JNDIName"
                  value="jms/coherence/ConnectionFactory"/>
              <set attribute="Targets" value="${server}"/>
          </create>
        </wlconfig>
    </target>
</project>
```

Launching an Extend-JMS DefaultCacheServer Process

To start a `DefaultCacheServer` that uses the cluster-side Coherence cache configuration described earlier to allow Extend-JMS clients to connect to the Coherence cluster by using JMS, you need to do the following:

- Change the current directory to the Coherence library directory (*%COHERENCE_HOME%*`\lib` on Windows and *$COHERENCE_HOME*`/lib` on UNIX).
- Make sure that the paths are configured so that the Java command will run.
- Start the `DefaultCacheServer` command line application with the directory that contains your `jndi.properties` file and your JMS provider's libraries on the classpath and the `-Dtangosol.coherence.cacheconfig` system property set to the location of the cluster-side Coherence cache configuration descriptor described earlier.

For example, if you are using WebLogic server as your JMS provider, run the following command on Windows (note that it is broken up into multiple lines here only for formatting purposes; this is a single command typed on one line):

Example 18–5 Windows Command to Start the Default Cache Server for the Cluster-Side

```
java -cp coherence.jar;<directory containing jndi.properties>;<WebLogic
home>\server\lib\wljmsclient.jar
     -Dtangosol.coherence.cacheconfig=file://<path to the server-side cache
configuration descriptor>
     com.tangosol.net.DefaultCacheServer
```

On UNIX:

Example 18–6 UNIX Command to Start the Default Cache Server for the Cluster-Side

```
java -cp coherence.jar:<directory containing jndi.properties>:<WebLogic
home>/server/lib/wljmsclient.jar
```

```
     -Dtangosol.coherence.cacheconfig=file://<path to the server-side cache
configuration descriptor>
     com.tangosol.net.DefaultCacheServer
```

Launching an Extend-JMS Client Application

To start a client application that uses Extend-JMS to connect to a remote Coherence cluster by using JMS, you need to do the following:

- Change the current directory to the Coherence library directory (*%COHERENCE_HOME%*`\lib` on Windows and *$COHERENCE_HOME*`/lib` on UNIX).
- Make sure that the paths are configured so that the Java command will run.
- Start your client application with the directory that contains your `jndi.properties` file and your JMS provider's libraries on the classpath and the `-Dtangosol.coherence.cacheconfig` system property set to the location of the client-side Coherence cache configuration descriptor described earlier.

For example, if you are using WebLogic server as your JMS provider, you would run the following command on Windows (note that it is broken up into multiple lines here only for formatting purposes; this is a single command typed on one line):

Example 18–7 Windows Command to Start the Client Application

```
java -cp coherence.jar;<directory containing jndi.properties>;<WebLogic
home>\server\lib\wljmsclient.jar
     -Dtangosol.coherence.cacheconfig=file://<path to the client-side cache
configuration descriptor>
     <client application Class name>
```

On UNIX:

Example 18–8 UNIX Command to Start the Client Application

```
java -cp coherence.jar:<directory containing jndi.properties>:<WebLogic
home>/server/lib/wljmsclient.jar
     -Dtangosol.coherence.cacheconfig=file://<path to the client-side cache
configuration descriptor>
     <client application Class name>
```

Configuring and Using Coherence*Extend-TCP

Client-side Cache Configuration Descriptor

A Coherence*Extend client that uses the Extend-TCP transport binding must define a Coherence cache configuration descriptor which includes a `<remote-cache-scheme>` and/or `<remote-invocation-scheme>` element with a child `<tcp-initiator>` element containing various TCP/IP-specific configuration information. Example 18–9 illustrates a sample descriptor.

Example 18–9 Coherence*Extend Client Descriptor that uses Extend-TCP

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">

<cache-config>
```

```
<caching-scheme-mapping>
  <cache-mapping>
    <cache-name>dist-extend</cache-name>
    <scheme-name>extend-dist</scheme-name>
  </cache-mapping>

  <cache-mapping>
    <cache-name>dist-extend-near</cache-name>
    <scheme-name>extend-near</scheme-name>
  </cache-mapping>
</caching-scheme-mapping>

<caching-schemes>
  <near-scheme>
    <scheme-name>extend-near</scheme-name>
    <front-scheme>
      <local-scheme>
        <high-units>1000</high-units>
      </local-scheme>
    </front-scheme>
    <back-scheme>
      <remote-cache-scheme>
        <scheme-ref>extend-dist</scheme-ref>
      </remote-cache-scheme>
    </back-scheme>
    <invalidation-strategy>all</invalidation-strategy>
  </near-scheme>

  <remote-cache-scheme>
    <scheme-name>extend-dist</scheme-name>
    <service-name>ExtendTcpCacheService</service-name>
    <initiator-config>
      <tcp-initiator>
        <remote-addresses>
          <socket-address>
            <address>localhost</address>
            <port>9099</port>
          </socket-address>
        </remote-addresses>
        <connect-timeout>10s</connect-timeout>
      </tcp-initiator>
      <outgoing-message-handler>
        <request-timeout>5s</request-timeout>
      </outgoing-message-handler>
    </initiator-config>
  </remote-cache-scheme>

  <remote-invocation-scheme>
    <scheme-name>extend-invocation</scheme-name>
    <service-name>ExtendTcpInvocationService</service-name>
    <initiator-config>
      <tcp-initiator>
        <remote-addresses>
          <socket-address>
            <address>localhost</address>
            <port>9099</port>
          </socket-address>
        </remote-addresses>
        <connect-timeout>10s</connect-timeout>
      </tcp-initiator>
```

```
        <outgoing-message-handler>
          <request-timeout>5s</request-timeout>
        </outgoing-message-handler>
      </initiator-config>
    </remote-invocation-scheme>
  </caching-schemes>
</cache-config>
```

This cache configuration descriptor defines two caching schemes, one that uses Extend-TCP to connect to a remote Coherence cluster (`<remote-cache-scheme>`) and one that maintains an in-process size-limited near cache of remote Coherence caches (again, accessed by using Extend-TCP). Additionally, the cache configuration descriptor defines a `<remote-invocation-scheme>` that allows the client application to execute tasks within the remote Coherence cluster. Both the `<remote-cache-scheme>` and `<remote-invocation-scheme>` elements have a `<tcp-initiator>` child element which includes all TCP/IP-specific information needed to connect the client with the Coherence*Extend clustered service running within the remote Coherence cluster.

When the client application retrieves a `NamedCache` by using the `CacheFactory` using, for example, the name `dist-extend`, the Coherence*Extend adapter library will connect to the Coherence cluster by using TCP/IP (using the address `localhost` and `port 9099`) and return a `NamedCache` implementation that routes requests to the NamedCache with the *same name* running within the remote cluster. Likewise, when the client application retrieves a `InvocationService` by calling `CacheFactory.getConfigurableCacheFactory().ensureService("Extend TcpInvocationService")`, the Coherence*Extend adapter library will connect to the Coherence cluster by using TCP/IP (again, using the address `localhost` and `port 9099`) and return an InvocationService implementation that executes synchronous Invocable tasks within the remote clustered JVM to which the client is connected.

Note that the `<remote-addresses>` configuration element can contain multiple `<socket-address>` child elements. The Coherence*Extend adapter library will attempt to connect to the addresses in a random order, until either the list is exhausted or a TCP/IP connection is established.

Cluster-side Cache (a.k.a Coherence Extend Proxy) Configuration Descriptor

For a **Coherence*Extend-TCP** client to connect to a Coherence cluster, one or more `DefaultCacheServer` processes must be running that use a Coherence cache configuration descriptor. This descriptor must include a `<proxy-scheme>` element with a child `<tcp-acceptor>` element containing various TCP/IP-specific configuration information. Example 18–10 illustrates a sample descriptor.

Example 18–10 Cluster-Side Cache Configuration Descriptor for Extend-TCP

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>dist-*</cache-name>
      <scheme-name>dist-default</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>
```

```
  <caching-schemes>
    <distributed-scheme>
      <scheme-name>dist-default</scheme-name>
      <lease-granularity>member</lease-granularity>
      <backing-map-scheme>
        <local-scheme/>
      </backing-map-scheme>
      <autostart>true</autostart>
    </distributed-scheme>

    <proxy-scheme>
      <service-name>ExtendTcpProxyService</service-name>
      <thread-count>5</thread-count>
      <acceptor-config>
        <tcp-acceptor>
          <local-address>
            <address>localhost</address>
            <port>9099</port>
          </local-address>
        </tcp-acceptor>
      </acceptor-config>
      <autostart>true</autostart>
    </proxy-scheme>
  </caching-schemes>
</cache-config>
```

This cache configuration descriptor defines two clustered services, one that uses Extend-TCP to allow remote Extend-TCP clients to connect to the Coherence cluster and a standard Partitioned cache service. Since this descriptor is used by a `DefaultCacheServer` it is important that the `<autostart>` configuration element for each service is set to true so that clustered services are automatically restarted upon termination. The `<proxy-scheme>` element has a `<tcp-acceptor>` child element which includes all TCP/IP-specific information needed to accept client connection requests over TCP/IP.

The Coherence*Extend clustered service will listen to a TCP/IP ServerSocket (bound to address `localhost` and `port 9099`) for connection requests. When, for example, a client attempts to connect to a Coherence `NamedCache` called `dist-extend-direct`, the Coherence*Extend clustered service will proxy subsequent requests to the `NamedCache` with the *same name* which, in this case, will be a Partitioned cache.

Launching an Extend-TCP DefaultCacheServer Process

To start a `DefaultCacheServer` that uses the cluster-side Coherence cache configuration described earlier to allow Extend-TCP clients to connect to the Coherence cluster by using TCP/IP, you need to do the following:

- Change the current directory to the Coherence library directory (*%COHERENCE_HOME%*`\lib` on Windows and *$COHERENCE_HOME*`/lib` on UNIX)
- Make sure that the paths are configured so that the Java command will run
- Start the `DefaultCacheServer` command line application with the `-Dtangosol.coherence.cacheconfig` system property set to the location of the cluster-side Coherence cache configuration descriptor described earlier

For example (note that the following command is broken up into multiple lines here only for formatting purposes; this is a single command typed on one line):

```
java -cp coherence.jar:<classpath to client application>
     -Dtangosol.coherence.cacheconfig=file://<path to the server-side cache
configuration descriptor>
     com.tangosol.net.DefaultCacheServer
```

Launching an Extend-TCP Client Application

To start a client application that uses Extend-TCP to connect to a remote Coherence cluster by using TCP/IP, you need to do the following:

- Change the current directory to the Coherence library directory (*%COHERENCE_HOME%*`\lib` on Windows and *$COHERENCE_HOME*`/lib` on UNIX)
- Make sure that the paths are configured so that the Java command will run
- Start your client application with the `-Dtangosol.coherence.cacheconfig` system property set to the location of the client-side Coherence cache configuration descriptor described earlier

For example (note that the command in Example 18–11 is broken up into multiple lines here only for formatting purposes; this is a single command typed on one line):

Example 18–11 Command to Start a Client Application that Uses Extend-TCP

```
java -cp coherence.jar:<classpath to client application>
     -Dtangosol.coherence.cacheconfig=file://<path to the client-side cache
configuration descriptor>
     <client application Class name>
```

Sample Coherence*Extend Client Application

Example 18–12 demonstrates how to retrieve and use a Coherence*Extend `CacheService` and `InvocationService`. This example increments an `Integer` value in a remote Partitioned cache and then retrieves the value by executing an `Invocable` on the clustered JVM to which the client is attached:

Example 18–12 Sample Coherence*Extend Application

```
public static void main(String[] asArg)
        throws Throwable
    {
    NamedCache cache  = CacheFactory.getCache("dist-extend");
    Integer    IValue = (Integer) cache.get("key");
    if (IValue == null)
        {
        IValue = new Integer(1);
        }
    else
        {
        IValue = new Integer(IValue.intValue() + 1);
        }
    cache.put("key", IValue);

    InvocationService service = (InvocationService)
            CacheFactory.getConfigurableCacheFactory()
                .ensureService("ExtendTcpInvocationService");

    Map map = service.query(new AbstractInvocable()
            {
            public void run()
```

```
                {
                setResult(CacheFactory.getCache("dist-extend").get("key"));
                }
            }, null);

    Integer IValue = (Integer) map.get(service.getCluster().getLocalMember());
    }
```

Note that this example could also be run on a Coherence node (that is, *within* the cluster) verbatim. The fact that operations are being sent to a remote cluster node (over either JMS or TCP) is completely transparent to the client application.

Coherence*Extend InvocationService

Since, by definition, a Coherence*Extend client has no direct knowledge of the cluster and the members running within the cluster, the Coherence*Extend `InvocationService` only allows `Invocable` tasks to be executed on the JVM to which the client is connected. Therefore, you should always pass a null member set to the `query()` method. As a consequence of this, the single result of the execution will be keyed by the local `Member`, which will be null if the client is not part of the cluster. This `Member` can be retrieved by calling `service.getCluster().getLocalMember()`. Additionally, the Coherence*Extend `InvocationService` only supports synchronous task execution (that is, the `execute()` method is not supported).

Advanced Configuration

Network Filters

Like Coherence clustered services, Coherence*Extend services support pluggable network filters. Filters can be used to modify the contents of network traffic before it is placed "on the wire". Most standard Coherence network filters are supported, including the compression and symmetric encryption filters. For more information on configuring filters, see Chapter 8, "Network Filters."

To use network filters with Coherence*Extend, a `<use-filters>` element must be added to the `<initiator-config>` element in the client-side cache configuration descriptor and to the `<acceptor-config>` element in the cluster-side cache configuration descriptor.

For example, to encrypt network traffic exchanged between a Coherence*Extend client and the clustered service to which it is connected, configure the client-side `<remote-cache-scheme>` and `<remote-invocation-scheme>` elements as illustrated in Example 18–13 (assuming the symmetric encryption filter has been named symmetric-encryption):

Example 18–13 Client-Side Configuration to Encrypt Network Traffic

```
<remote-cache-scheme>
  <scheme-name>extend-dist</scheme-name>
  <service-name>ExtendTcpCacheService</service-name>
  <initiator-config>
    <tcp-initiator>
      <remote-addresses>
        <socket-address>
          <address>localhost</address>
```

```
          <port>9099</port>
        </socket-address>
      </remote-addresses>
      <connect-timeout>10s</connect-timeout>
    </tcp-initiator>
    <outgoing-message-handler>
      <request-timeout>5s</request-timeout>
    </outgoing-message-handler>
    <use-filters>
      <filter-name>symmetric-encryption</filter-name>
    </use-filters>
  </initiator-config>
</remote-cache-scheme>

<remote-invocation-scheme>
  <scheme-name>extend-invocation</scheme-name>
  <service-name>ExtendTcpInvocationService</service-name>
  <initiator-config>
    <tcp-initiator>
      <remote-addresses>
        <socket-address>
          <address>localhost</address>
          <port>9099</port>
        </socket-address>
      </remote-addresses>
      <connect-timeout>10s</connect-timeout>
    </tcp-initiator>
    <outgoing-message-handler>
      <request-timeout>5s</request-timeout>
    </outgoing-message-handler>
    <use-filters>
      <filter-name>symmetric-encryption</filter-name>
    </use-filters>
  </initiator-config>
</remote-invocation-scheme>
```

Example 18–14 illustrates the configuration for the cluster-side `<proxy-scheme>` element:

Example 18–14 Cluster-Side Proxy Scheme Configuration

```
<proxy-scheme>
  <service-name>ExtendTcpProxyService</service-name>
  <thread-count>5</thread-count>
  <acceptor-config>
    <tcp-acceptor>
      <local-address>
        <address>localhost</address>
        <port>9099</port>
      </local-address>
    </tcp-acceptor>
    <use-filters>
      <filter-name>symmetric-encryption</filter-name>
    </use-filters>
  </acceptor-config>
  <autostart>true</autostart>
</proxy-scheme>
```

Note: The contents of the <use-filters> element must be the same in the client and cluster-side cache configuration descriptors.

Connection Error Detection and Failover

When a Coherence*Extend service detects that the connection between the client and cluster has been severed (for example, due to a network, software, or hardware failure), the Coherence*Extend client service implementation (that is, CacheService or InvocationService) will dispatch a MemberEvent.MEMBER_LEFT event to all registered MemberListeners and the service will be stopped. If the client application attempts to subsequently use the service, the service will automatically restart itself and attempt to reconnect to the cluster. If the connection is successful, the service will dispatch a MemberEvent.MEMBER_JOINED event; otherwise, a fatal exception will be thrown to the client application.

A Coherence*Extend service has several mechanisms for detecting dropped connections. Some are inherent to the underlying protocol (that is, a javax.jms.ExceptionListener in Extend-JMS and TCP/IP in Extend-TCP), whereas others are implemented by the service itself. The latter mechanisms are configured by using the <outgoing-message-handler> configuration element.

The primary configurable mechanism used by a Coherence*Extend client service to detect dropped connections is a request timeout. When the service sends a request to the remote cluster and does not receive a response within the request timeout interval (see the <request-timeout> subelement of <outgoing-message-handler>), the service assumes that the connection has been dropped. The Coherence*Extend client and clustered services can also be configured to send a periodic heartbeat over the connection (see <heartbeat-interval> and <heartbeat-timeout> subelements of <outgoing-message-handler>). If the service does not receive a response within the configured heartbeat timeout interval, the service assumes that the connection has been dropped.

Notes:

- You should *always* enable heartbeats when using a connectionless transport, as is the case with Extend-JMS.
- If you do not specify a <request-timeout/>, a Coherence*Extend service will use an infinite request timeout. In general, this is not a recommended configuration, as it could result in an unresponsive application. For most use cases, you should specify a reasonable finite request timeout.

Read-only NamedCache Access

By default, the Coherence*Extend clustered service allows both read and write access to proxied NamedCache instances. To prohibit Coherence*Extend clients from modifying cached content, use the <cache-service-proxy> child configuration element. Example 18–15 illustrates a sample configuration.

Example 18–15 Client-Side Configuration to Allow Read-only Access to the Cache

```
<proxy-scheme>
  ...

  <proxy-config>
```

```
    <cache-service-proxy>
      <read-only>true</read-only>
    </cache-service-proxy>
  </proxy-config>

  <autostart>true</autostart>
</proxy-scheme>
```

Client-side NamedCache Locking

By default, the Coherence*Extend clustered service disallows Coherence*Extend clients from acquiring `NamedCache` locks. To enable client-side locking, use the `<cache-service-proxy>` child configuration element. For example:

Example 18–16 Client Configuration to Allow NamedCache Locking

```
<proxy-scheme>
  ...

  <proxy-config>
    <cache-service-proxy>
      <lock-enabled>true</lock-enabled>
    </cache-service-proxy>
  </proxy-config>

  <autostart>true</autostart>
</proxy-scheme>
```

If you enable client-side locking and your client application uses the `NamedCache.lock()` and `unlock()` methods, it is important that you specify the **member-based** (rather than thread-based) locking strategy for any Partitioned or Replicated cache services defined in your cluster-side Coherence cache configuration descriptor. Because the Coherence*Extend clustered service uses a pool of threads to execute client requests concurrently, it cannot guarantee that the same thread will execute subsequent requests from the same Coherence*Extend client.

To specify the member-based locking strategy for a Partitioned or Replicated cache service, use the `<lease-granularity>` configuration element. Example 18–17 illustrates a sample configuration.

Example 18–17 Client Configuration to Allow Locking for Partitioned or Replicated Caches

```
<distributed-scheme>
  <scheme-name>dist-default</scheme-name>
  <lease-granularity>member</lease-granularity>
  <backing-map-scheme>
    <local-scheme/>
  </backing-map-scheme>
  <autostart>true</autostart>
</distributed-scheme>
```

Disabling Proxied Services

By default, the Coherence*Extend clustered service exposes two proxied services to clients: a `CacheService` proxy and an `InvocationService` proxy. In some cases, it may be desirable to disable one of the two proxies. This is possible by using the `<enabled>` configuration element in each of the corresponding proxy configuration sections. For example, to disable the InvocationService proxy so that remote clients

cannot execute `Invocable` objects within the cluster, you'd configure the Coherence*Extend clustered service as illustrated in Example 18–18:

Example 18–18 Client Configuration to Disable Proxy Service

```
<proxy-scheme>
  ...

  <proxy-config>
    <invocation-service-proxy>
      <enabled>false</enabled>
    </invocation-service-proxy>
  </proxy-config>

  <autostart>true</autostart>
</proxy-scheme>
```

Likewise, to prevent remote clients from accessing caches in the cluster, you would use a configuration similar to the one illustrated in Example 18–19:

Example 18–19 Client Configuration to Prevent Cache Access

```
<proxy-scheme>
  ...

  <proxy-config>
    <cache-service-proxy>
      <enabled>false</enabled>
    </cache-service-proxy>
  </proxy-config>

  <autostart>true</autostart>
</proxy-scheme>
```

19

High Resolution Timesource (Linux)

Linux has several high resolution timesources to choose from, the fastest TSC (Time Stamp Counter) unfortunately is not always reliable. Linux chooses TSC by default, and during boot checks for inconsistencies, if found it switches to a slower safe timesource. The slower time sources can be 10 to 30 times more expensive to query then the TSC timesource, and may have a measurable impact on Coherence performance. Note that Coherence and the underlying JVM are not aware of the timesource which the operating system is using. It is suggested that you check your system logs (/var/log/dmesg) to verify that the following is not present.

```
kernel: Losing too many ticks!
kernel: TSC cannot be used as a timesource.
kernel: Possible reasons for this are:
kernel:   You're running with Speedstep,
kernel:   You don't have DMA enabled for your hard disk (see hdparm),
kernel:   Incorrect TSC synchronization on an SMP system (see dmesg).
kernel: Falling back to a sane timesource now.
```

As the log messages suggest, this can be caused by a variable rate CPU (SpeedStep), having DMA disabled, or incorrect TSC synchronization on multi CPU machines. If present it is suggested that you work with your system administrator to identify the cause and allow the TSC timesource to be used.

20

Performance Tuning

To achieve maximum performance with Coherence it is suggested that you test and tune your operating environment. Testing is covered in Chapter 17, "Performing a Datagram Test for Network Performance."

Tuning recommendations are available for:

- Operating System Tuning
- Network Tuning
- JVM Tuning
- Coherence Network Tuning

Operating System Tuning

- Socket Buffer Sizes
- High Resolution timesource (Linux)
- Datagram size (Microsoft Windows)
- Thread Scheduling (Microsoft Windows)
- Swapping

Socket Buffer Sizes

To help minimization of packet loss, the operating system socket buffers need to be large enough to handle the incoming network traffic while your Java application is paused during garbage collection. By default Coherence will attempt to allocate a socket buffer of 2MB. If your operating system is not configured to allow for large buffers Coherence will use smaller buffers. Most versions of UNIX have a very low default buffer limit, which should be increased to at least 2MB.

Starting with Coherence 3.1 you will receive the following warning if the operating system failed to allocate the full size buffer.

Example 20–1 Message Indicating OS Failed to Allocate the Full Buffer Size

```
UnicastUdpSocket failed to set receive buffer size to 1428 packets (2096304
bytes); actual size is 89 packets (131071 bytes). Consult your OS documentation
regarding increasing the maximum socket buffer size. Proceeding with the actual
value may cause sub-optimal performance.
```

Though it is safe to operate with the smaller buffers it is recommended that you configure your operating system to allow for larger buffers.

On Linux execute (as root):

```
sysctl -w net.core.rmem_max=2096304
sysctl -w net.core.wmem_max=2096304
```

On Solaris execute (as root):

```
ndd -set /dev/udp udp_max_buf 2096304
```

On AIX execute (as root):

```
no -o rfc1323=1
no -o sb_max=4194304
```

> **Note:** Note that AIX only supports specifying buffer sizes of 1MB, 4MB, and 8MB. Additionally there is an issue with IBM's 1.4.2, and 1.5 JVMs which may prevent them from allocating socket buffers larger then 64K. This issue has been addressed in IBM's 1.4.2 SR7 SDK and 1.5 SR3 SDK.

On Windows:

Windows does not impose a buffer size restriction by default.

Other:

For information on increasing the buffer sizes for other operating systems please refer to your operating system's documentation.

You may configure Coherence to request alternate sized buffers for packet publishers and unicast listeners by using the `coherence/cluster-config/packet-publisher/packet-buffer/maximum-packets` and `coherence/cluster-config/unicast-listener/packet-buffer/maximum-packets` elements. For more information, see "packet-buffer" on page H-32.

High Resolution timesource (Linux)

Linux has several high resolution timesources to choose from, the fastest TSC (Time Stamp Counter) unfortunately is not always reliable. Linux chooses TSC by default, and during boot checks for inconsistencies, if found it switches to a slower safe timesource. The slower time sources can be 10 to 30 times more expensive to query then the TSC timesource, and may have a measurable impact on Coherence performance. Note that Coherence and the underlying JVM are not aware of the timesource which the operating system is using. It is suggested that you check your system logs (`/var/log/dmesg`) to verify that the following is not present. Example 20–2 illustrates a sample timesource log.

Example 20–2 Log Message from a Linux Timesource

```
kernel: Losing too many ticks!
kernel: TSC cannot be used as a timesource.
kernel: Possible reasons for this are:
kernel:   You're running with Speedstep,
kernel:   You don't have DMA enabled for your hard disk (see hdparm),
kernel:   Incorrect TSC synchronization on an SMP system (see dmesg).
kernel: Falling back to a sane timesource now.
```

As the log messages suggest, this can be caused by a variable rate CPU (SpeedStep), having DMA disabled, or incorrect TSC synchronization on multi CPU machines. If present it is suggested that you work with your system administrator to identify the cause and allow the TSC timesource to be used.

Datagram size (Microsoft Windows)

Microsoft Windows supports a fast I/O path which is used when sending "small" datagrams. The default setting for what is considered a small datagram is 1024 bytes; increasing this value to match your network MTU (normally 1500) can significantly improve network performance.

To adjust this parameter:

1. Run Registry Editor (`regedit`)
2. Locate the following registry key `HKLM\System\CurrentControlSet\Services\AFD\Parameters`
3. Add the following new `DWORD` value `Name: FastSendDatagramThreshold Value: 1500` (decimal)
4. Reboot

> **Note:** Included in Coherence 3.1 and above is an `optimize.reg` script which will perform this change for you, it can be found in the `coherence/bin` directory of your installation. After running the script you must reboot your computer for the changes to take effect.

For more details on this parameter see Appendix C of
http://technet.microsoft.com/en-us/library/bb726981.aspx

Thread Scheduling (Microsoft Windows)

Windows (including NT, 2000 and XP) is optimized for desktop application usage. If you run two console ("DOS box") windows, the one that has the focus can use almost 100% of the CPU, even if other processes have high-priority threads in a running state. To correct this imbalance, you must configure the Windows thread scheduling to less-heavily favor foreground applications.

1. Open the Control Panel.
2. Open **System**.
3. Select the **Advanced** tab.
4. Under **Performance** select **Settings**.
5. Select the **Advanced** tab.
6. Under **Processor** scheduling, choose **Background services**.

> **Note:** Coherence includes an `optimize.reg` script which will perform this change for you, it can be found in the `coherence/bin` directory of your installation.

Swapping

Ensure that you have sufficient memory such that you are not making active use of swap space on your machines. You may monitor the swap rate using tools such as `vmstat` and `top`. If you find that you are actively moving through swap space this will likely have a significant impact on Coherence's performance. Often this will manifest itself as Coherence nodes being removed from the cluster due to long periods of unresponsiveness caused by them having been "swapped out".

Network Tuning

Network Interface Settings

Verify that your Network card (NIC) is configured to operate at it's maximum link speed and at full duplex. The process for doing this varies between operating systems.

On Linux execute (as root):

```
ethtool eth0
```

See the man page on `ethtool` for further details and for information on adjust the interface settings.

On Solaris execute (as root):

```
kstat ce:0 | grep link_
```

This will display the link settings for interface 0. Items of interest are `link_duplex` (2 = full), and `link_speed` which is reported in Mbps.

> **Note:** If running on Solaris 10, please review Sun issues 102712 and 102741 which relate to packet corruption and multicast disconnections. These will most often manifest as either `EOFExceptions`, "Large gap" warnings while reading packet data, or frequent packet timeouts. It is highly recommend that the patches for both issues be applied when using Coherence on Solaris 10 systems.

On Windows:

1. Open the Control Panel.
2. Open **Network Connections**.
3. Open the **Properties** dialog for desired network adapter.
4. Select **Configure**.
5. Select the **Advanced** tab.
6. Locate the driver specific property for **Speed** & **Duplex**.
7. Set it to either auto or to a specific speed and duplex setting.

Bus Considerations

For 1Gb and faster PCI network cards the system's bus speed may be the limiting factor for network performance. PCI and PCI-X busses are half-duplex, and all devices will run at the speed of the slowest device on the bus. Standard PCI buses have a maximum throughput of approximately 1Gb/sec and thus are not capable of fully using a full-duplex 1Gb NIC. PCI-X has a much higher maximum throughput (1GB/sec), but can be hobbled by a single slow device on the bus. If you find that you are not able to achieve satisfactory bidirectional data rates it is suggested that you evaluate your machine's bus configuration. For instance simply relocating the NIC to a private bus may improve performance.

Network Infrastructure Settings

If you experience frequent multi-second communication pauses across multiple cluster nodes you may need to increase your switch's buffer space. These communication pauses can be identified by a series of Coherence log messages identifying communication delays with multiple nodes which are not attributable to local or remote GCs.

Example 20–3 Message Indicating a Communication Delay

```
Experienced a 4172 ms communication delay (probable remote GC) with Member(Id=7,
Timestamp=2006-10-20 12:15:47.511, Address=192.168.0.10:8089, MachineId=13838);
320 packets rescheduled, PauseRate=0.31, Threshold=512
```

Some switches such as the Cisco 6500 series support configuration the amount of buffer space available to each Ethernet port or ASIC. In high load applications it may be necessary to increase the default buffer space. On Cisco this can be accomplished by executing:

```
fabric buffer-reserve high
```

See Cisco's documentation for additional details on this setting.

Ethernet Flow-Control

Full duplex Ethernet includes a flow-control feature which allows the receiving end of a point to point link to slow down the transmitting end. This is implemented by the receiving end sending an Ethernet PAUSE frame to the transmitting end, the transmitting end will then halt transmissions for the interval specified by the PAUSE frame. Note that this pause blocks **all** traffic from the transmitting side, even traffic destined for machines which are not overloaded. This can induce a head of line blocking condition, where one overloaded machine on a switch effectively slows down all other machines. Most switch vendors will recommend that Ethernet flow-control be disabled for inter switch links, and at most be used on ports which are directly connected to machines. Even in this setup head of line blocking can still occur, and thus it is advisable to disable Ethernet flow-control all together. Higher level protocols such as TCP/IP and Coherence TCMP include their own flow-control mechanisms which are not subject to head of line blocking, and also negate the need for the lower level flow-control.

For more details on this subject see
http://www.networkworld.com/netresources/0913flow2.html.

Path MTU

By default Coherence assumes a 1500 byte network MTU, and uses a default packet size of 1468 based on this assumption. Having a packet size which does not fill the MTU will result is an under used network. If your equipment uses a different MTU, then configure Coherence by specifying a packet size which is 32 bytes smaller then the network path's minimal MTU. The packet size may be specified in `coherence/cluster-config/packet-publisher/packet-size/maximum-length` and `preferred-length` configuration elements. For more information on these elements, see "packet-size" on page H-38.

If you are unsure of your equpiment's MTU along the full path between nodes you can use either the standard ping or traceroute utility to determine it. To do this, execute a series of ping or traceroute operations between the two machines. With each attempt you will specify a different packet size, starting from a high value and progressively moving downward until the packets start to make it through without fragmentation. You will need to specify a particular packet size, and to not fragment the packets.

On Linux execute:

```
ping -c 3 -M do -s 1468 serverb
```

On Solaris execute:

```
traceroute -F serverb 1468
```

On Windows execute:

```
ping -n 3 -f -l 1468 serverb
```

On other operating systems: Consult the documentation for the `ping` or `traceroute` command to see how to disable fragmentation, and specify the packet size.

If you receive a message stating that packets must be fragmented then the specified size is larger then the path's MTU. Decrease the packet size until you find the point at which packets can be transmitted without fragmentation. If you find that you need to use packets smaller then 1468 you may want to contact your network administrator to get the MTU increased to at least 1500.

JVM Tuning

Server Mode

It is recommended that you run all your Coherence JVMs in server mode, by specifying the `-server` on the JVM command line. This allows for several performance optimizations for long running applications.

Sizing the Heap

It is generally recommended that heap sizes be kept at 1GB or below as larger heaps will have a more significant impact on garbage collection times. On 1.5 and higher JVMs larger heaps are reasonable, but will likely require additional GC tuning. For more information, see "Heap Size Considerations".

Running with a fixed sized heap will save your JVM from having to grow the heap on demand and will result in improved performance. To specify a fixed size heap use the `-Xms` and `-Xmx` JVM options, setting them to the same value. For example:

```
java -server -Xms1024m -Xmx1024m ...
```

Note that the JVM process will consume more system memory then the specified heap size, for instance a 1GB JVM will consume 1.3GB of memory. This should be taken into consideration when determining the maximum number of JVMs which you will run on a machine. The actual allocated size can be monitored with tools such as top. See "Heap Size Considerations" for additional details on heap size considerations.

GC Monitoring & Tuning

Frequent garbage collection pauses which are in the range of 100ms or more are likely to have a noticeable impact on network performance. During these pauses a Java application is unable to send or receive packets, and in the case of receiving, the operating system buffered packets may be discarded and need to be retransmitted.

Specify `-verbose:gc` or `-Xloggc:` on the JVM command line to monitor the frequency and duration of garbage collection pauses.

See `http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html` for details on GC tuning.

Starting with Coherence 3.2 log messages will be generated when one cluster node detects that another cluster node has been unresponsive for a period, generally indicating that a target cluster node was in a GC cycle.

Example 20–4 Message Indicating Target Cluster Node is in Garbage Collection Mode

```
Experienced a 4172 ms communication delay (probable remote GC) with Member(Id=7,
Timestamp=2006-10-20 12:15:47.511, Address=192.168.0.10:8089, MachineId=13838);
320 packets rescheduled, PauseRate=0.31, Threshold=512
```

`PauseRate` indicates the percentage of time for which the node has been considered unresponsive since the stats were last reset. Nodes reported as unresponsive for more then a few percent of their lifetime may be worth investigating for GC tuning.

Coherence Network Tuning

Coherence includes configuration elements for throttling the amount of traffic it will place on the network; see the documentation for <traffic-jam>, <flow-control> and <burst-mode>, these settings are used to control the rate of packet flow within and between cluster nodes.

Validation

To determine how these settings are affecting performance you need to check if you're cluster nodes are experiencing packet loss and/or packet duplication. This can be obtained by looking at the following JMX stats on various cluster nodes:

- `ClusterNodeMBean.PublisherSuccessRate`—If less then 1.0, packets are being detected as lost and being resent. Rates below 0.98 may warrant investigation and tuning.
- `ClusterNodeMBean.ReceiverSuccessRate`—If less then 1.0, the same packet is being received multiple times (duplicates), likely due to the publisher being overly aggressive in declaring packets as lost.

- `ClusterNodeMBean.WeakestChannel`—Identifies the remote cluster node which the current node is having most difficulty communicating with.

For information on using JMX to monitor Coherence see Chapter 22, "How to Manage Coherence Using JMX."

21

Setting Single Server Mode

If you want to perform unit testing or quick restarts, you might find it more convenient to avoid the network and run in single-server mode. To constrain Coherence to run on a single server, set the multicast packet time-to-live to 0, and set the unicast IP address to an address that is not currently being used; for example: 127.0.0.1.

You can configure these properties either by declaring system properties on the command line or by editing the values in the operational configuration descriptor, `tangosol-coherence.xml` file.

Setting Single Server Mode in the Operation Configuration Descriptor

In the `tangosol-coherence.xml` file, the multicast packet time to live value is defined by the `<time-to-live>` subelement of the `<multicast-listener>` element. The `<time-to-live>` value determines the maximum number of "hops" a packet may traverse between network segments. Setting this subelement to `0` keeps the packets from leaving the originating machine.

The unicast IP address is defined by the `<address>` subelement of the `<unicast-listener>` element. This subelement specifies the IP address that a Socket will listen or publish on. Setting this subelement to an IP address that is never used will prevent Coherence from joining the network.

The following XML code fragment illustrates a single server mode configuration in the `tangosol-coherence.xml` file.

Example 21–1 Single Server Mode Configuration

```
<coherence>
    <cluster-config>
        ...
        <multicast-listener>
            <time-to-live>0<\time-to-live>
            ...
        <mulitcast-listener>
        ...
        <unicast-listener>
            <address>127.0.0.1<\address>
            ...
        <\unicast-listener>
        ...
   <\cluster-config>
<\coherence>
```

Setting Single Server Mode on the Command Line

Coherence defines system properties that allow you to set the multicast packet time-to-live and the unicast IP address for single server mode on the command line. This feature is useful when you need to change the settings for a single JVM, or if you want to start an application with settings that differ from those in the descriptor files.

The following system properties can be used to define single server mode.

- `tangosol.coherence.ttl`—Multicast packet time to live. Set to "0" to keep the packets from leaving the originating machine.
- `tangosol.coherence.localhost`—Unicast IP address. Set to an address that is not currently being used; for example: 127.0.0.1.

The sample command line in Example 21–2 illustrates starting coherence in single server mode:

Example 21–2 Command to Start Coherence in Single Server Mode

```
java -Dtangosol.coherence.localhost=127.0.0.1 -Dtangosol.coherence.ttl=0 -jar
coherence.jar
```

See Appendix L, "Command Line Overrides" for more information on system properties defined by Coherence.

Part III

Managing and Monitoring Oracle Coherence

This section contains the following chapters:

22

How to Manage Coherence Using JMX

Coherence includes facilities for managing and monitoring Coherence resources by using the Java Management Extensions (JMX) API. JMX is a Java standard for managing and monitoring Java applications and services. It defines a management architecture, design patterns, APIs, and services for building general solutions to manage Java-enabled resources. This section assumes familiarity with JMX terminology. If you are new to JMX, you should start is with this article: *"Getting Started with Java Management Extensions (JMX): Developing Management and Monitoring Solutions"*.

To manage Coherence using JMX:

- Add JMX libraries to the Coherence classpath (if necessary)
- Configure the Coherence Management Framework
- Access Coherence MBeans to view and manipulate them using a JMX client of your choice

> **Note: JMX support**:
>
> Coherence Enterprise Edition and higher support clustered JMX, allowing access to JMX statistics for the entire cluster from any member. Coherence Standard Edition provides only local JMX information.

Add JMX libraries to the Coherence classpath

To manage a Coherence cluster using JMX, ensure that you have the necessary JMX 1.0 or later classes (javax.management.*) in the classpath of at least one Coherence cluster node, known as an MBeanServer host. The cluster nodes that are not MBeanServer hosts will be managed by the MBeanServer host(s) by using the Coherence Invocation service.

All compliant Java SE 5.0 JREs and Java EE application servers supply a JMX 1.0 or later implementation; therefore, if the MBeanServer host node is running within a Java SE 5.0 JVM or Java EE application server, no additional actions are necessary. For standalone applications running within a pre-Java SE 5.0 JVM, you can download the necessary JMX libraries from the JMX download Web site and add them to the classpath.

Configure the Coherence Management Framework

In most cases, you can enable JMX management simply by setting the `tangosol.coherence.management` Java system property on all Coherence cluster nodes that are acting as MBeanServer hosts and the `tangosol.coherence.management.remote` Java system property on all cluster nodes:

```
-Dtangosol.coherence.management=all

-Dtangosol.coherence.management.remote=true
```

The use of dedicated JMX cluster members is a common pattern. This approach avoids loading JMX software into every single cluster member, while still providing fault-tolerance should a single JMX member run into issues.

In general, the Coherence Management Framework is configured by the `management-configuration` operational configuration element in the Coherence Operational Configuration deployment descriptor (`tangosol-coherence.xml`). The following subelements control the behavior of the Management Framework:

Table 22–1 Elements that Control the Behavior of the Management Framework

Element	Description
allow-remote-management	Specifies whether this cluster node will register its MBeans in a remote `MBeanServer`(s).
domain-name	Specifies the name of the JMX domain used to register MBeans exposed by the Coherence Management Framework.
managed-nodes	Specifies whether a cluster node's JVM has an in-process `MBeanServer` and if so, whether the node allows management of other nodes' managed objects. Valid values are none, local-only, remote-only and all. For example, if a node has an in-process `MBeanServer` and you would like this node to manage other nodes' MBeans, then set this attribute to all.
read-only	Specifies whether the MBeans exposed by this cluster node allow operations that modify run-time attributes.

For additional information on each of these attributes, see Appendix H, "Operational Configuration Elements."

Access Coherence MBeans

After configuring the Coherence Management Framework and launching one or more Coherence cluster nodes (at least one being an MBeanServer host) you can view and manipulate the Coherence MBeans registered by all cluster nodes using standard JMX API calls. See the Javadoc for the `com.tangosol.net.management.Registry` class for details on the various MBean types registered by Coherence clustered services.

Coherence ships with two examples that demonstrate accessing Coherence MBeans by using JMX. The first uses the `HttpAdapter`, shipped as part of the JMX reference implementation (`jmxtools.jar`). To run the example on a pre-Java SE 5.0 JVM, start the Coherence command line application using the following command on Windows (note that it is broken up into multiple lines here only for formatting purposes; this is a single command entered on one line):

```
java -cp jmxri.jar;jmxtools.jar;coherence.jar
```

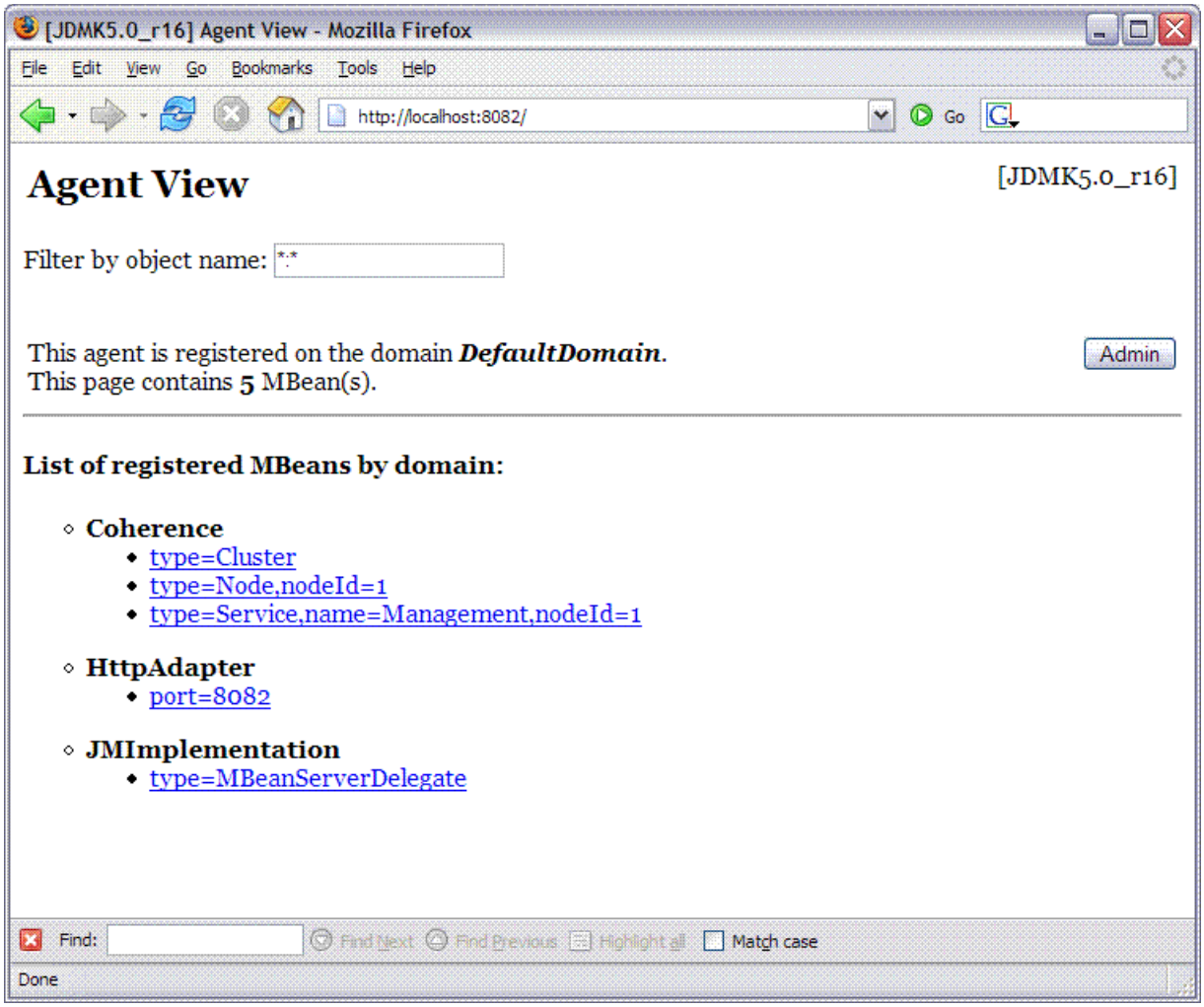
```
    -Dtangosol.coherence.management=all
    -Dtangosol.coherence.management.remote=true
     com.tangosol.net.CacheFactory
```

On UNIX:

```
java -cp jmxri.jar:jmxtools.jar:coherence.jar
     -Dtangosol.coherence.management=all
     -Dtangosol.coherence.management.remote=true
      com.tangosol.net.CacheFactory
```

When the Coherence command line application has started, enter `jmx 8082` and press **return**. This starts the `HttpAdapter` on `http://localhost:8082` in the cluster node's JVM and makes the cluster node an `MBeanServer` host. You can now use the `HttpAdapter` Web application to view and manipulate Coherence MBeans registered by all cluster nodes:

Figure 22–1 Viewing the HttpAdapter Web Application in a Browser



This figure is described in the text.

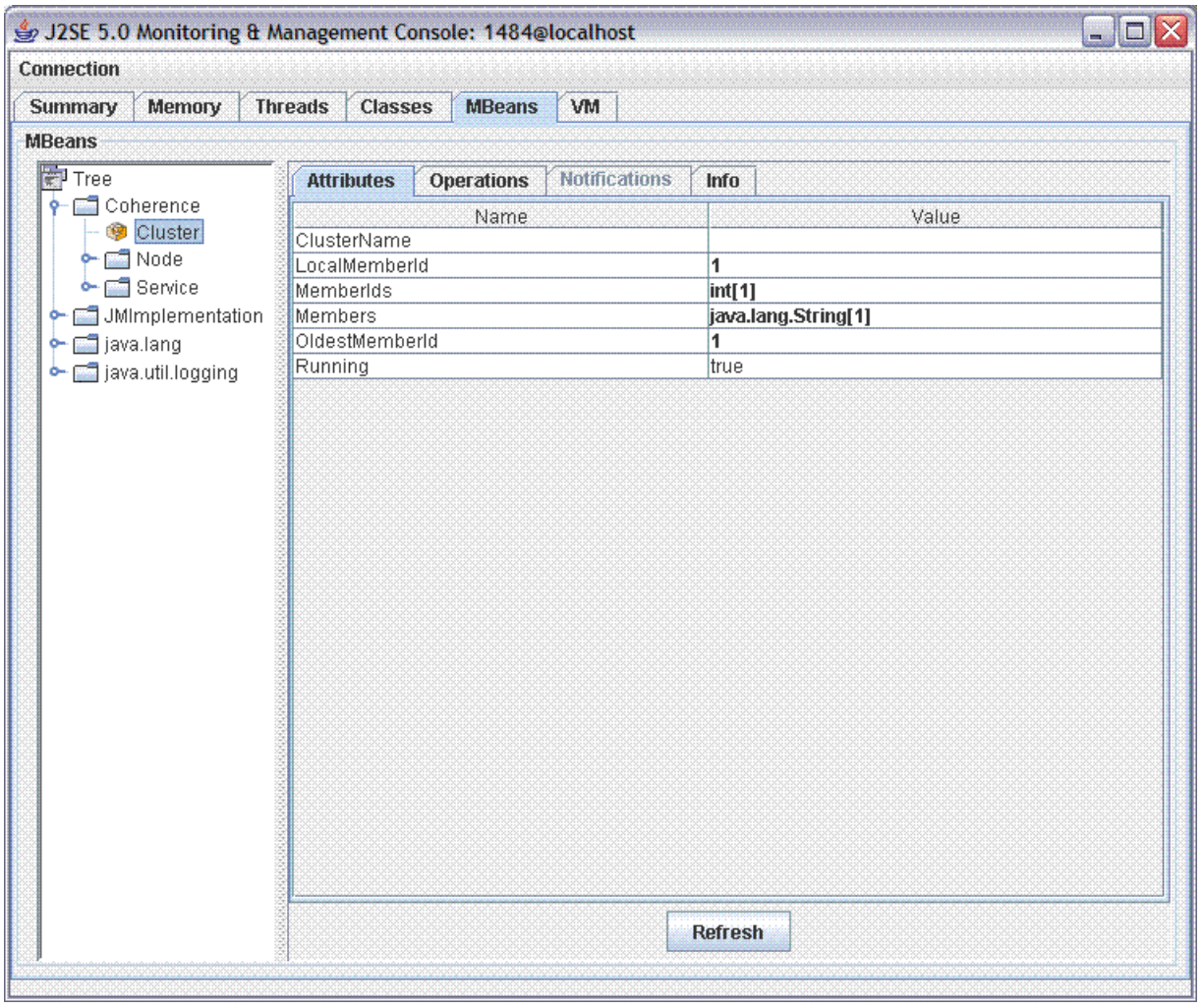
****

Alternatively, you can run this example with the Sun Java SE 5.0 JVM and use the JConsole utility included with the Sun Java SE 5.0 JDK to view and manipulate Coherence MBeans. To do so, start the Coherence command line application using the following command (note that it is broken up into multiple lines here only for formatting purposes; this is a single command entered on one line):

```
java -Dcom.sun.management.jmxremote
     -Dtangosol.coherence.management=all
     -Dtangosol.coherence.management.remote=true
     -jar coherence.jar
```

When the Coherence command line application has started, launch the JConsole utility (located in the `bin` directory of the Sun Java SE 5.0 JDK distribution) and open a new connection to the JVM running the Coherence command line application:

Figure 22–2 Using the JConsole Utility to Display and Manipulate Coherence MBeans



This figure is described in the text.

****

The second example is a JSP page (`JmxCacheExplorer.jsp`) that displays basic information on each running Coherence cache using JMX API calls. You can find this

example in the `examples/jsp/explore` directory under the root of your Coherence installation.

Additional JMX examples may be found on the Coherence Forums.

Using Coherence MBeanConnector to Access MBeans

Coherence ships with a program to launch a cluster node as a dedicated MBeanServer host. This program provides access to Coherence MBeans by using the JMX Remote API using RMI or the HTTP server provided by Sun's JMX RI. The RMI and HTTP ports are user-configurable, allowing for access through a firewall. The server is started using the following command (note that it is broken up into multiple lines here only for formatting purposes; this is a single command entered on one line):

```
java -Dtangosol.coherence.management=all
     -cp coherence.jar com.tangosol.net.management.MBeanConnector [-http -rmi]
```

To allow access by using JMX RMI, include the `-rmi` flag. To allow access by using HTTP and a web browser, include the -http flag. Both flags may be included; however at least one must present for the node to start.

Table 22–2 describes optional properties that can be used for JMX RMI configuration:

Table 22–2 Optional Properties that can be used for JMX RMI Configuration

Property	Description
tangosol.coherence.management.remote.host	The host that the JMX server will bind to. Default is `localhost`. (NOTE: on Redhat Linux this may have to be changed to the host name or IP address)
tangosol.coherence.management.remote.registryport	The port used for the JMX RMI registry. Default is `9000`.
tangosol.coherence.management.remote.connectionport	The port used for the JMX RMI connection. Default is `3000`.

Table 22–3 describes optional properties that can be used for HTTP configuration. (NOTE: This flag requires Sun's JMX RI in the classpath):

Table 22–3 Optional Properties that can be used for Http Configuration

Property	Description
tangosol.coherence.management.remote.httpport	The port used for the HTTP connection. Default is `8888`.

To connect by using JConsole with default settings use the following command:

```
jconsole service:jmx:rmi://localhost:3000/jndi/rmi://localhost:9000/server
```

To connect by using HTTP with default settings use the following URL:

```
http://localhost:8888
```

Configuring Management Refresh Methodology

The current release of Coherence offers several ways to reduce the latency of management information. Refresh policy was introduced in Coherence 3.3 to allow for optimization of the retrieval of information from remotely managed nodes. Two new

settings were added to help integrators and administrators configure the refresh policy.

The `tangosol.coherence.management.refresh.expiry` property specifies the minimum time interval between the remote retrieval of management information from remote nodes.

```
-Dtangosol.coherence.management.refresh.expiry
```

The value of this element must be in the following format:

```
[\d]+[[.][\d]+]?[MS|ms|S|s|M|m|H|h|D|d]?
```

where the first non-digits (from left to right) indicate the unit of time duration:

- MS or ms (milliseconds)
- S or s (seconds)
- M or m (minutes)
- H or h (hours)
- D or d (days)

If the value does not contain a unit, a unit of milliseconds is assumed.

The `tangosol.coherence.management.refresh.policy` property defines the refresh policy for the MBean.

```
-Dtangosol.coherence.management.refresh.policy
```

Table 22–4 describes valid values for this property.

Table 22–4 Values for the tangosol.coherence.management.refresh.policy Property

Setting	Description
refresh-ahead	MBeans are refreshed before they are requested based on prior usage patterns after the expiry delay has passed. This setting can reduce latency of the management information with a minor increase in network consumption. This setting is best when MBeans are accessed in a repetitive/programmatic pattern.
refresh-behind	Each MBean will be refreshed after the data is accessed. This method ensures optimal response time. However, the information returned will be offset by the last refresh time.
refresh-expired (default)	This setting has the same functionality as in pre-3.4 Coherence releases. Each MBean will be refreshed from the remote node when it is accessed and the expiry delay has passed from the last refresh. This setting is best used when MBeans are accessed in a random pattern.

23

JMX Reporter

Coherence 3.4 provides a JMX reporting capability (the Reporter). The Reporter provides out-of-the-box reports that help administrators and developers manage capacity and trouble shoot problems.

> **Note:** Plan for archiving and removing. Due to the volume of the information created by the Reporter, you must have a plan for archiving and/or removing the results BEFORE starting the Reporter.

Basic Configuration

Enabling the Reporter with basic content requires setting the system properties:

Example 23–1 illustrates the properties on the "management" node.

Example 23–1 System Properties for Reporter on the "Management" Node

```
-Dtangosol.coherence.management.report.autostart=true
-Dtangosol.coherence.management=all
-Dcom.sun.management.jmxremote
```

Example 23–2 illustrates the properties on the "managed" node.

Example 23–2 System Properties for Reporter on the "Managed" Node

```
-Dtangosol.coherence.management.remote=true
```

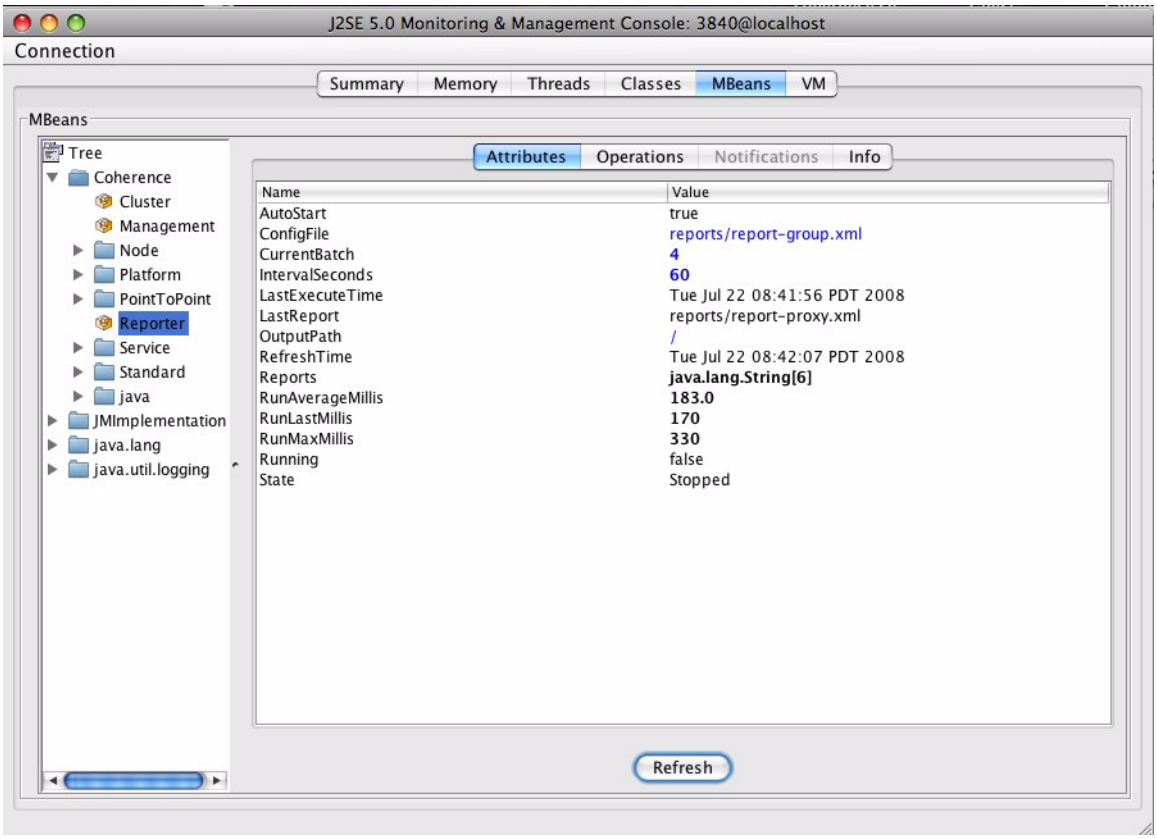
Basic configuration will create a single Reporter node that will log the JMX statistics for all nodes in the cluster. The log files will be placed in the working directory of the application.

Administration

The JMX Reporter is managed through an MBean under the Coherence Domain. The Reporter MBean provides information related to the status and performance of the Reporter. The MBean also provides the capability to start and stop the service and run a report on demand.

Figure 23–1 illustrates the attributes available to the Reporter MBean. The JConsole is being used to view the MBean.

Figure 23–1 Reporter Attributes in JConsole

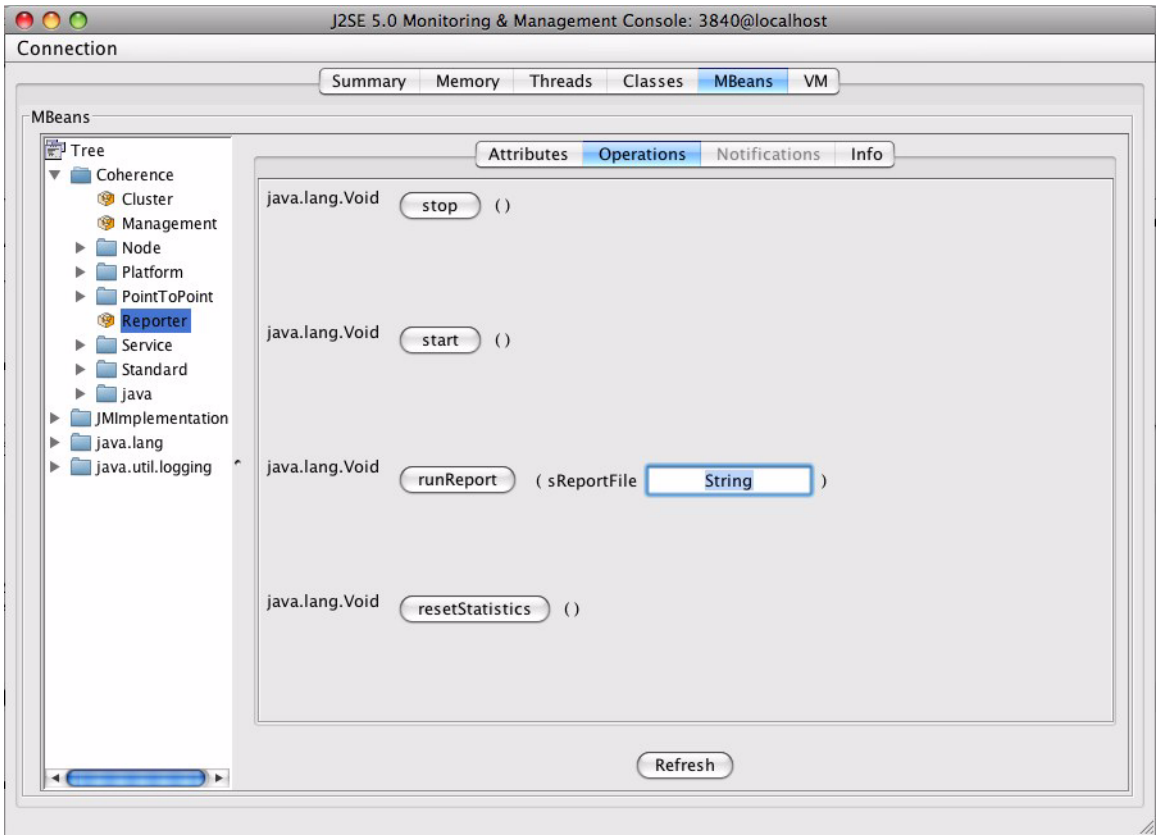


This figure is described in the text.

****

Figure 23–2 illustrates the operations available to the Reporter MBean. For a full description of the Reporter Attributes see the `Reporter` section of the javadoc.

Figure 23–2 Reporter Operations in JConsole



This figure is described in the text.

****

Data Analysis

Seven files are created each hour by the Reporter. Each file is prefixed with the date and hour the report was executed in a *YYYYMMDDHH* format. This allows for easy location and purging of unwanted information. The files generated are described in Table 23–1:

Table 23–1 File Names Generated by Reporter

File Name	Description
YYYYMMDDHH-memory-status.txt	Contains memory and garbage collection information about each node.
YYYYMMDDHH-network-health.txt	Contains the publisher success rates and receiver success rates for the entire grid
YYYYMMDDHH-network-health-detail.txt	Contains the publisher success rates and receiver success rates for each node
YYYYMMDDHH-node.txt	Contains the list of nodes that were members of the grid
YYYYMMDDHH-service.txt	Contains Request and Task information for each service.
YYYYMMDDHH-proxy.txt	Contains utilization information about each proxy node in the grid

Table 23–1 (Cont.) File Names Generated by Reporter

File Name	Description
YYYYMMDDHH-cache-usage.txt	Contains cache utilization (put, get, and so on) statistic for each cache

See Chapter 26, "Analyzing Reporter Content" for a complete description of the data contained in each file.

Advanced Configuration

Creating Custom Reports

1. Create the custom report configuration file. See Chapter 24, "How to Create a Custom Report."
2. Update report batch to execute the report. See Chapter 25, "How to Modify Report Batch."
3. Run on demand. See Chapter 27, "How to Run a Report on Demand."

Running Reporter in a Distributed Configuration

A distributed configuration is only recommended in situations where grid stability is an issue. In this configuration, the distributed reporters will run independently, and the execution times will not align. Therefore, grid level analysis is extremely difficult but node level analysis during periods when nodes may be leaving or joining the grid will still be available.

When running in distributed mode, each node logs local JMX statistics while allowing for centralized management of the Reporters. To enable this configuration set the following system properties

On the "managing" node:

Example 23–3 System Properties for Reporter in Distributed Mode on the "Managing" Node

```
-Dtangosol.coherence.management.report.autostart=false
-Dtangosol.coherence.management.report.distributed=true
-Dtangosol.coherence.management=all
-Dcom.sun.management.jmxremote
```

On the "managed" node:

Example 23–4 System Properties for Reporter in Distributed Mode on the "Managed" Node

```
-Dtangosol.coherence.management.report.autostart=true
-Dtangosol.coherence.management.report.distributed=true
-Dtangosol.coherence.management=local-only
-Dtangosol.coherence.management.remote=true
```

24

How to Create a Custom Report

The Coherence reporting feature provides a capable query definition that allows for any information residing in the Coherence JMX data source to be logged to a text file. After a custom report has been created, it can be included in a report batch and executed on a specified time interval by the `ReportControl` MBean. For a complete description of the report configuration XML file see the `report-config.dtd` which is packaged in the `coherence.jar` file.

Configuring a Report File

To correctly generate the report file, several elements must be configured. These elements are described in Table 24–1.

Table 24–1 Elements to Configure an Output File

Element	Optional/ Requited	Description
<file-name>	Required	The file name to create or update when the report is executed. For more information, see "file-name Element".
<delim>	Optional	The column delimiter for the report. Valid values are `{tab}`, `{space}` or a printable character. The default value is `{tab}`. If a string longer than one character is entered, the first character in the string is used.
<hide-headers>	Optional	A boolean element to determine if the headers are to be included in the report. If `true`, the column headers and the report description are not included in the file. The default value is `false`.

file-name Element

The value of this element will have the output path from the `<report-path>` element pre-pended to it and the report will be generated in this location. If the Coherence node cannot access this path, then the file will not be created.

file-name Macros

There are pre-defined macros that you can use with the file-name element. These macros can add a node name, a batch number, or a date to the generated file name.

Table 24–2 Macros that can be Used with the file-name Element

Macro	Description
batch	Will include a batch Identifier into the filename of the report. If the information is kept for a short amount of time or is frequently uploaded into and RDBMS.
date	Will include the date (with the format `YYYYMMDD`), into the file name of the report. This is used mostly when the data will only be kept for a certain period and then will be discarded.
node	Will include the node ID into the file name string. This configuration setting is helpful when many nodes are executing the same report and the output files will be integrated for the analysis.

file-name Macro Examples

The following example will create a file `20090101_network_status.txt` on January 1, 2009. The filename will change with the system time on the node executing the report.

```
<file-name>{{date}}_network_status.txt</file-name>
```

The following example will create a file `00012_network_status.txt` when the report is executed on node 12. Note that due to the volatile nature of the Node Id, long term storage in this manner is not recommended.

```
<file-name>{node}_network_status.txt</file-name>
```

The following example will create a file `0000000021_network_status.txt` on the 21st execution of the report. Note that due to the volatile nature of the batch, long term storage in this manner is not recommended.

```
<file-name>{batch}_network_status.txt</file-name>
```

Specifying Data Columns

Data columns can be sourced from JMX Attributes, `ObjectName` key part, JMX composite attributes, JMX joined attributes, Report macros, and Report Constants.

How to Include an Attribute

To include data from MBeans returned from the query-pattern, the report must have a column with an attribute source. This is the most common item that will be included in the report.

Example 24–1 illustrates how to include the `RoleName` attribute from the query pattern `Coherence:type=Node,*`.

Example 24–1 Including an Attribute Obtained from a Query Pattern

```
<column id = "RoleName">
  <type>attribute</type>
  <name>RoleName</name>
  <header>Role Name</header>
</column>
```

How to Include Part of the Key

A value that is present in an `ObjectName` key can be obtained from the `ObjectNames` returned from the query-pattern. This value can subsequently be included in the report.

Example 24–2 illustrates how to include the `nodeId` key part from the query pattern `Coherence:type=Node,*`.

Example 24–2 Including Part of an ObjectName Key in a Report

```
<column id ="NodeId">
  <type>key</type>
  <name>nodeId</name>
  <header>Node Id</header>
</column>
```

How to Include Information from Composite Attributes

JMX composite values can be used to include part of a composite data attribute in a report.

Example 24–9 illustrates how to include the `startTime` of the `LastGCInfo` attribute from the query pattern `java.lang:type=GarbageCollector,*`.

Example 24–3 Including Information from a Composite Attribute in a Report

```
<column id="LastGCStart">
   <type>attribute</type>
   <name>LastGcInfo/startTime</name>
   <header>Last GC Start Time</header>
</column>
```

How to Include Information from Multiple MBeans

A JMX join attribute is required when a report requires information from multiple MBeans. The major considerations when creating a join is to determine both the primary query, the join query and the foreign key. The primary query should be the query that returns the appropriate number of rows for the report. The join query pattern must reference a single MBean and can not contain a wild card (`*`). The foreign key is determined by what attributes from the primary query that are required to complete the join query string.

The reporter feature that enables joins between MBeans is a column substitution macro. The column substitution allows for the resulting value from a column to be included as part of a string. A column substitution macro is a column ID attribute surrounded by curly braces "`{}`". The reporter does not check for cyclical references and will fail during execution if a cycle is configured.

Including Multiple MBean Information Example

You can draw information from more than one MBean and include it in a report. This requires a `join` between the MBeans.

Note: The major limitation of `join` attributes is that the result of the join must have only one value.

For example, if a report requires the `TotalGets` from the `Cache` MBean (`Coherence:type=cache,*`) and `RoleName` from the `Node` MBean (`Coherence:type=Node,*`), then a `join` attribute must be used.

Since a greater number of MBeans will come from the `Cache` MBean, `Coherence:type=Cache,*` would be the primary query and the `RoleName` would be the `join` attribute. The foreign key for this join is the `nodeId` key part from the `Cache` MBean and it must be included in the report. The configuration for this scenario is illustrated in Example 24–4.

Example 24–4 Including Information from Multiple MBeans in a Report

```
<column id="RoleName">
   <type>attribute</type>
   <name>RoleName</name>
   <header>Role Name</header>
    <query>
       <pattern>Coherence:type=Node,nodeId={NodeFK}</pattern>
    </query>
</column>

<column id ="NodeFK">
  <type>key</type>
  <name>nodeId</name>
  <header>Node Id</header>
</column>
```

How to Use Report Macros

There are three report macros that can be included in a report:

- Report Time (`report-time`)—is the time and date that the report was executed. This information is useful for time series analysis.
- Report Batch/Count (`report-count`)—is a long identifier that can be used to correlate information from different reports executed at the same time.
- Reporting Node (`report-node`)—is used when integrating information from the same report executed on different nodes or excluding the executing node information from the report.

To include the execution time into the report:

Example 24–5 Including Execution Time in a Report

```
<column id ="ReportTime">
   <type>global</type>
   <name>{report-time}</name>
   <header>Report Time</header>
</column>
```

To include the Report Batch/Count:

Example 24–6 Including the Report Batch/Count in a Report

```
<column id="ReportBatch">
  <type>global</type>
  <name>{report-count}</name>
  <header>batch</header>
</column>
```

To include the execution node:

Example 24–7 Including the Execution Node

```
<column id="ReportNode">
  <type>global</type>
  <name>{report-node}</name>
  <header>ExecNode</header>
  <hidden>true</hidden>
</column>
```

How to Include Constant Values

Report constants can be used to either static values or report parameters. These constants can be either double or string values. Often, these are used in filters to limit the results to a particular data set or in calculations.

Example 24–8 illustrates how to include a constant double of 1.0 in a report:

Example 24–8 Including a Constant Numeric Value in a Report

```
<column id ="One">
  <type>constant</type>
  <header>Constant1</header>
  <data-type>double</data-type>
  <value>1.0</value>
  <hidden>true</hidden>
</column>
```

Example 24–9 illustrates how to include the constant string `dist-Employee` in a report:

Example 24–9 Including a Constant String in a Report

```
<column id ="EmployeeCacheName">
  <type>constant</type>
  <header>Employee Cache Name</header>
  <data-type>string</data-type>
  <value>dist-Employee</value>
  <hidden>true</hidden>
</column>
```

Including Queries in a Report

The query is the foundation of the information included in a report. Each query includes a query pattern, column references, and an optional filter reference. The query pattern is a string that is a JMX `ObjectName` query string. This string can return one or more MBeans. The column references must be defined in the `<columns>` section of the report definition file. The filter reference must be defined in the `<filters>` section of the report section.

Example 24–10 illustrates how to include the list all the Node IDs and `RoleNames` in the cluster where the `RoleName` equals `CoherenceServer`.

Example 24–10 Including a List of the Cluster's NodeIDs and RoleNames in a Report

```
<filters>
   <filter id="equalsRef">
     <type>equals</type>
     <params>
```

```
        <column-ref>RoleRef</column-ref>
        <column-ref>StringRef</column-ref>
     </params>
   </filter>
</filters>

<query>
   <pattern>Coherence:type=Node,*</pattern>
   <filter-ref>equalsRef</filter-ref>
</query>

<row>
  <column id ="NodeRef">
    <type>key</type>
    <name>nodeId</name>
    <header>Node Id</header>
  </column>

  <column id ="RoleRef">
    <name>RoleName</name>
    <header>Role</header>
  </column>

  <column id = "StringRef">
    <type>constant</type>
    <name>ConstString</name>
    <data-type>string</data-type>
    <value>CoherenceServer</value>
    <hidden>true</hidden>
  </column>

</row>
```

Using Filters to Construct Reports

Filters limit the data returned in the Report. Filters are either comparison filters or composite filters. Comparison Filters evaluate the results of two columns while composite filters evaluate the boolean results from one or two filters. Comparison filters are `equals`, `greater`, and `less`.

Composite Filter types are `and`, `or`, and `not`. Each composite filter evaluates the filter parameters first to last and apply standard boolean logic. Composite filter evaluation uses standard short circuit logic. Cyclic references checks are not performed during execution. If a cyclic reference occurs, it will create a runtime error.

Example 24–11 illustrates how to define an `equals` filter where `RoleRef` and `StringRef` are defined columns.

Example 24–11 Using an Equals Filter for a Report

```
<filters>
   <filter id="equals">
     <type>equals</type>
     <params>
        <column-ref>RoleRef</column-ref>
        <column-ref>StringRef</column-ref>
     </params>
   </filter>
</filters>
```

Example 24–12 illustrates how to define a filter where the number of `PacketsResent` are greater than `PacketsSent` (assuming `PacketsResent` and `PacketsSent` are valid column references).

Example 24–12 Defining a "Greater Than" Filter for a Report

```
<filters>
   <filter id="greaterRef">
     <type>greater</type>
     <params>
        <column-ref>PacketsResent</column-ref>
        <column-ref>PacketsSent</column-ref>
     </params>
   </filter>
</filters>
```

Example 24–13 illustrates how to define an filter where the number of `PacketsResent` are less than `PacketsSent` (assuming `PacketsResent` and `PacketsSent` are valid column references).

Example 24–13 Defining a "Less Than" Filter for a Report

```
<filters>
   <filter id="greaterRef">
     <type>less</type>
     <params>
        <column-ref>PacketsResent</column-ref>
        <column-ref>PacketsSent</column-ref>
     </params>
   </filter>
</filters>
```

Example 24–14 illustrates how to define an `and` filter (assuming all `column-ref` values are valid).

Example 24–14 Defining an "And" Filter for a Report

```
<filters>
   <filter id="equalsRef">
     <type>equals</type>
     <params>
        <column-ref>RoleRef</column-ref>
        <column-ref>StringRef</column-ref>
     </params>
   </filter>

   <filter id="greaterRef">
     <type>greater</type>
     <params>
        <column-ref>PacketsResent</column-ref>
        <column-ref>PacketsSent</column-ref>
     </params>
   </filter>

   <filter>
     <type>and</type>
     <params>
        <filter-ref>greaterRef</filter-ref>
        <filter-ref>equalsRef</filter-ref>
     <params>
```

```
   </filter>
</filters>
```

Example 24–15 illustrates how to define an `or` filter (assuming all `column-ref` values are valid).

Example 24–15 Defining an "Or" Filter for a Report

```
<filters>
   <filter id="equalsRef">
     <type>equals</type>
     <params>
        <column-ref>RoleRef</column-ref>
        <column-ref>StringRef</column-ref>
     </params>
   </filter>

   <filter id="greaterRef">
     <type>greater</type>
     <params>
        <column-ref>PacketsResent</column-ref>
        <column-ref>PacketsSent</column-ref>
     </params>
   </filter>

   <filter>
     <type>or</type>
     <params>
        <filter-ref>greaterRef</filter-ref>
        <filter-ref>equalsRef</filter-ref>
     <params>
   </filter>
</filters>
```

Example 24–16 illustrates how to define a `not equals` filter, where `RoleRef` and `StringRef` are defined columns.

Example 24–16 Defining a "Not Equals" Filter for a Report

```
<filters>
   <filter id="equals">
     <type>equals</type>
     <params>
        <column-ref>RoleRef</column-ref>
        <column-ref>StringRef</column-ref>
     </params>
   </filter>

   <filter id = "Not">
     <type>not</type>
     <params>
        <filter-ref>equals</filter-ref>
     </params>
   </filter>
</filters>
```

Using Functions to Construct a Report

Reporter functions allow mathematical calculations to be performed on data elements within the same row of the report. The supported functions are `Add`, `Subtract`, `Multiply`, and `Divide`. Function columns can then be included as parameters into other function columns.

Function Examples

Example 24–17 illustrates how to add columns (`Attribute1` and `Attribute2`) and place the results into a third column (`Addition`).

Example 24–17 Adding Column Values and Including Results in a Different Column

```
<column id="AttributeID1">
  <name>Attribute1</name>
</column>

<column id="AttributeID2">
  <name>Attribute2</name>
</column>

<column id="Addition">
  <type>function</type>
  <name>Add2Columns</name>
  <header>Adding Columns</header>
  <function-name>add</function-name>
  <params>
    <column-ref>AttributeID1</column-ref>
    <column-ref>AttributeID2</column-ref>
  </params>
</column>
```

Example 24–18 illustrates how to subtract one column value (`Attribute2`) from another (`Attribute1`) and place the results into a third column (`Subtraction`).

Example 24–18 Subtracting Column Values and Including Results in a Different Column

```
<column id="AttributeID1">
  <name>Attribute1</name>
</column>

<column id="AttributeID2">
  <name>Attribute2</name>
</column>

<column id="Subtraction">
  <type>function</type>
  <name>Subtract2Columns</name>
  <header>Difference</header>
  <function-name>subtract</function-name>
  <params>

    <column-ref>AttributeID1</column-ref>
    <column-ref>AttributeID2</column-ref>
  </params>
</column>
```

Example 24–19 illustrates how to multiply column values (`Attribute1` and `Attribute2`) place the results into a third column (`Multiplication`).

Example 24–19 Multiplying Column Values and Including Results in a Different Column

```
<column id="AttributeID1">
  <name>Attribute1</name>
</column>

<column id="AttributeID2">
  <name>Attribute2</name>
</column>

<column id="Multiplication">
  <type>function</type>
  <name>Multiply2Columns</name>
  <header>Multiply Columns</header>
  <function-name>multiply</function-name>
  <params>
    <column-ref>AttributeID1</column-ref>
    <column-ref>AttributeID2</column-ref>
  </params>
</column>
```

Example 24–20 illustrates how to divide one column (`Attribute1`) by another (`Attribute2`) into a third column (`Division`). The result of all division is a Double data type.

Example 24–20 Dividing Column Values and Including Results in a Different Column

```
<column id="AttributeID1">
  <name>Attribute1</name>
</column>

<column id="AttributeID2">
  <name>Attribute2</name>
</column>

<column id="Division">
  <type>function</type>
  <name>Dividing2Columns</name>
  <header>Division</header>
  <function-name>Divide</function-name>
  <params>
    <column-ref>AttributeID1</column-ref>
    <column-ref>AttributeID2</column-ref>
  </params>
</column>
```

Using Aggregates to Construct a Report

Reporter aggregates allow for multiple rows to be aggregated into a single value or row. Table 24–3 describes the available aggregate types.

Table 24–3 Reporter Aggregate Types

Type	Description
avg	Calculate the mean value for all values in the column.
max	Return the maximum value for all values in the column.
min	Return the minimum value for all values in the column.
sum	Add all the values from a column.

Aggregate Examples

Sum the values in the `size` column

Example 24–21 Adding the Values in a Column

```
<column id ="SumRef">
   <type>function</type>
   <function-name>sum</function-name>
   <column-ref>size</column-ref>>
   <header>Sum</header>
</column>
```

Average the values in the `size` column

Example 24–22 Calculating the Average of Values in a Column

```
<column id ="AverageRef">
   <type>function</type>
   <header>Average</header>
   <function-name>avg</function-name>
   <column-ref>size</column-ref>>
</column>
```

Find the maximum the value in the `size` column

Example 24–23 Finding the Maximum Value in a Column

```
<column id ="MaximumRef">
   <type>function</type>
   <header>Maximum</header>
   <function-name>max</function-name>
   <column-ref>size</column-ref>>
</column>
```

Find the minimum the value in the `size` column

Example 24–24 Finding the Minimum Value in a Column

```
<column id ="MinimumRef">
   <type>function</type>
   <header>Minimum</header>
   <function-name>min</function-name>
   <column-ref>size</column-ref>>
</column>
```

Constructing Delta Functions

Many numeric attributes in the Coherence report are cumulative. These values are reset only when the `resetStatistics` operation is executed on the MBean. To determine the state of the system without resetting the statistics, the Reporter uses a delta function. The delta function subtracts the prior value of a column from the current value of a column and returns the difference.

The prior values for a report are stored in a map on the Reporter client. This map is keyed by the "delta key". By default, the delta key is the MBean name for the attribute. However, when one-to-one relationship does not exist between the MBean and the rows in the report, or the MBean name is subject to change between executions of the report, the delta key will be calculated using the columns provided in the `<params>` section.

Note: Accuracy of Delta Functions: delta functions are only correct when the report is running as part of a report batch.

Delta Function Examples

Example 24–25 illustrates how to include a delta calculation of an attribute. (Assume `PacketsSent` is a defined column)

Example 24–25 Delta Calculation for an Attribute

```
<column id="DeltaPacketsSent">
  <type>function</type>
  <name>PacketsSent</name>
  <header>Delta Sent</header>
  <function-name>delta</function-name>
  <column-ref>PacketsSent</column-ref>
</column>
```

Example 24–26 illustrates how to include a delta calculation of an attribute with an alternate `delta` key. (Assume `PacketsSent`, `NodeID` and `TimeStamp` are defined columns)

Example 24–26 Delta Calculation for an Attribute with an Alternate Delta Key

```
<column id="DeltaPacketsSent">
  <type>function</type>
  <name>PacketsSent</name>
  <header>Delta Sent</header>
  <function-name>delta</function-name>
  <column-ref>PacketsSent</column-ref>
  <params>
     <column-ref>NodeID</column-ref>
     <column-ref>TimeStamp</column-ref>
  </params>
</column>
```

25

How to Modify Report Batch

Configuring a report batch is one of the steps in creating a custom report. You typically configure it after creating report configuration files. This configuration file determines what reports the reporter executes, how often the reports get executed, and where the reports are saved. If a single report can be used with different parameters, these parameters are also configured in the report batch. For more information on report configuration files, see Chapter 24, "How to Create a Custom Report".

Report Batch Deployment Descriptor

Use the report batch deployment descriptor to specify the various options for creating custom reports.

Document Location

The name and location of the descriptor defaults to `report-group.xml`. The default descriptor (packaged in `coherence.jar`) will be used unless a custom file is found in the application's classpath.

Document Root

The root element of the POF user type descriptor is report-group. This is where you may begin specifying the format of the custom report.

System Properties

Table 25–1 describes the system properties that can be used to control report batch from the command line.

Table 25–1 System Properties for Controlling Report Batch

Property	Default	Description
`tangosol.coherence.management.report.configuration`	`reports/report-group.xml`	The XML file containing the Reporter configuration settings, such as the list of reports, the report frequency, and so on.
`tangosol.coherence.management.report.autostart`	`false`	Flag to automatically start the reporter when the node is started.
`tangosol.coherence.management.report.distributed`	`false`	Determines if the reporter is running in a central model (`false`) or on every node in the cluster (`true`).

Document Format

The report batch descriptor should begin with the following `DOCTYPE` declaration:

```
<!DOCTYPE report-group SYSTEM "report-group.dtd">
```

Example 25–1 illustrates the nesting of elements in a report batch document.

Example 25–1 Format of a Report Batch Configuration File (report-group.xml)

```
<report group>
  <frequency/>
  <output-directory/>
  <report-list>
    <location/>
  <report-config>
    <init-params>
      <init-param>
      </init-param>
  <report-config/>
<report-group>
```

Report Batch Element Index

Table 25–2 describes the relationship between the report batch elements.

Table 25–2 Report Batch Elements

Element	Used in:
frequency	report-group
location	report-list
init-param	init-params
init-params	report-config
output-directory	report-group
param-name	init-param
param-type	init-param
param-value	init-param
report-config	report-group
report-group	*root element*
report-list	report-group

frequency

Used in: report-group

Description

Required. A string containing the number of seconds, minutes between each execution of the report batch. 10s will run the report ever 10 seconds. 5m will run the report every 5 minutes. Selecting an appropriate frequency is critical. If the frequency is too short, the reporter can generate a large amount of data and consume significant disk space. If the frequency is too long, the information will not be useful. It is recommended that a process for purging and archiving historical information is in place before configuring the reporter.

location

Used in: report-list

Description

Required. The path to the report configuration file. For more information on this file, see Chapter 24, "How to Create a Custom Report".

init-param

Used in: init-params

Description

The `init-param` element contains an initialization parameter for a report. The parameter consists of either a parameter name or type, and its value.

init-params

Used in: report-config

Description

Optional. The `init-params` element contains a list of initialization parameters.

output-directory

Used in: report-group

Description

Optional. The directory path to prepend to the output file names from the report configuration files. The username which the node is executing **must** have read write access to this path.

param-name

Used in: init-param

Description

The `param-name` element specifies the name of the initialization parameter.

param-type

Used in: init-param

Description

The `param-type` element specifies the Java type of the initialization parameter. Supported types are:

- `string`—indicates that the value is a `java.lang.String`
- `long`—indicates that the value is a `java.lang.Long`
- `double`—indicates that the value is a `java.lang.Double`

param-value

Used in: init-param

Description

The `param-value` element specifies a value of the initialization parameter. The value is in a format specific to the type of the parameter.

report-config

Used in: report-group

Description

The `report-config` contains the configuration file name and the initialization parameters for the report.

report-group

Used in: *root element*

Description

Describes the report list, the frequency, the report parameters, and the output directory for the batch.

report-list

Used in: report-group

Description

Required. The list of reports to include in the batch. This element contains the `<report-config>` subelement.

26

Analyzing Reporter Content

Coherence provides out of the box information that helps administrators and developers better analyze usage and configuration issues that may occur.

Network Health

The Network Health report contains the primary aggregates for determining the health of the network communications. The network health file is a tab delimited file that is prefixed with the date in `YYYYMMDD` format and post fixed with `-network-health.txt`. For example `20090131-network-health.txt` would be created on January 1, 2009. Table 26–1 describes the content of the Network Health report.

Table 26–1 Contents of the Network Health Report

Column	Type	Description
Batch Counter	Long	A sequential counter to help integrate information between related files. This value does reset when the reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
Report Time	Date	The system time when the report executed.
Min Node Rx Success	Double	The minimum receiver success rate for a node in the cluster. If this value is considerably less (10%) than the Grid Rx Success rate. Further analysis using the Network Health Detail should be done.
Grid Rx Success	Double	The receiver success rate for the grid as a whole. If this value is below 90%. Further analysis of the network health detail should be done.
Min Node Tx Success	Double	The minimum publisher success rate for a node in the cluster. If this value is considerably less (10%) than the Grid Rx Success rate. Further analysis using the Network Health Detail should be done.
Grid TX Success	Double	The publisher success rate for the grid as a whole. If this value is below 90%. Further analysis of the network health detail should be done.

Network Health Detail

The Network Health report supporting node level details for determining the health of the network communications. The network health detail file is a tab delimited file that is prefixed with the date in `YYYYMMDD` format and post fixed with `-network-health-detail.txt`. For example `20090131-network-health.txt`

would be created on January 1, 2009. Table 26–2 describes the content of the Network Health Detail report.

Table 26–2 Contents of the Network Health Detail Report

Column	Data Type	Description
Batch Counter	Long	A sequential counter to help integrate information between related files. This value does reset when the reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
Report Time	Date	The system time when the report executed.
Node Id	Long	The node for the network statistics.
Tx Success	Double	The publisher success rate for the node. If this value is within 2%-3% of the "Min Node Tx Success" and more than 10% less than the "Grid Tx Success" for the batch in the Network Health File, the corresponding node may be having difficulty communicating with the cluster. Constrained CPU, constrained network bandwidth or high network latency could cause this to occur.
RX Success	Double	The receiver success rate for the node. If this value is within 2%-3% of the "Min Node Rx Success" and more than 10% less than the "Grid Tx Success" for the batch in the Network Health File, the corresponding node may be having difficulty communicating with the cluster. Constrained CPU, constrained network bandwidth or high network latency could cause this to occur.
PacketsSent	Double	The total number of network packets sent by the node.
Current Packets Sent	Long	The number of packets sent by the node since the prior execution of the report.
PacketsResent	Long	The total number of network packets resent by the node. Packets will be resent when the receiver of the packet receives and invalid packet or when an acknowledge packet is not sent within the appropriate amount of time.
Current Packet Resent	Long	The number of network packets resent by the node since the prior execution of the report.
PacketsRepeated	Long	The total number of packets received more than once.
Current Packets Repeated	Long	The number of packets received since the last execution of the report.
PacketsReceived	Long	The total number of packets received by the node.
Current Packets Received	Long	The total number of packets received by the node since the last execution of the report.

Memory Status

The Memory Status report must be run as part of a report batch. The values are helpful in understanding memory consumption on each node and across the grid. For data to be included nodes must be configured to publish platform MBean information. The memory status file is a tab delimited file that is prefixed with the date in `YYYYMMDD` format and post fixed with `-memory-status.txt`. For example

20090131-memory-status.txt would be created on January 1, 2009. Table 26–3 describes the content of the Memory Status report.

Table 26–3 Contents of the Memory Status Report

Column	Data Type	Description
Batch Counter	Long	A sequential counter to help integrate information between related files. This value does reset when the reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
Report Time	Date	The system time when the report executed.
Node Id	Long	The node for the memory statistics.
Gc Name	String	The name of the Garbage Collector information.
CollectionCount	Long	The number of garbage collections that have happened since the virtual machine started.
Delta Collection Count	Long	The number of garbage collections that have occurred since the last execution of the report.
CollectTime	Long	The number of milliseconds the JVM has spent on garbage collection since the start of the JVM.
Delta Collect Time	Long	The number of milliseconds the JVM has spent on garbage collection since the last execution of the report.
Last GC Start Time	Long	The start time of the last Garbage Collection.
Last GC Stop Time	Long	The stop time of the last garbage collection.
Heap Committed	Long	The number of heap bytes committed at the time of report.
Heap Init	Long	The number of heap bytes initialized at the time of the report.
Heap Max	Long	The Maximum number of bytes used by the JVM since the start of the JVM.
Heap Used	Long	The bytes used by the JVM at the time of the report.

Cache Size

The cache size report can be executed either on demand or it can be added as part of the report batch and the Caches should have the <unit-calculator> subelement of <local-scheme> set to BINARY. The cache size file is a tab delimited file that is prefixed with the date in YYYYMMDD format and post fixed with -cache-size.txt. For example 20090131-cache-size.txt would be created on January 1, 2009. Table 26–4 describes the content of the Cache Size report.

Table 26–4 Contents of the Cache Size Report

Column	Data Type	Description
Batch Counter	Long	A sequential counter to help integrate information between related files. This value does reset when the reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
Report Time	Date	The system time when the report executed.
Cache Name	String	The name of the cache.

Table 26–4 (Cont.) Contents of the Cache Size Report

Column	Data Type	Description
MemoryMB	Double	The MB consumed by the objects in the cache. This does not include indexes or over head.
Avg Object Size	Double	The Average memory consumed by each object.
Cache Size	Double	The number of objects in the cache.
Memory Bytes	Double	The number of bytes consumed by the objects in the cache. This does not include indexes or over head.

Service Report

The service report provides information to the requests processed, request failures, and request backlog, tasks processed, task failures and task backlog. Request Count and Task Count are useful to determine performance and throughput of the service. RequestPendingCount and Task Backlog are useful in determining capacity issues or blocked processes. Task Hung Count, Task Timeout Count, Thread Abandoned Count, Request Timeout Count are the number of unsuccessful executions that have occurred in the system. Table 26–5 describes the contents of the Service report.

Table 26–5 Contents of the Service Report

Column	Data Type	Description
Batch Counter	Long	A sequential counter to help integrate information between related files. This value does reset when the reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
Report Time	Date	The system time when the report executed.
Service	String	The service name.
Node Id	String	The numeric node identifier.
Refresh Time	Date	The system time when the service information was updated from a remote node.
Request Count	Long	The number of requests since the last report execution.
RequestPendingCount	Long	The number of pending requests at the time of the report.
RequestPendingDuration	Long	The duration for the pending requests at the time of the report.
Request Timeout Count	Long	The number of request timeouts since the last report execution.
Task Count	Long	The number of tasks executed since the last report execution.
Task Backlog	Long	The task backlog at the time of the report execution.
Task Timeout Count	Long	The number of task timeouts since the last report execution.
Task Hung Count	Long	The number of tasks that hung since the last report execution.
Thread Abandoned Count	Long	The number of threads abandoned since the last report execution.

Node List

Due to the transient nature of the node identifier (nodeId), the reporter logs out a list of nodes and the user defined <member-identity> information. The node list file is a tab delimited file that is prefixed with the date in `YYYYMMDD` format and post fixed with `-nodes.txt`. For example `20090131-nodes.txt` would be created on January 1, 2009. Table 26–6 describes the content of the Node List report.

Table 26–6 Contents of the Node List Report

Column	Data Type	Description
Batch Counter	Long	A sequential counter to help integrate information between related files. This value does reset when the reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
Report Time	Date	The system time when the report executed.
Node Id	String	The numeric node identifier.
UnicastAddress	String	The Unicast address for the node.
MemberName	String	The member name for the node.
ProcessName	String	The process name for the node.
RoleName	String	The role name for the node.
MachineName	String	The machine name for the node.
RackName	String	The rack name for the node.
SiteName	String	The site name for the node.
Refresh Time	Date/Time	The time which the information was refreshed from a remote node. If the time is not the same as the refresh time on other rows in the batch, the node did not respond in a timely matter. This is often caused by a node preforming a garbage collection. Any information regarding a node with an "old" refresh date is questionable.

Proxy Report

The proxy file provides information about proxy servers and the information being transferred to clients. The Proxy file is a tab delimited file that is prefixed with the date in `YYYYMMDD` format and post fixed with `-report-proxy.txt`. For example `20090131-report-proxy.txt` would be created on January 1, 2009. Table 26–7 describes the content of the Proxy report.

Table 26–7 Contents of the Proxy Report

Column	Type	Description
Batch Counter	Long	A sequential counter to help integrate information between related files. This value does reset when the reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
Report Time	Date	The system time when the report executed.
Node Id	String	The numeric node identifier.
Service Name	String	The name of the proxy service.
HostIp	String	The IP Address and Port of the proxy service.

Table 26–7 (Cont.) Contents of the Proxy Report

Column	Type	Description
ConnectionCount	Long	The current number of connections to the proxy service.
OutgoingByteBacklog	Long	The number of bytes queued to be sent by the proxy service.
OutgoingMessageBacklog	Long	The number of messages queued by the proxy service.
Bytes Sent	Long	The number of bytes sent by the proxy service since the last execution of the report.
Bytes Received	Long	The number of bytes received by the proxy service since the last execution of the report.
Messages Sent	Long	The number of messages sent by the proxy service since the last execution of the report.
Messages Received	Long	The number of messages received by the proxy service since the last execution of the report.

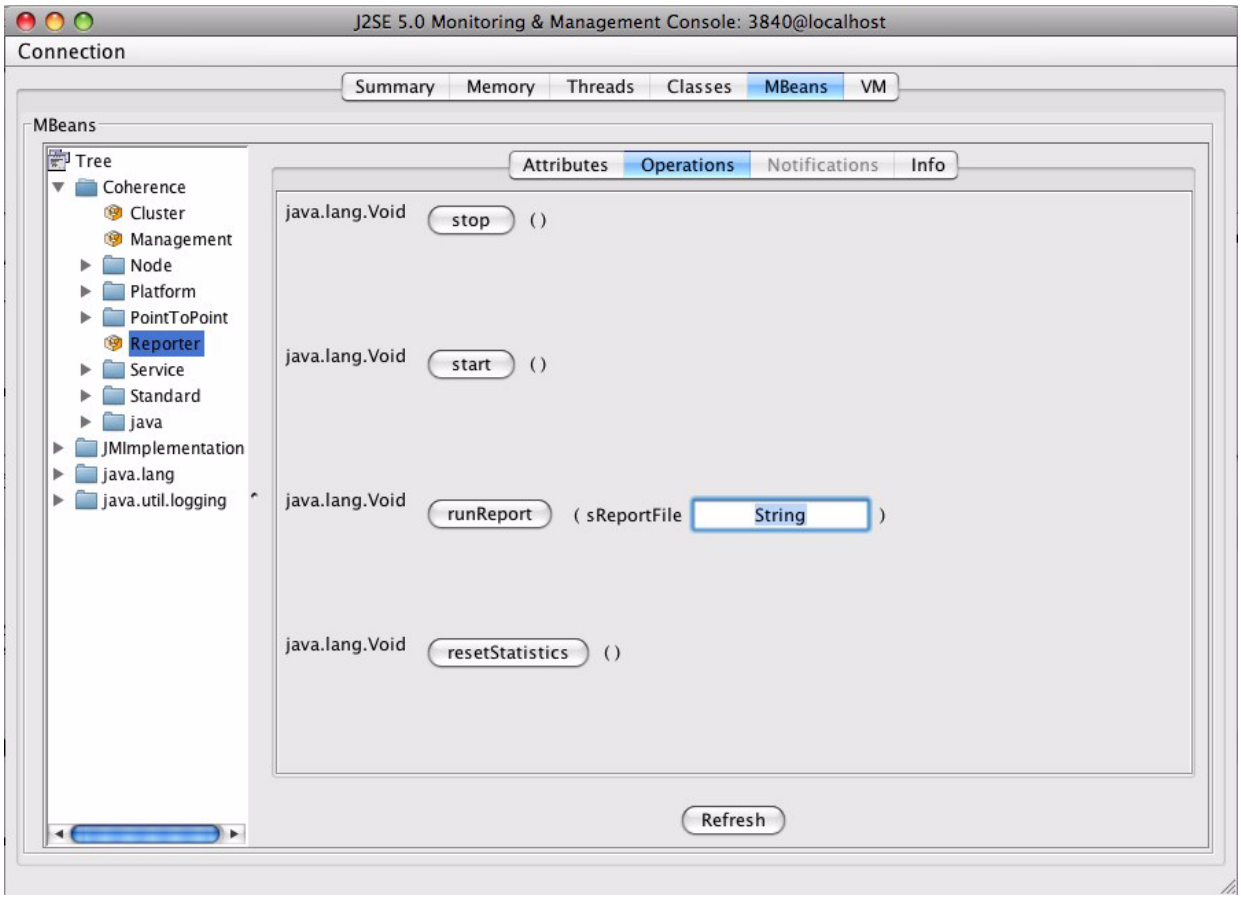
27

How to Run a Report on Demand

A report can be run on demand by using either JConsole or the JMX HTTP Adapter. The Reporter MBean operations contain a `runReport(String sReportPath)` method. The report path can either be a resource in `coherence.jar` or a file URL.

Figure 27–1 illustrates the reporter operations in JConsole.

Figure 27–1 Reporter Operations in JConsole



This figure is described in the text.

****

How to Run ReportControl MBean at Node Startup

When set to `true`, the `tangosol.coherence.management.report.autostart` system property allows the `ReportControl` MBean to start execution when then node is started. This property must be used with the `tangosol.coherence.management.report.group` system property and the configuration of the custom MBean XML file.

In Example 27–1, the `tangosol.coherence.management.report.autostart` system property is set to `true`.

Example 27–1 tangosol.coherence.management.report.autostart System Property

```
-Dtangosol.coherence.management.report.autostart=true
```

How to Configure the ReportControl MBean

The report group system property, `tangosol.coherence.management.report.group` configures the `ReportControl` MBean with the specified configuration file. This property must be used in correlation with the `tangosol.coherence.management.report.autostart` property and the configuration of the custom MBean XML file.

In Example 27–2, the `tangosol.coherence.management.report.group` property points to the custom MBean XML file `report-batch.xml`.

Example 27–2 tangosol.coherence.management.report.group System Property

```
-Dtangosol.coherence.management.report.group=./report-batch.xml
```

28

Configuring Custom MBeans

This chater provides information on configuring standard, MX, and JMX MBeans.

Creating an MBean XML Configuration File

Custom MBeans are configured in an XML configuration file. The elements in the file describe the MBean type, MBean implementation, and the target MBean `ObjectName`. The current release of Coherence supports these types of custom MBeans.

- Standard MBeans
- MXBeans
- JMX MBeans

See Appendix K, "MBean Configuration Elements" for a complete descriptions of the elements used in this chapter.

Configuring Standard MBeans

The configuration in Example 28–1 will create a `Coherence:type=Query,nodeId=<nodeId>` using the standard MBean `com.oracle.customMBeans.Query` class for the node. This example specifies an MBean class (`mbean-class`), an MBean name (`mbean-name`), and whether it is registered (`enabled`) in the instance.

Example 28–1 Using an MBean to Create a Query Node

```
<mbeans>
  <mbean id="100">
    <mbean-class>com.oracle.customMBeans.Query</mbean-class>
    <mbean-name>type=Query</mbean-name>
    <enabled>true</enabled>
  </mbean>
</mbeans>
```

Configuring MXBeans

The configuration in Example 28–2 will execute the standard Java method `getMemoryMXBean` in the `java.lang.management.ManagementFactory` class and use the result to create a `Coherence:type=java,SubSystem=Memory,nodeId=<nodeId>` for the node. The example specifies an MBean factory (`mbean-factory`), an accessor method name on the factory (`mbean-accessor`), an MBean name (`mbean-name`), and whether it is registered (`enabled`) in the instance.

Example 28–2 Getting an MBean for the Memory System of a Java Virtual Machine

```
<mbeans>
  <mbean id="2">
    <mbean-factory>java.lang.management.ManagementFactory</mbean-factory>
    <mbean-accessor>getMemoryMXBean</mbean-accessor>
    <mbean-name>type=java,SubSystem=Memory</mbean-name>
    <enabled>true</enabled>
  </mbean>
</mbeans>
```

Configuring JMX MBeans

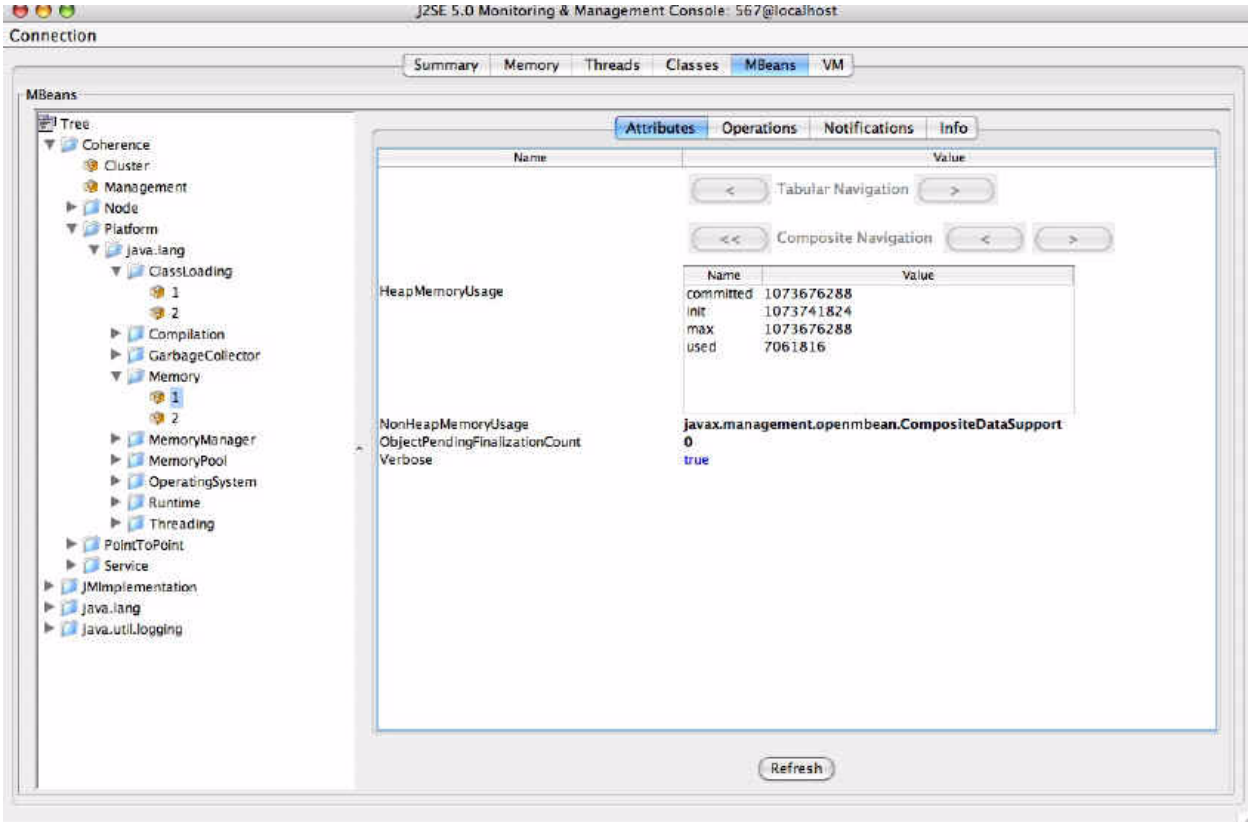
JMX MBeans are MBeans that exist in a local MBean server that need to be added to the Coherence Management structure. This allows consolidation of `MBeanServer` information into a single source. The configuration in Example 28–3 executes the JMX query, `java.lang:*`, on the node's local MBean server and uses the results to create corresponding MBeans on the centralized Coherence MBean server. The example specifies a JMX MBean query(`mbean-query`), an MBean name (`mbean-name`), and whether it is registered(`enabled`) in the instance.

Example 28–3 Executing a JMX Query and Creating an MBean on the MBean Server

```
<mbeans>
  <mbean id="1">
    <mbean-query>java.lang:*</mbean-query>
    <mbean-name>type=Platform</mbean-name>
    <enabled>true</enabled>
  </mbean>
<mbeans>
```

Figure 28–1 illustrates the results on the query in JConsole.

Figure 28–1 MBean Query Displayed in the JConsole



This figure is described in the text.

****

Enabling a Custom MBean Configuration File

You can enable the custom MBean configuration file by setting a system property or by including a specially named file in the class path.

Setting a System Property

Coherence provides the following system property to specify the name and location of a custom MBean configuration file. Setting this system property will cause the Coherence node to load the MBeans defined in the file represented by `filename`.

Example 28–4 System Property to Load an MBean

```
-Dtangosol.coherence.mbeans=<filename>
```

Adding a Custom MBean Configuration File to the Class Path

By convention, Coherence recognizes the configuration file named `custom-mbeans.xml` as containing a custom MBean configuration. If you name your custom MBean configuration file `custom-mbeans.xml` and include it in the class path, then the Coherence node will load the configured MBeans.

29

How to Manage Custom MBeans Within the Cluster

In addition to managing Coherence with JMX, Coherence provides the ability to manage and monitor "custom MBeans" (that is, application-level MBeans) within the Coherence JMX Management and Monitoring framework. This enables you to manage or monitor any application-level MBean from any JVM, node, or end-point within the cluster.

In addition to the standard Coherence managed object types, any dynamic or standard MBean type may be registered using the com.tangosol.net.management.Registry interface.

Custom MBean Configuration

Coherence 3.4 can be configured to load platform and standard MBeans on connection to the cluster. This allows administrators and support personnel to update and view system and application information from all nodes in a cluster from a single location. This feature also eliminates the need for JMX programs to connect to multiple sources to gather information.

How to Add a Standard MBean to Coherence

The following instructions describe how to add a standard MBean to Coherence:

1. Create a standard MBean.
2. Add a standard MBean Class or JAR to the Coherence classpath (including central management node).
3. Create a custom MBean XML configuration file (see "Creating an MBean XML Configuration File" on page 28-1).
4. Modify node startup scripts to reference `custom-mbean.xml` (see "Enabling a Custom MBean Configuration File" on page 28-3.

How to Programatically Add a Standard MBean to Coherence

Example 29–1 illustrates sample code that programmatically adds a standard MBean to Coherence.

Example 29–1 Adding a Standard MBean to Coherence Programatically

```
Registry    registry = CacheFactory.ensureCluster().getManagement();
Custom      bean     = new Custom();
String      sName    = registry.ensureGlobalName("type=Custom");
```

```
registry.register(sName, bean);
```

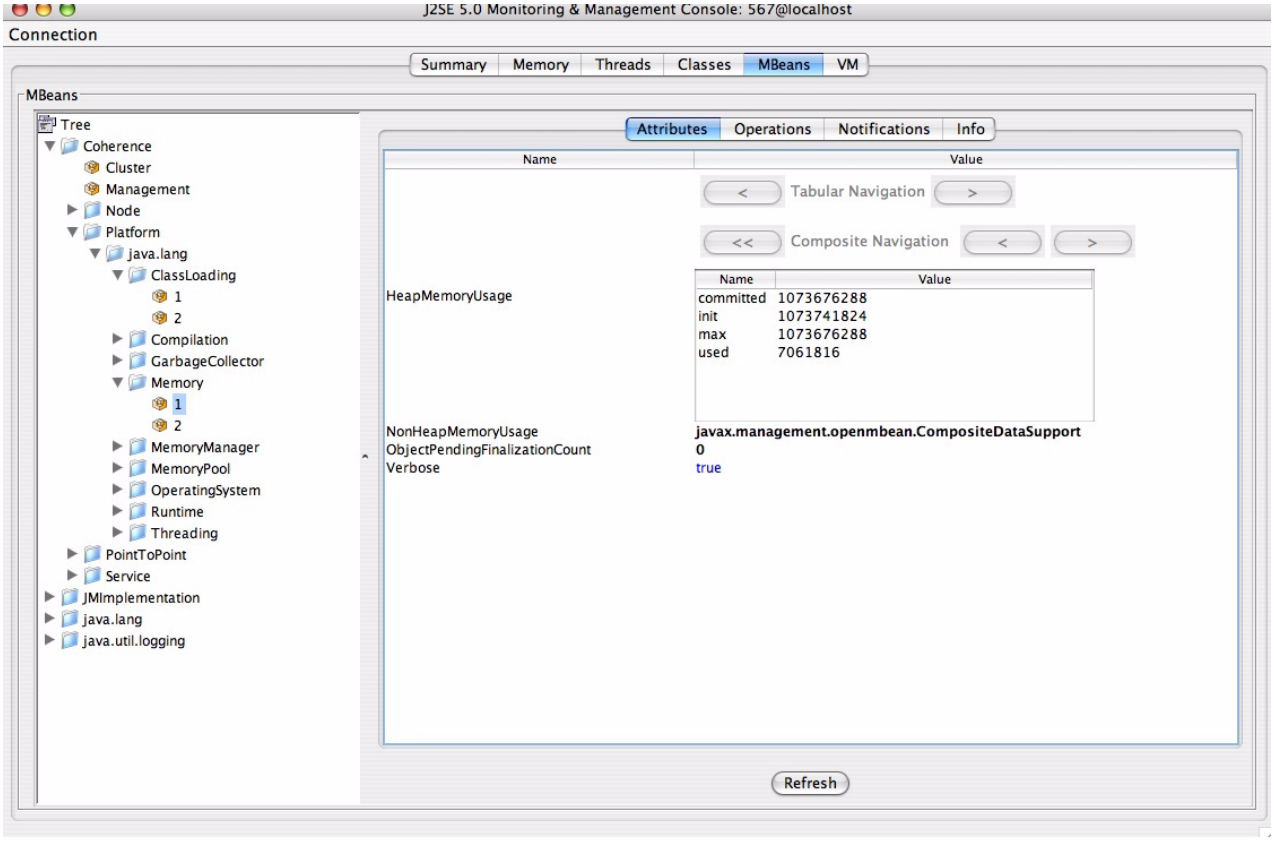
How to Add a the Results of a JMX Query to Coherence

The following instructions describe how to add the results of a JMX query to Coherence.

1. Create a custom MBean XML file (see "Creating an MBean XML Configuration File" on page 28-1).
2. Configure node startup script to include JMX MBean Server
3. Configure a node startup script to reference custom-mbean.xml (see "Enabling a Custom MBean Configuration File" on page 28-3).

Figure 29–1 illustrates an example of running a JMX Query in JConsole.

Figure 29–1 JMX Query Run in JConsole



This figure is described in the text.

****

A

Production Checklist

Note: Deploying Coherence in a production environment is very different from using Coherence in a development environment.

Development environments do not reflect the challenges of a production environment.

Coherence tends to be so simple to use in development that developers do not take the necessary planning steps and precautions when moving an application using Coherence into production. This article is intended to accomplish the following:

- Create a healthy appreciation for the complexities of deploying production software, particularly large-scale infrastructure software and enterprise applications;
- Enumerate areas that require planning when deploying Coherence;
- Define why production awareness should exist for each of those areas;
- Suggest or require specific approaches and solutions for each of those areas; and
- Provide a check-list to minimize risk when deploying to production.

Deployment recommendations are available for:

- Network
- Hardware
- Operating System
- JVM
- Java Security Manager
- Application Instrumentation
- Coherence Editions and Modes
- Coherence Operational Configuration
- Coherence Cache Configuration
- Large Cluster Configuration
- Death Detection

Network

During development, a Coherence-enabled application on a developer's local machine can accidentally form a cluster with the application running on other developers' machines.

Developers often use and test Coherence locally on their workstations. There are several ways in which they may accomplish this, including:

- Setting the multicast TTL to zero,
- Using a "loopback", or
- By each developer using a different multi-cast address and port from all other developers.

If one of these approaches is not used, then multiple developers on the same network will find that Coherence has clustered across different developers' locally running instances of the application; in fact, this happens relatively often and causes confusion when it is not understood by the developers.

Setting the TTL to zero on the command line is very simple: Add the following to the JVM startup parameters:

```
-Dtangosol.coherence.ttl=0
```

Starting with Coherence version 3.2, setting the TTL to zero for all developers is also very simple. Edit the `tangosol-coherence-override-dev.xml` in the `coherence.jar` file, changing the TTL setting as follows:

```
<time-to-live system-property="tangosol.coherence.ttl">0</time-to-live>
```

On some UNIX operating systems, including some versions of Linux and Mac OS X, setting the TTL to zero may not be enough to isolate a cluster to a single machine. To be safe, assign a different cluster name for each developer, for example using the developer's email address as the cluster name. If the cluster communication does go across the network to other developer machines, then the different cluster name will cause an error on the node that is attempting to start up.

To ensure that the clusters are completely isolated, select a different multicast IP address and port for each developer. In some organizations, a simple approach is to use the developer's phone extension number as part of the multicast address and as the port number (or some part of it). For information on configuring the multicast IP address and port, see "multicast-listener" on page H-26.

During development, clustered functionality is often not being tested.

After the POC or prototype stage is complete, and until load testing begins, it is not out of the ordinary for the application to be developed and tested by engineers in a non-clustered form. This is dangerous, as testing primarily in the non-clustered configuration can hide problems with the application architecture and implementation that will show up later in staging, or even production.

Make sure that the application is being tested in a clustered configuration as development proceeds. There are several ways for clustered testing to be a natural part of the development process; for example:

- Developers can test with a locally clustered configuration (at least two instances running on their own machine). This works well with the `TTL=0` setting, since clustering on a single machine works with the `TTL=0` setting.

- Unit and regression tests can be introduced that run in a test environment that is clustered. This may help automate certain types of clustered testing that an individual developer would not always remember (or have the time) to do.

What is the type and speed of the production network?

Most production networks are based on gigabit Ethernet, with a few still built on slower 100Mb Ethernet or faster ten-gigabit Ethernet. It is important to understand the topology of the production network, and what the full set of devices that will connect all of the servers that will be running Coherence. For example, if there are ten different switches being used to connect the servers, are they all the same type (make and model) of switch? Are they all the same speed? Do the servers support the network speeds that are available?

In general, all servers should share a reliable, fully switched network. This generally implies sharing a single switch (ideally, two parallel switches and two network cards per server for availability). There are two primary reasons for this. The first is that using more than one switch almost always results in a reduction in effective network capacity. The second is that multi-switch environments are more likely to have network "partitioning" events where a partial network failure will result in two or more disconnected sets of servers. While partitioning events are rare, Coherence cache servers ideally should share a common switch.

To demonstrate the impact of multiple switches on bandwidth, consider several servers plugged into a single switch. As additional servers are added, each server receives dedicated bandwidth from the switch backplane. For example, on a fully switched gigabit backplane, each server receives a gigabit of inbound bandwidth and a gigabit of outbound bandwidth for a total of 2Gbps "full duplex" bandwidth. Four servers would have an aggregate of 8Gbps bandwidth. Eight servers would have an aggregate of 16Gbps. And so on up to the limit of the switch (in practice, usually in the range of 160-192Gbps for a gigabit switch). However, consider the case of two switches connected by a 4Gbps (8Gbps full duplex) link. In this case, as servers are added to each switch, they will have full "mesh" bandwidth up to a limit of four servers on each switch (e.g all four servers on one switch can communicate at full speed with the four servers on the other switch). However, adding additional servers will potentially create a bottleneck on the inter-switch link. For example, if five servers on one switch send data to five servers on the other switch at 1Gbps per server, then the combined 5Gbps will be restricted by the 4Gbps link. Note that the actual limit may be much higher depending on the traffic-per-server and also the portion of traffic that actually needs to move across the link. Also note that other factors such as network protocol overhead and uneven traffic patterns may make the usable limit much lower from an application perspective.

Avoid mixing and matching network speeds: Make sure that all servers can **and do** connect to the network at the same speed, and that all of the switches and routers between those servers run at that same speed or faster.

Oracle strongly suggests GigE or faster: Gigabit Ethernet is supported by most servers built since 2004, and Gigabit switches are economical, available and widely deployed.

Before deploying an application, you must run the Datagram Test to test the actual network speed and determine its capability for pushing large amounts of data. Furthermore, the Datagram test must be run with an increasing ratio of publishers to consumers, since a network that appears fine with a single publisher and a single consumer may completely fall apart as the number of publishers increases, such as occurs with the default configuration of Cisco 6500 series switches. See "Deploying to Cisco Switches" on page M-2 for more information.

Will the production deployment use multicast?

The term "multicast" refers to the ability to send a packet of information from one server and to have that packet delivered in parallel by the network to many servers. Coherence supports both multicast and multicast-free clustering. Oracle *suggests* the use of multicast when possible because it is an efficient option for many servers to communicate. However, there are several common reasons why multicast cannot be used:

- Some organizations disallow the use of multicast.
- Multicast cannot operate over certain types of network equipment; for example, many WAN routers disallow or do not support multicast traffic.
- Multicast is occasionally unavailable for technical reasons; for example, some switches do not support multicast traffic.

First determine if multicast will be used. In other words, determine if the desired deployment configuration is to use multicast.

Before deploying an application that will use multicast, you must run the Multicast Test to verify that multicast is working and to determine the correct (the minimum) TTL value for the production environment. See Chapter 16, "Performing a Multicast Connectivity Test" for more information.

Applications that cannot use multicast for deployment must use the WKA configuration. See "well-known-addresses" on page H-51 and "Network Protocols" for more information.

Are your network devices configured optimally?

If the above datagram and/or multicast tests have failed or returned poor results, it is possible that there are configuration problems with the network devices in use. Even if the tests passed without incident and the results were perfect, it is still possible that there are lurking issues with the configuration of the network devices.

Review the suggestions in "Network Tuning" on page 20-4.

How will the cluster handle a sustained network outage?

The Coherence cluster protocol is capable of detecting and handling a wide variety of connectivity failures. The clustered services are able to identify the connectivity issue, and force the offending cluster node to leave and re-join the cluster. In this way the cluster ensures a consistent shared state among its members.

See "Death Detection" on page A-14 for more details. See also:

- "Deploying to Cisco Switches" on page M-2
- "Deploying to Foundry Switches" on page M-4

Hardware

During development, developers can form unrealistic performance expectations.

Most developers have relatively fast workstations. Combined with test cases that are typically non-clustered and tend to represent single-user access (that is, only the developer), the application may seem extraordinarily responsive.

Include as a requirement that realistic load tests be built that can be run with simulated concurrent user load.

Test routinely in a clustered configuration with simulated concurrent user load.

During development, developer productivity can be adversely affected by inadequate hardware resources, and certain types of quality can also be affected negatively.

Coherence is compatible with all common workstation hardware. Most developers use PC or Apple hardware, including notebooks, desktops and workstations.

Developer systems should have a significant amount of RAM to run a modern IDE, debugger, application server, database and at least two cluster instances. Memory utilization varies widely, but to ensure productivity, the suggested minimum memory configuration for developer systems is 2GB. Desktop systems and workstations can often be configured with 4GB for minimal additional cost.

Developer systems should have two CPU cores or more. Although this will have the likely side-effect of making developers happier, the actual purpose is to increase the quality of code related to multi-threading, since many bugs related to concurrent execution of multiple threads will only show up on multi-CPU systems (systems that contain multiple processor sockets and/or CPU cores).

What are the supported and suggested server hardware platforms for deploying Coherence on?

The short answer is that Oracle works to support the hardware that the customer has standardized on or otherwise selected for production deployment.

- Oracle has customers running on virtually all major server hardware platforms. The majority of customers use "commodity x86" servers, with a significant number deploying Sun Sparc (including Niagra) and IBM Power servers.
- Oracle continually tests Coherence on "commodity x86" servers, both Intel and AMD.
- Intel, Apple and IBM provide hardware, tuning assistance and testing support to Oracle.
- Oracle conducts internal Coherence certification on all IBM server platforms at least once a year.
- Oracle and Azul test Coherence regularly on Azul appliances, including the newly-announced 48-core "Vega 2" chip.

If the server hardware purchase is still in the future, the following are suggested for Coherence (as of December 2006):

The most cost-effective server hardware platform is "commodity x86", either Intel or AMD, with one to two processor sockets and two to four CPU cores per processor socket. If selecting an AMD Opteron system, it is strongly recommended that it be a two processor socket system, since memory capacity is usually halved in a single socket system. Intel "Woodcrest" and "Clovertown" Xeons are **strongly** recommended over the previous Intel Xeon CPUs due to significantly improved 64-bit support, much lower power consumption, much lower heat emission and far better performance. These new Xeons are currently the fastest commodity x86 CPUs, and can support a large memory capacity per server regardless of the processor socket count by using fully buffered memory called "FB-DIMMs".

It is strongly recommended that servers be configured with a minimum of 4GB of RAM. For applications that plan to store massive amounts of data in memory - tens or hundreds of gigabytes, or more - it is recommended to evaluate the cost-effectiveness of 16GB or even 32GB of RAM per server. As of December, 2006, commodity x86 server RAM is readily available in a density of 2GB per DIMM, with higher densities available from only a few vendors and carrying a large price premium; this means that a server with 8 memory slots will only support 16GB in a cost-effective manner. Also

note that a server with a very large amount of RAM will likely need to run more Coherence nodes (JVMs) per server to use that much memory, so having a larger number of CPU cores will help. Applications that are "data heavy" will require a higher ratio of RAM to CPU, while applications that are "processing heavy" will require a lower ratio. For example, it may be sufficient to have two dual-core Xeon CPUs in a 32GB server running 15 Coherence "Cache Server" nodes performing mostly identity-based operations (cache accesses and updates), but if an application makes frequent use of Coherence features such as indexing, parallel queries, entry processors and parallel aggregation, then it will be more effective to have two quad-core Xeon CPUs in a 16GB server - a 4:1 increase in the CPU:RAM ratio.

A minimum of 1000Mbps for networking (for example, Gigabit Ethernet or better) is **strongly** recommended. NICs should be on a high bandwidth bus such as PCI-X or PCIe, and not on standard PCI. In the case of PCI-X having the NIC on an isolated or otherwise lightly loaded 133MHz bus may significantly improve performance.

How many servers are optimal?

Coherence is primarily a scale-out technology. While Coherence can effectively scale-up on large servers by using multiple JVMs per server, the natural mode of operation is to span several small servers (for example, 2-socket or 4-socket commodity servers). Specifically, failover and failback are more efficient in larger configurations. And the impact of a server failure is lessened. As a rule of thumb, a cluster should contain at least four physical servers. In most WAN configurations, each data center will have independent clusters (usually interconnected by Extend-TCP). This will increase the total number of discrete servers (four servers per data center, multiplied by the number of data centers).

Coherence is quite often deployed on smaller clusters (one, two or three physical servers) but this practice has increased risk if a server failure occurs under heavy load. As discussed in the network section of this document, Coherence clusters are ideally confined to a single switch (for example, fewer than 96 physical servers). In some use cases, applications that are compute-bound or memory-bound applications (as opposed to network-bound) may run acceptably on larger clusters.

Also note that given the choice between a few large JVMs and a lot of small JVMs, the latter may be the better option. There are several production environments of Coherence that span hundreds of JVMs. Some care is required to properly prepare for clusters of this size, but smaller clusters of dozens of JVMs are readily achieved. Please note that disabling UDP multicast (by using WKA) or running on slower networks (for example, 100Mbps Ethernet) will reduce network efficiency and make scaling more difficult.

Does it matter how JVMs are distributed among servers?

The following rules should be followed in determining how many servers are required for reliable high availability configuration and how to configure the number of *storage-enabled* JVMs.

1. There must be more than two servers. A grid with only two servers stops being machine-safe as soon as several JVMs on one server is not the same as the number of JVMs on the other server, so even if we start with two servers with equal number of JVMs, losing one JVM will force the grid out of machine-safe state. Four or more machines present the most stable topology, but deploying on just three servers would work if the other rules are adhered to.
2. For a server that has the largest number of JVMs in the cluster, that number of JVMs must not exceed the total number of JVMs on all the other servers in the cluster.

3. A server with the smallest number of JVMs should run at least half the number of JVMs as a server with the largest number of JVMs; this rule is particularly important for smaller clusters.
4. The margin of safety improves as the number of JVMs tends toward equality on all machines in the cluster; this is more of a "rule of thumb" than the preceding "hard" rules.

See also:

- "Deploying to IBM BladeCenters" on page M-4
- "Deploying to Virtual Machines" on page M-8

Operating System

During development, developers typically use a different operating system than the one that the application will be deployed to.

The top three operating systems for application development using Coherence are, in this order: Windows 2000/XP (~85%), Mac OS X (~10%) and Linux (~5%). The top four operating systems for production deployment are, in this order: Linux, Solaris, AIX and Windows. Thus, it is relatively unlikely that the development and deployment operating system will be the same.

Make sure that regular testing is occurring on the target operating system.

What are the supported and suggested server operating systems for deploying Coherence on?

Oracle tests on and supports various Linux distributions (including customers that have custom Linux builds), Sun Solaris, IBM AIX, Windows Vista/2003/2000/XP, Apple Mac OS X, OS/400 and z/OS. Additionally, Oracle supports customers running HP-UX and various BSD UNIX distributions.

If the server operating system decision is still in the future, the following are suggested for Coherence (as of December 2006):

For commodity x86 servers, Linux distributions based on the Linux 2.6 kernel are recommended. While it is expected that most 2.6-based Linux distributions will provide a good environment for running Coherence, the following are recommended by Oracle: RedHat Enterprise Linux (version 4 or later) and Suse Linux Enterprise (version 10 or later). Oracle also routinely tests using distributions such as RedHat Fedora Core 5 and even Knoppix "Live CD".

Review and follow the instructions in Appendix M, "Platform-Specific Deployment Considerations" for the operating system that Coherence will be deployed on.

Avoid using virtual memory (paging to disk).

In a Coherence-based application, primary data management responsibilities (for example, Dedicated Cache Servers) are hosted by Java-based processes. Modern Java distributions do not work well with virtual memory. In particular, garbage collection (GC) operations may slow down by several orders of magnitude if memory is paged to disk. With modern commodity hardware and a modern JVM, a Java process with a reasonable heap size (512MB-2GB) will typically perform a full garbage collection in a few seconds if all of the process memory is in RAM. However, this may grow to many minutes if the JVM is partially resident on disk. During garbage collection, the node will appear unresponsive for an extended period, and the choice for the rest of the cluster is to either wait for the node (blocking a portion of application activity for a

corresponding amount of time), or to mark the unresponsive node as "failed" and perform failover processing. Neither of these is a good option, and so it is important to avoid excessive pauses due to garbage collection. JVMs should be pinned into physical RAM, or at least configured so that the JVM will not be paged to disk.

Note that periodic processes (such as daily backup programs) may cause memory usage spikes that could cause Coherence JVMs to be paged to disk.

See also:

- "Deploying to AIX" on page M-1
- "Deploying to Linux" on page M-5
- "Deploying to OS X" on page M-6
- "Deploying to Solaris" on page M-7
- "Deploying to Windows" on page M-8
- "Deploying to z OS" on page M-9

JVM

During development, developers typically use the latest Sun JVM or a direct derivative such as the Mac OS X JVM.

The main issues related to using a different JVM in production are:

- Command line differences, which may expose problems in shell scripts and batch files;
- Logging and monitoring differences, which may mean that tools used to analyze logs and monitor live JVMs during development testing may not be available in production;
- Significant differences in optimal GC configuration and approaches to GC tuning;
- Differing behaviors in thread scheduling, garbage collection behavior and performance, and the performance of running code.

Make sure that regular testing is occurring on the JVM that will be used in production.

Which JVM configuration options should be used?

JVM configuration options vary over versions and between vendors, but the following are generally suggested:

- Using the `-server` option will result in substantially better performance.
- Using identical heap size values for both `-Xms` and `-Xmx` will yield substantially better performance, and "fail fast" memory allocation.
- For naive tuning, a heap size of 512MB is a good compromise that balances per-JVM overhead and garbage collection performance.
    - Larger heap sizes are allowed and commonly used, but may require tuning to keep garbage collection pauses manageable.

What are the supported and suggested JVMs for deploying Coherence on?

In terms of Oracle Coherence versions:

- Coherence 3.x versions are supported on the Sun JDK versions 1.4 and 1.5, and JVMs corresponding to those versions of the Sun JDK. Starting with Coherence 3.4 the 1.6 JVMs are also supported.

- Coherence version 2.x (currently at the 2.5.1 release level) is supported on the Sun JDK versions 1.2, 1.3, 1.4 and 1.5, and JVMs corresponding to those versions of the Sun JDK.

Often the choice of JVM is dictated by other software. For example:

- IBM only supports IBM WebSphere running on IBM JVMs. Most of the time, this is the IBM "Sovereign" or "J9" JVM, but when WebSphere runs on Sun Solaris/Sparc, IBM builds a JVM using the Sun JVM source code instead of its own.
- BEA WebLogic typically includes a JVM which is intended to be used with it. On some platforms, this is the BEA WebLogic JRockit JVM.
- Apple Mac OS X, HP-UX, IBM AIX and other operating systems only have one JVM vendor (Apple, HP and IBM respectively).
- Certain software libraries and frameworks have minimum Java version requirements because they take advantage of relatively new Java features.

On commodity x86 servers running Linux or Windows, the Sun JVM is recommended. Generally speaking, the recent update versions are recommended. For example:

- Oracle recommends testing and deploying using the latest supported Sun JVM based on your platform and Coherence version.

Basically, at some point before going to production, a JVM vendor and version should be selected and well tested, and absent any flaws appearing during testing and staging with that JVM, that should be the JVM that is used when going to production. For applications requiring continuous availability, a long-duration application load test (for example, at least two weeks) should be run with that JVM before signing off on it.

Review and follow the instructions in Appendix M, "Platform-Specific Deployment Considerations" for the JVM that Coherence will be deployed on.

Must all nodes run the same JVM vendor and version?

No. Coherence is pure Java software and can run in clusters composed of any combination of JVM vendors and versions, and Oracle tests such configurations.

Note that it is *possible* for different JVMs to have slightly different serialization formats for Java objects, meaning that it is possible for an incompatibility to exist when objects are serialized by one JVM, passed over the wire, and a different JVM (vendor and/or version) attempts to deserialize it. Fortunately, the Java serialization format has been very stable for several years, so this type of issue is extremely unlikely. However, it is highly recommended to test mixed configurations for consistent serialization before deploying in a production environment.

See also:

- "Deploying to BEA JRockit JVMs" on page M-2
- "Deploying to IBM JVMs" on page M-5
- "Deploying to Sun JVMs" on page M-7

Java Security Manager

The minimum set of privileges required for Coherence to function are specified in the security.policy file which is included as part of the Coherence installation. This file can be found in `coherence/lib/security/security.policy`. If using the Java Security Manager these privileges must be granted in order for Coherence to function properly.

Application Instrumentation

Be cautious when using instrumented management and monitoring solutions.

Some Java-based management and monitoring solutions use instrumentation (for example, bytecode-manipulation and `ClassLoader` substitution). While there are no known open issues with the latest versions of the primary vendors, Oracle has observed issues in the past.

Coherence Editions and Modes

During development, use the development mode.

The Coherence download includes a fully functional Coherence product supporting all editions and modes. The default configuration is for Grid Edition in Development mode.

Coherence may be configured to operate in either development or production mode. These modes do not limit access to features, but instead alter some default configuration settings. For instance, development mode allows for faster cluster startup to ease the development process.

It is recommended to use the development mode for all pre-production activities, such as development and testing. This is an important safety feature, because Coherence automatically prevents these nodes from joining a production cluster. The production mode must be explicitly specified when using Coherence in a production environment.

Coherence may be configured to support a limited feature set, based on the customer license agreement.

Only the edition and the number of licensed CPUs specified within the customer license agreement can be used in a production environment.

When operating outside of the production environment it is allowable to run any Coherence edition. However, it is recommended that only the edition specified within the customer license agreement be used. This will protect the application from unknowingly making use of unlicensed features.

All nodes within a cluster must use the same license edition and mode.

Starting with Oracle Coherence 3.4, customer-specific license keys are no longer part of product deployment.

Be sure to obtain enough licenses for the all the cluster members in the production environment. The servers hardware configuration (number or type of processor sockets, processor packages or CPU cores) may be verified using `ProcessorInfo` utility included with Coherence.

Example A–1 Verifying Hardware Configuration

```
java -cp tangosol.jar com.tangosol.license.ProcessorInfo
```

If the result of the `ProcessorInfo` program differs from the licensed configuration, send the program's output and the actual configuration to the "support" email address at Oracle.

How are the edition and mode configured?

There is a <license-config> configuration section in `tangosol-coherence.xml` (located in `coherence.jar`) for edition and mode related information.

Example A–2 Sample Coherence License Configuration

```
<license-config>
    <edition-name system-property="tangosol.coherence.edition">GE</edition-name>
    <license-mode system-property="tangosol.coherence.mode">dev</license-mode>
  </license-config>
```

In addition to preventing mixed mode clustering, the `license-mode` also dictates the operational override file which will be used. When in `dev` mode the `tangosol-coherence-override-dev.xml` file will be used, whereas the `tangosol-coherence-override-prod.xml` file will be used when the `prod` mode is specified. As the mode controls which override file is used, the `<license-mode>` configuration element is only usable in the base `tangosol-coherence.xml` file and not within the override files.

These elements are defined by the corresponding `coherence.dtd` in `coherence.jar`. It is possible to specify this edition on the command line using the command line override:

```
-Dtangosol.coherence.edition=RTC
```

Valid values are listed in Table A–1:

Table A–1 Valid tangosol.coherence.edition Values

Value	Coherence Edition	Compatible Editions
GE	Grid Edition	RTC, DC
EE	Enterprise Edition	DC
SE	Standard Edition	DC
RTC	Real-Time Client	GE
DC	Data Client	GE, EE, SE

- Note: clusters running different editions may connect by using Coherence*Extend as a Data Client.

For more information on overrides, see Appendix L, "Command Line Overrides".

Ensuring that RTC nodes don't use Coherence TCMP

The RTC nodes can connect to clusters using either Coherence TCMP or Coherence Extend. If the intention is to connect over Extend it is advisable to disable TCMP on that node to ensure that it only connects by using Extend. TCMP may be disabled using the system property `tangosol.coherence.tcmp.enabled`. See the `<enabled>` subelement of "packet-publisher" on page H-36.

Coherence Operational Configuration

Operational configuration relates to the configuration of Coherence at the cluster level including such things as:

- Cluster and member descriptors

- Network settings
- Security
  - Membership restrictions
  - Access Control
  - Encryption

The operational aspects are normally configured by using the `tangosol-coherence-override.xml` file. See "Operational Configuration Deployment Descriptors" on page H-1 for more information on this file.

The contents of this file will likely differ between development and production. It is recommended that these variants be maintained independently due to the significant differences between these environments. The production operational configuration file should not be the responsibility of the application developers, instead it should fall under the jurisdiction of the systems administrators who are far more familiar with the workings of the production systems.

All cluster nodes should use the same operational configuration descriptor. A centralized configuration file may be maintained and accessed by specifying the file's location as a URL using the tangosol.coherence.override system property. Any node specific values may be specified by using system properties. See Appendix L, "Command Line Overrides" for more information on the properties.

The override file should contain only the subset of configuration elements which you want to customize. This will not only make your configuration more readable, but will allow you to take advantage of updated defaults in future Coherence releases. All override elements should be copied exactly from the original tangosol-coherence.xml, including the id attribute of the element.

Member descriptors may be used to provide detailed identity information that is useful for defining the location and role of the cluster member. Specifying these items will aid in the management of large clusters by making it easier to identify the role of a remote nodes if issues arise.

Coherence Cache Configuration

Cache configuration relates to the configuration of Coherence at a per-cache level including such things as:

- Cache topology (`<distributed-scheme>`, `<replicated-scheme>`, `<near-scheme>`, and so on)
- Cache capacities (see `<high-units>` subelement of `<local-scheme>`)
- Cache redundancy level (`<backup-count>` subelement of `<distributed-scheme>`)

The cache configuration aspects are normally configured by using the `coherence-cache-config.xml` file. See "Cache Configuration Deployment Descriptor" on page D-1 for more information this file.

The default `coherence-cache-config.xml` file included within `coherence.jar` is intended only as an example and is not suitable for production use. It is suggested that you produce your own cache configuration file with definitions tailored to your application needs.

All cluster nodes should use the same cache configuration descriptor. A centralized configuration file may be maintained and accessed by specifying the file's location as a URL using the `tangosol.coherence.cacheconfig` system property.

Choose the cache topology which is most appropriate for each cache's usage scenario.

It is important to size limit your caches based on the allocated JVM heap size. Even if you never expect to fully load the cache, having the limits in place will help protect your application from `OutOfMemoryExceptions` if your expectations are later negated.

For a 1GB heap that at most ¾ of the heap be allocated for cache storage. With the default one level of data redundancy this implies a per server cache limit of 375MB for primary data, and 375MB for backup data. The amount of memory allocated to cache storage should fit within the tenured heap space for the JVM. See Sun's GC tuning guide for details.

It is important to note that when multiple cache schemes are defined for the same cache service name the first to be loaded will dictate the service level parameters. Specifically the `<partition-count>`, `<backup-count>`, and `<thread-count>` subelements of <distributed-scheme> are shared by all caches of the same service.

For multiple caches which use the same cache service it is recommended that the service related elements be defined only once, and that they be inherited by the various cache-schemes which will use them.

If you want different values for these items on a cache by cache basis then multiple services may be configured.

For partitioned caches Coherence will evenly distribute the storage responsibilities to all cache servers, regardless of their cache configuration or heap size. For this reason it is recommended that all cache server processes be configured with the same heap size. For machines with additional resources multiple cache servers may be used to effectively make use of the machine's resources.

To ensure even storage responsibility across a partitioned cache the `<partition-count>` subelement of <distributed-scheme>, should be set to a prime number which is at least the square of the number of cache servers which will be used.

For caches which are backed by a cache store it is recommended that the parent service be configured with a thread pool as requests to the cache store may block on I/O. The pool is enabled by using the `<thread-count>` subelement of <distributed-scheme> element. For non-CacheStore-based caches more threads are unlikely to improve performance and should left disabled.

Unless explicitly specified all cluster nodes will be storage enabled, that is, will act as cache servers. It is important to control which nodes in your production environment will be storage enabled and storage disabled. The `tangosol.coherence.distributed.localstorage` system property may be used to control this, setting it to either `true` or `false`. Generally, only dedicated cache servers, all other cluster nodes should be configured as storage disabled. This is especially important for short lived processes which may join the cluster perform some work, and exit the cluster, having these nodes as storage disable will introduce unneeded re-partitioning. See the `<local-storage>` subelement of <distributed-scheme> for more information about the system property.

Large Cluster Configuration

Are there special considerations for large clusters?

- The general recommendation for the `<partition-count>` subelement of `<distributed-scheme>` is to be a prime number close to the square of the number of storage enabled nodes. While is a good suggestion for small to medium sized clusters, for large clusters it can add too much overhead. For clusters exceeding 128 storage enabled JVMs, the partition count should be fixed, at roughly 16,381.
- Coherence clusters which consist of over 400 TCMP nodes need to increase the default maximum packet size Coherence will use. The default of 1468 should be increased relative to the size of the cluster, that is, a 600 node cluster would need the maximum packet size increased by 50%. The maximum packet size is configured as part of the coherence operational configuration file, see "packet-size" on page H-38 for details on changing this setting.
- For large clusters which have hundreds of JVMs it is also recommended that <multicast-listener> be enabled, as it will allow for more efficient cluster wide transmissions. These cluster wide transmissions are rare, but when they do occur multicast can provide noticeable benefits in large clusters.

Death Detection

The Coherence death detection algorithms are based on sustained loss of connectivity between two or more cluster nodes. When a node identifies that it has lost connectivity with any other node it will consult with other cluster nodes to determine what action should be taken.

In attempting to consult with others, the node may find that it cannot communicate with any other nodes, and will assume that it has been disconnected from the cluster. Such a condition could be triggered by physically unplugging a node's network adapter. In such an event the isolated node will restart it's clustered services and attempt to rejoin the cluster.

If connectivity with other cluster nodes remains unavailable, the node may (depending on well known address configuration) form a new isolated cluster, or continue searching for the larger cluster. In either case when connectivity is restored the previously isolated cluster nodes will rejoin the running cluster. As part of rejoining the cluster, the nodes former cluster state is discarded, including any cache data it may have held, as the remainder of the cluster had already taken on ownership of that data (restoring from backups).

Without connectivity it is obviously not possible for a node to identify the state of other nodes. This means that from the point of view of a single node, local network adapter failure and network wide switch failure look identical, and are thus handled in the same way, as described above. The important difference is that in the case of a switch failure all nodes are attempting to re-join the cluster, which is the equivalent of a full cluster restart, and all prior state and data is dropped.

Obviously dropping all data is not desirable, and thus if you want to avoid this as part of a sustained switch failure you must take additional precautions. Options include:

- Extend allowable outage duration: The maximum time a node(s) may be unresponsive before being removed from the cluster is configured by using the `<timeout-milliseconds>` subelement of `<packet-delivery>`, and defaults to one minute for production configurations. Increasing this value will allow the cluster to wait longer for connectivity to return. The downside of increasing this

value it may also take longer to handle the case where just a single node has lost connectivity.

- Persist data to external storage: By using aRead Write Backing Map, the cluster persists data to external storage, and can retrieve it after a cluster restart. So long as write-behind is disabled (the `<write-delay>` subelement of `<read-write-backing-map-scheme>`) no data would be lost in the event of a switch failure. The downside here is that synchronously writing through to external storage increases the latency of cache update operations, and the external storage may become a bottleneck.
- Delay node restart: The cluster death detection action can be reconfigured to delay the node restart until connectivity is restored. By delaying the restart until connectivity is restored an isolated node is allowed to continue running with whatever data it had available at the time of disconnect. When connectivity is restored the nodes will detect each other and form a new cluster. In forming a new cluster all but the most senior node will be required to restart. This results in behavior which is nearly identical to the default behavior because the majority of the nodes will restart, and drop their data. It may be beneficial for cases in which replicated caches are in use as the senior most node's copy of the data will survive the restart. To enable the delayed restart the `tangosol.coherence.departure.threshold` system property must be set to a value that is greater then the size of the cluster.

  Note: When running on Microsoft Windows it is also necessary to ensure the Windows does not disable the network adapter when it is disconnected. To do this add the following Windows registry DWORD, setting it to 1:HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\DisableDHCPMediaSenseNote that despite the name this setting affects static IPs as well.

- Add network level fault tolerance: Adding a redundant layer to the cluster's network infrastructure allows for individual pieces of networking equipment to fail without disrupting connectivity. This is commonly achieved by using at least two network adapters per machine, and having each adapter connected to a separate switch. This is not a feature of Coherence but rather of the underlying operating system or network driver. The only change to Coherence is that it should be configured to bind to the virtual rather then physical network adapter. This form of network redundancy goes by different names depending on the operating system, see Linux bonding, Solaris trunking and Windows teaming for further details.

tangosol-license.xml Deprecated

As of Coherence 3.4, the `tangosol-license.xml` file is no longer used.

B

Types of Caches in Coherence

This appendix provides an overview of the types of caches offered by Coherence.

Replicated Cache

In a replicated cache service, data is fully replicated to every member in the cluster. Offers the fastest read performance. Clustered, fault-tolerant cache with linear performance scalability for reads, but poor scalability for writes (as writes must be processed by every member in the cluster). Because data is replicated to all machines, adding servers does not increase aggregate cache capacity.

Optimistic Cache

Optimistic Cache is a clustered cache implementation similar to the Replicated Cache implementation, but without any concurrency control. This implementation has the highest possible throughput. It also allows to use an alternative underlying store for the cached data (for example, a MRU/MFU-based cache). However, if two cluster members are independently pruning or purging the underlying local stores, it is possible that a cluster member may have a different store content than that held by another cluster member.

Distributed (Partitioned) Cache

A distributed, or partitioned cache is clustered, fault-tolerant, and has linear scalability. Data is partitioned among all the machines of the cluster. For fault-tolerance, partitioned caches can be configured to keep each piece of data on one, two or more unique machines within a cluster.

Near Cache

A near cache is a hybrid cache; it fronts a fault-tolerant, scalable partitioned cache with a local cache. Near cache invalidates front cache entries, using configurable invalidation strategy, and provides excellent performance and synchronization. Near cache backed by a partitioned cache offers zero-millisecond local access for repeat data access, while enabling concurrency and ensuring coherency and fail-over, effectively combining the best attributes of replicated and partitioned caches.

Summary of Cache Types

Numerical Terms:

- **JVMs** = number of JVMs
- **DataSize** = total size of cached data (measured without redundancy)
- **Redundancy** = number of copies of data maintained
- **LocalCache** = size of local cache (for near caches)

Table B–1 Summary of Cache Types and Characteristics

	Replicated Cache	Optimistic Cache	Partitioned Cache	Near Cache backed by partitioned cache	LocalCache not clustered
Topology	Replicated	Replicated	Partitioned Cache	Local Caches + Partitioned Cache	Local Cache
Read Performance	Instant **5**	Instant **5**	Locally cached: instant **5** Remote: network speed **1**	Locally cached: instant **5** Remote: network speed **1**	Instant **5**
Fault Tolerance	Extremely High	Extremely High	Configurable **4** Zero to Extremely High	Configurable 4 Zero to Extremely High	Zero
Write Performance	Fast **2**	Fast **2**	Extremely fast **3**	Extremely fast **3**	Instant **5**
Memory Usage (Per JVM)	DataSize	DataSize	DataSize/JVMs x Redundancy	LocalCache + [DataSize / JVMs]	DataSize
Coherency	fully coherent	fully coherent	fully coherent	fully coherent **6**	n/a
Memory Usage (Total)	JVMs x DataSize	JVMs x DataSize	Redundancy x DataSize	[Redundancy x DataSize] + [JVMs x LocalCache]	n/a
Locking	fully transactional	none	fully transactional	fully transactional	fully transactional
Typical Uses	Metadata	n/a (see Near Cache)	Read-write caches	Read-heavy caches w/ access affinity	Local data

Notes:

1. As a rough estimate, with 100mbit Ethernet, network reads typically require ~20ms for a 100KB object. With gigabit Ethernet, network reads for 1KB objects are typically sub-millisecond.
2. Requires UDP multicast or a few UDP unicast operations, depending on JVM count.
3. Requires a few UDP unicast operations, depending on level of redundancy.
4. Partitioned caches can be configured with as many levels of backup as desired, or zero if desired. Most installations use one backup copy (two copies total)
5. Limited by local CPU/memory performance, with negligible processing required (typically sub-millisecond performance).
6. Listener-based Near caches are coherent; expiry-based near caches are partially coherent for non-transactional reads and coherent for transactional access.

C

Cache Semantics

Use Coherence caches to cache value objects. These objects may represent data from any source, either internal (such as session data, transient data, and so on) or external (such as a database, mainframe, and so on).

Objects placed in the cache must be serializable. The simplest approach to doing this is to implement `java.io.Serializable`. For higher performance, Coherence also supports the `java.io.Externalizable` and (even faster) `com.tangosol.io.ExternalizableLite` interfaces. The primary difference between `Externalizable` and `ExternalizableLite` is the I/O stream used. In most cases, porting from one to the other is simple.

> **Note:** Remember, when serializing an object, Java serialization automatically crawls every visible object (by using object references, including collections like `Map` and `List`). As a result, cached objects **should not** refer to their parent objects directly (holding onto an identifying value like an integer is OK).
>
> Objects that implement their own serialization routines are not affected.

Any objects that implement `com.tangosol.run.xml.XmlBean` will automatically support `ExternalizableLite`. For more details, see the API Javadoc for `XmlBean`.

D

Cache Configuration Elements

This appendix provides a listing of the elements that can be used in a cache configuration. In addition, it describes the deployment descriptor file in which they appear.

Cache Configuration Deployment Descriptor

Use the cache configuration deployment descriptor to specify the various types of caches which can be used within a cluster. For information on configuring cluster communication and services see Appendix H, "Operational Configuration Elements."

Document Location

The name and location of the descriptor is specified in the operational deployment descriptor and defaults to `coherence-cache-config.xml`. The default configuration descriptor (packaged in `coherence.jar`) will be used unless a custom one is found within the application's classpath. It is recommended that all nodes within a cluster use identical cache configuration descriptors.

Document Root

The root element of the configuration descriptor is <cache-config>. This is where you may begin configuring your caches.

Document Format

The Cache Configuration descriptor should begin with the following DOCTYPE declaration:

```
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">
```

> **Note:** When deploying Coherence into environments where the default character set is EBCDIC rather than ASCII, make sure that this descriptor file is in ASCII format and is deployed into its runtime environment in the binary format.

Command Line Override

Oracle Coherence provides a powerful command line override feature which allows any element defined in this descriptor to be overridden from the Java command line if

it has a system-property attribute defined in the descriptor. For more information on this feature, see Appendix L, "Command Line Overrides".

Examples

See Appendix F, "Sample Cache Configurations" for usage examples.

Element Index

The following table lists all non-terminal elements which may be used from within a cache configuration.

Table D–1 Cache Configuration Elements

Element	Used In:
acceptor-config	proxy-scheme
async-store-manager	external-scheme, paged-external-scheme
authorized-hosts	tcp-acceptor
backup-storage	distributed-scheme
bdb-store-manager	external-scheme, paged-external-scheme, async-store-manager
cache-config	*root element*
cache-mapping	caching-scheme-mapping
cache-service-proxy	proxy-config
caching-scheme-mapping	cache-config
caching-schemes	cache-config
class-scheme	caching-schemes, local-scheme, distributed-scheme, replicated-scheme, optimistic-scheme, near-scheme, overflow-scheme, read-write-backing-map-scheme, cachestore-scheme, listener
cachestore-scheme	local-scheme, read-write-backing-map-scheme
custom-store-manager	external-scheme, paged-external-scheme, async-store-manager
disk-scheme	caching-schemes
distributed-scheme	caching-schemes, near-scheme, overflow-scheme
external-scheme	caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme, near-scheme, overflow-scheme, read-write-backing-map-scheme
init-param	init-params
init-params	class-scheme
initiator-config	remote-cache-scheme, remote-invocation-scheme
invocation-scheme	caching-schemes
jms-acceptor	acceptor-config
jms-initiator	initiator-config
key-associator	distributed-scheme
key-partitioning	distributed-scheme
lh-file-manager	external-scheme, paged-external-scheme, async-store-manager

Table D–1 (Cont.) Cache Configuration Elements

Element	Used In:
listener	disk-scheme, local-scheme, external-scheme, paged-external-scheme, distributed-scheme, replicated-scheme, optimistic-scheme, near-scheme, overflow-scheme, read-write-backing-map-scheme
local-scheme	caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme, near-scheme, overflow-scheme, read-write-backing-map-scheme
near-scheme	caching-schemes
nio-file-manager	external-scheme, paged-external-scheme, async-store-manager
nio-memory-manager	external-scheme, paged-external-scheme, async-store-manager
operation-bundling	cachestore-scheme, distributed-scheme, remote-cache-scheme
optimistic-scheme	caching-schemes, near-scheme, overflow-scheme
outgoing-message-handler	acceptor-config, initiator-config
overflow-scheme	caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme, read-write-backing-map-scheme
paged-external-scheme	caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme, near-scheme, overflow-scheme, read-write-backing-map-scheme
proxy-config	proxy-scheme
proxy-scheme	caching-schemes
read-write-backing-map-scheme	caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme
remote-cache-scheme	cachestore-scheme, caching-schemes, near-scheme
remote-invocation-scheme	caching-schemes
replicated-scheme	caching-schemes, near-scheme, overflow-scheme
tcp-acceptor	acceptor-config
tcp-initiator	initiator-config

acceptor-config

Used in: proxy-scheme

Description

The acceptor-config element specifies the configuration information for a protocol-specific connection acceptor. The connection acceptor is used by a proxy service to enable Coherence*Extend clients to connect to the cluster and use the services offered by the cluster without having to join the cluster.

The acceptor-config element must contain exactly one protocol-specific connection acceptor configuration element (either jms-acceptor or tcp-acceptor).

Elements

Table D–2 describes the elements you can define within the acceptor-config element.

Table D–2 acceptor-config Subelements

Element	Required/ Optional	Description
<connection-limit>	Optional	The maximum number of simultaneous connections allowed by this connection acceptor. Valid values are positive integers and zero. A value of zero implies no limit. Default value is zero.
<jms-acceptor>	Optional	Specifies the configuration info for a connection acceptor that enables Coherence*Extend clients to connect to the cluster over JMS.
<outgoing-message-handler>	Optional	Specifies the configuration info used by the connection acceptor to detect dropped client-to-cluster connections.

Table D–2 (Cont.) acceptor-config Subelements

Element	Required/ Optional	Description

`<serializer>`	Optional	Specifies the class configuration info for a `com.tangosol.io.Serializer` implementation used by the connection acceptor to serialize and deserialize user types. For example, the following configures a `ConfigurablePofContext` that uses the `my-pof-types.xml` POF type configuration file to deserialize user types to and from a POF stream:
		`<serializer>`
		`  <class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>`
		`  <init-params>`
		`  <init-param>`
		`  <param-type>string</param-type>`
		`  <param-value>my-pof-types.xml</param-value>`
		`  </init-param>`
		`  </init-params>`
		`</serializer>`

`<tcp-acceptor>`	Optional	Specifies the configuration info for a connection acceptor that enables Coherence*Extend clients to connect to the cluster over TCP/IP.

`<use-filters>`	Optional	Contains the list of <filters> names to be used by this connection acceptor.For example, specifying `use-filter` as follows will activate gzip compression for all network messages, which can help substantially with WAN and low-bandwidth networks.
		`<use-filters>`
		`  <filter-name>gzip</filter-name>`
		`</use-filters>`

address-provider

Used in: tcp-initiator

Description

Contains the configuration info for an address factory that implements the com.tangosol.net.AddressProvider interface.

Elements

Table D–3 describes the subelements you can define within the address-provider element.

Table D–3 address-provider Subelements

Element	Required/ Optional	Description
<class-factory-name>	Optional	Specifies a fully specified name of a Java class that will be used as a factory for object instantiation.
<class-name>	Required	The name of a class that implements the com.tangosol.net.AddressProvider interface.
<init-params>	Optional	Specifies initialization parameters which are accessible by implementations which support the com.tangosol.run.xml.XmlConfigurable interface, or which include a public constructor with a matching signature.
<method-name>	Optional	Specifies the name of a static factory method on the factory class which will perform object instantiation.

async-store-manager

Used in: external-scheme, paged-external-scheme.

Description

The async-store-manager element adds asynchronous write capabilities to other store manager implementations. Supported store managers include:

- custom-store-manager—allows definition of custom implementations of store managers
- bdb-store-manager—uses Berkeley Database JE to implement an on disk cache
- lh-file-manager—uses a Coherence LH on disk database cache
- nio-file-manager—uses NIO to implement memory-mapped file based cache
- nio-memory-manager—uses NIO to implement an off JVM heap, in-memory cache

Implementation

This store manager is implemented by the com.tangosol.io.AsyncBinaryStoreManager class.

Elements

Table D–4 describes the subelements you can define within the async-store-manager element.

Table D–4 async-store-manager Subelements

Element	Required/ Optional	Description
<async-limit>	Optional	Specifies the maximum number of bytes that will be queued to be written asynchronously. Setting the value to zero does not disable the asynchronous writes; instead, it indicates that the implementation default for the maximum number of bytes are necessary.The value of this element must be in the following format: [\d]+[[.][\d]+]?[K|k|M|m]?[B|b]? where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied: ■ K (kilo, 210) ■ M (mega, 220) If the value does not contain a factor, a factor of one is assumed. Valid values are any positive memory sizes and zero. Default value is 4MB.
<bdb-store-manager>	Optional	Configures the external cache to use Berkeley Database JE on disk databases for cache storage.
<class-name>	Optional	Specifies a custom implementation of the async-store-manager. Any custom implementation must extend the com.tangosol.io.AsyncBinaryStoreManager class and declare the exact same set of public constructors.
<custom-store-manager>	Optional	Configures the external cache to use a custom storage manager implementation.

Table D–4 (Cont.) async-store-manager Subelements

Element	Required/ Optional	Description
<init-params>	Optional	Specifies initialization parameters, for use in custom async-store-manager implementations which implement the `com.tangosol.run.xml.XmlConfigurable` interface.
<lh-file-manager>	Optional	Configures the external cache to use a Coherence LH on disk database for cache storage.
<nio-file-manager>	Optional	Configures the external cache to use a memory-mapped file for cache storage.
<nio-memory-manager>	Optional	Configures the external cache to use an off JVM heap, memory region for cache storage.

backup-storage

Used in: distributed-scheme.

Description

The backup-storage element specifies the type and configuration of backup storage for a partitioned cache.

Elements

The following table describes the elements you can define within the backup-storage element.

Table D–5 backup-storage Subelements

Element	Required/ Optional	Description
<class-name>	Optional	Only applicable with the custom type. Specifies a class name for the custom storage implementation. If the class implements com.tangosol.run.xml.XmlConfigurable interface then upon construction, the setConfig method is called passing the entire backup-storage element. Default value is the backup-storage/class-name value specified in the tangosol-coherence.xml descriptor. See "DistributedCache Service Parameters" on page I-3 for more information.
<directory>	Optional	Only applicable with the file-mapped type. Specifies the path name for the directory that the disk persistence manager (com.tangosol.util.nio.MappedBufferManager) will use as "root" to store files in. If not specified or specifies a non-existent directory, a temporary file in the default location is used. Default value is the backup-storage/directory value specified in the tangosol-coherence.xml descriptor. See "DistributedCache Service Parameters" on page I-3 for more information.
<initial-size>	Optional	Only applicable with the off-heap and file-mapped types.Specifies the initial buffer size in bytes.The value of this element must be in the following format: [\d]+[[.][\d]]?[K|k|M|m|G|g]?[B|b]? where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied: ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) If the value does not contain a factor, a factor of mega is assumed. Legal values are positive integers between 1 and Integer.MAX_VALUE - 1023 (that is, 2,147,482,624 bytes). Default value is the backup-storage/initial-size value specified in the tangosol-coherence.xml descriptor. See "DistributedCache Service Parameters" on page I-3 for more information.

Table D–5 (Cont.) backup-storage Subelements

Element	Required/ Optional	Description
<maximum-size>	Optional	Only applicable with the off-heap and file-mapped types. Specifies the initial buffer size in bytes.The value of this element must be in the following format: `[\d]+[[.][\d]]?[K|k|M|m|G|g]?[B|b]?` where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied: ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) If the value does not contain a factor, a factor of mega is assumed. Legal values are positive integers between 1 and `Integer.MAX_VALUE` - 1023 (that is, 2,147,482,624 bytes). Default value is the `backup-storage/maximum-size` value specified in the `tangosol-coherence.xml` descriptor. See "DistributedCache Service Parameters" on page I-3 for more information.
<scheme-name>	Optional	Only applicable with the scheme type. Specifies a scheme name for the `ConfigurableCacheFactory`. Default value is the `backup-storage/scheme-name` value specified in the `tangosol-coherence.xml` descriptor. See "DistributedCache Service Parameters" on page I-3 for more information.
<type>	Required	Specifies the type of the storage used to hold the backup data. Legal values are: ■ `on-heap`—The corresponding implementations class is `java.util.HashMap`. ■ `off-heap`—The corresponding implementations class is `com.tangosol.io.nio.BinaryMap` using the `com.tangosol.io.nio.DirectBufferManager`. ■ file-mapped—The corresponding implementations class is `com.tangosol.io.nio.BinaryMap` using the `com.tangosol.io.nio.MappedBufferManager`. ■ `custom`—The corresponding implementations class is the class specified by the `class-name` element. ■ `scheme`—The corresponding implementations class is specified as a caching-scheme by the `scheme-name` element. Default value is the value specified in the `tangosol-coherence.xml` descriptor. For more information, see the <backup-storage/type> parameter in "DistributedCache Service Parameters" on page I-3.

bdb-store-manager

Used in: external-scheme, paged-external-scheme, async-store-manager.

> **Note:** Berkeley Database JE Java class libraries are required to use a bdb-store-manager, see the Berkeley Database JE product page for additional information.

Description

The BDB store manager is used to define external caches which will use Berkeley Database JE on disk embedded databases for storage. See the examples of Berkeley-based store configurations in "Persistent Cache on Disk" on page F-3 and "In-memory Cache with Disk Based Overflow" on page F-4.

Implementation

This store manager is implemented by the com.tangosol.io.bdb.BerkeleyDBBinaryStoreManager class, and produces BinaryStore objects implemented by the com.tangosol.io.bdb.BerkeleyDBBinaryStore class.

Elements

Table D–6 describes the elements you can define within the bdb-store-manager element.

Table D–6 bdb-store-manager Subelements

Element	Required/ Optional	Description
<class-name>	Optional	Specifies a custom implementation of the Berkeley Database BinaryStoreManager. Any custom implementation must extend the com.tangosol.io.bdb.BerkeleyDBBinaryStoreManager class and declare the exact same set of public constructors.
<directory>	Optional	Specifies the path name to the root directory where the Berkeley Database JE store manager will store files. If not specified or specified with a non-existent directory, a temporary directory in the default location will be used.
<init-params>	Optional	Specifies additional Berkeley DB configuration settings. See Berkeley DB Configuration. Also used to specify initialization parameters, for use in custom implementations which implement the com.tangosol.run.xml.XmlConfigurable interface.
<store-name>	Optional	Specifies the name for a database table that the Berkeley Database JE store manager will use to store data in. Specifying this parameter will cause the bdb-store-manager to use non-temporary (persistent) database instances. This is intended only for local caches that are backed by a cache loader from a non-temporary store, so that the local cache can be pre-populated from the disk on startup. When specified, it is recommended that it use the {cache-name} macro. Normally this parameter should be left unspecified, indicating that temporary storage is to be used. See Appendix E, "Cache Configuration Parameter Macros" for more information on the {cache-name} macro.

bundle-config

Used in: operation-bundling.

Description

The bundle-config element specifies the bundling strategy configuration for one or more bundle-able operations.

Elements

Table D–7 describes the subelements you can define within the bundle-config element.

Table D–7 bundle-config Subelements

Element	Required/ Optional	Description
<auto-adjust>	Optional	Specifies whether the auto adjustment of the preferred-size value (based on the run-time statistics) is allowed. Valid values are true or false. Default value is false.
<delay-millis>	Optional	Specifies the maximum amount of time in milliseconds that individual execution requests are allowed to be deferred for a purpose of "bundling" them together and passing into a corresponding bulk operation. If the preferred-size threshold is reached before the specified delay, the bundle is processed immediately. Valid values are positive numbers. Default value is 1.
<operation-name>	Required	Specifies the operation name for which calls performed concurrently on multiple threads will be "bundled" into a functionally analogous "bulk" operation that takes a collection of arguments instead of a single one. Valid values depend on the bundle configuration context. For the <cachestore-scheme> the valid operations are: ▪ load" ▪ store ▪ erase For the <distributed-scheme> and <remote-cache-scheme> the valid operations are: ▪ get ▪ put ▪ remove In all cases there is a pseudo operation named all, referring to all valid operations. Default value is all.
<preferred-size>	Optional	Specifies the bundle size threshold. When a bundle size reaches this value, the corresponding "bulk" operation will be invoked immediately. This value is measured in context-specific units. Valid values are zero (disabled bundling) or positive values. Default value is zero.
<thread-threshold>	Optional	Specifies the minimum number of threads that must be concurrently executing individual (non-bundled) requests for the bundler to switch from a pass-through to a bundling mode. Valid values are positive numbers. Default value is 4.

cache-config

Root Element

Description

The `cache-config` element is the root element of the cache configuration descriptor, `coherence-cache-config.xml`. For more information on this document, see "Cache Configuration Deployment Descriptor" on page D-1.

At a high level, a cache configuration consists of cache schemes and cache scheme mappings. Cache schemes describe a type of cache, for instance a database backed, distributed cache. Cache mappings define what scheme to use for a given cache name.

Elements

Table D–8 describes the subelements you can define within the `cache-config` element.

Table D–8 cache-config Subelements

Element	Required/ Optional	Description
<caching-scheme-mapping>	Required	Specifies the caching-scheme that will be used for caches, based on the cache's name.
<caching-schemes>	Required	Defines the available caching-schemes for use in the cluster.

cache-mapping

Used in: caching-scheme-mapping

Description

Each `cache-mapping` element specifies the caching-schemes which are to be used for a given cache name or pattern.

Elements

Table D–9 describes the subelements you can define within the `cache-mapping` element.

Table D–9 cache-mapping Subelements

Element	Required/ Optional	Description
`<cache-name>`	Required	Specifies a cache name or name pattern. The name is unique within a cache factory.The following cache name patterns are supported: ▪ exact match, for example, `MyCache` ▪ prefix match, for example, `My*` that matches to any cache name starting with `My` ▪ any match "`*`", that matches to any cache name The patterns get matched in the order of specificity (more specific definition is selected whenever possible). For example, if both `MyCache` and `My*` mappings are specified, the scheme from the `MyCache` mapping will be used to configure a cache named `MyCache`.
`<scheme-name>`	Required	Contains the caching scheme name. The name is unique within a configuration file. Caching schemes are configured in the caching-schemes element.

`<init-params>`	Optional	Allows specifying replaceable cache scheme parameters. During cache scheme parsing, any occurrence of any replaceable parameter in format `param-name` is replaced with the corresponding parameter value. Consider the following cache mapping example:
		`<cache-mapping>`
		`  <cache-name>My*</cache-name>`
		`  <scheme-name>my-scheme</scheme-name>`
		`  <init-params>`
		`  <init-param>`
		`  <param-name>cache-loader</param-name>`
		`  <param-value>com.acme.MyCacheLoader</param-value>`
		`  </init-param>`
		`  <init-param>`
		`  <param-name>size-limit</param-name>`
		`  <param-value>1000</param-value>`
		`  </init-param>`
		`  </init-params>`
		`</cache-mapping>`
		For any cache name match `My*`, any occurrence of the literal `cache-loader` in any part of the corresponding `cache-scheme` element will be replaced with the string `com.acme.MyCacheLoader` and any occurrence of the literal size-limit will be replaced with the value of `1000`. Since Coherence 3.0

cache-service-proxy

Used in: proxy-config

Description

The `cache-service-proxy` element contains the configuration info for a cache service proxy managed by a proxy service.

Elements

Table D–10 describes the elements you can define within the `cache-service-proxy` element.

Table D–10 cache-service-proxy Subelements

Element	Required/ Optional	Description
`<enabled>`	Optional	Specifies whether the cache service proxy is enabled. If disabled, clients will not be able to access any proxied caches. Legal values are `true` or `false`. Default value is `true`.
`<lock-enabled>`	Optional	Specifies whether lock requests from remote clients are permitted on a proxied cache. Legal values are `true` or `false`. Default value is `false`.
`<read-only>`	Optional	Specifies whether requests from remote clients that update a cache are prohibited on a proxied cache. Legal values are `true` or `false`. Default value is `false`.

cachestore-scheme

Used in: local-scheme, read-write-backing-map-scheme, versioned-backing-map-scheme.

Description

Cache store schemes define a mechanism for connecting a cache to a back-end data store. The cache store scheme may use any class implementing either the `com.tangosol.net.cache.CacheStore` or `com.tangosol.net.cache.CacheLoader` interfaces, where the former offers read-write capabilities, where the latter is read-only. Custom implementations of these interfaces may be produced to connect Coherence to various data stores. See "Cache of a Database" on page F-4 for an example of using a `cachestore-scheme`.

Elements

Table D–11 describes the elements you can define within the `cachestore-scheme` element.

Table D–11 cachestore-scheme Subelements

Element	Required/ Optional	Description
<scheme-name>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<scheme-ref>	Optional	Specifies the name of another scheme to inherit from. See "Scheme Inheritance" on page D-19.
<class-scheme>	Optional	Specifies the implementation of the cache store. The specified class must implement one of the following two interfaces. ▪ `com.tangosol.net.cache.CacheStore`—for read-write support ▪ `com.tangosol.net.cache.CacheLoader`—for read-only support
<remote-cache-scheme>	Optional	Configures the cachestore-scheme to use Coherence*Extend as its cache store implementation.
<operation-bundling>	Optional	Specifies the configuration info for a bundling strategy.

caching-scheme-mapping

Used in: cache-config

Description

Defines mappings between cache names, or name patterns, and caching-schemes. For instance you may define that caches whose names start with `accounts-` will use a distributed (distributed-scheme) caching scheme, while caches starting with the name `rates-` will use a `replicated-scheme` caching scheme.

Elements

Table D–12 describes the subelement you can define within the `caching-scheme-mapping` element.

Table D–12 caching-scheme-mapping Subelement

Element	Required/ Optional	Description
<cache-mapping>	Optional	Contains a single binding between a cache name and the caching scheme this cache will use.

caching-schemes

Used in: cache-config

Description

The `caching-schemes` element defines a series of cache scheme elements. Each cache scheme defines a type of cache, for instance a database backed partitioned cache, or a local cache with an LRU eviction policy. Scheme types are bound to actual caches using caching-scheme-mappings.

Scheme Types and Names

Each of the cache scheme element types is used to describe a different type of cache, for instance distributed, versus replicated. Multiple instances of the same type may be defined so long as each has a unique scheme-name.

Example D–1 illustrates the configuration of two different distributed schemes

Example D–1 Configuring Two Different Distributed Schemes

```
<distributed-scheme>
  <scheme-name>DistributedInMemoryCache</scheme-name>
  <service-name>DistributedCache</service-name>
  <backing-map-scheme>
    <local-scheme/>
  </backing-map-scheme>
</distributed-scheme>

<distributed-scheme>
  <scheme-name>DistributedOnDiskCache</scheme-name>
  <service-name>DistributedCache</service-name>
  <backing-map-scheme>
    <external-scheme>
      <nio-file-manager>
        <initial-size>8MB</initial-size>
        <maximum-size>512MB</maximum-size>
        <directory></directory>
      </nio-file-manager>
    </external-scheme>
  </backing-map-scheme>
</distributed-scheme>
```

Nested Schemes

Some caching scheme types contain nested scheme definitions. For instance in the above example the distributed schemes include a nested scheme definition describing their backing map.

Scheme Inheritance

Caching schemes can be defined by specifying all the elements required for a given scheme type, or by inheriting from another named scheme of the same type, and selectively overriding specific values. Scheme inheritance is accomplished by including a `<scheme-ref>` element in the inheriting scheme containing the `scheme-name` of the scheme to inherit from.

For example, the two configurations in Example D–2 will produce equivalent `DistributedInMemoryCache` scheme definitions:

Example D–2 Configuring Equivalent Scheme Definitions

```
<distributed-scheme>
  <scheme-name>DistributedInMemoryCache</scheme-name>
  <service-name>DistributedCache</service-name>
  <backing-map-scheme>
    <local-scheme>
      <eviction-policy>LRU</eviction-policy>
      <high-units>1000</high-units>
      <expiry-delay>1h</expiry-delay>
    </local-scheme>
  </backing-map-scheme>
</distributed-scheme>

<distributed-scheme>
  <scheme-name>DistributedInMemoryCache</scheme-name>
  <service-name>DistributedCache</service-name>
  <backing-map-scheme>
    <local-scheme>
      <scheme-ref>LocalSizeLimited</scheme-ref>
    </local-scheme>
  </backing-map-scheme>
</distributed-scheme>

<local-scheme>
  <scheme-name>LocalSizeLimited</scheme-name>
  <eviction-policy>LRU</eviction-policy>
  <high-units>1000</high-units>
  <expiry-delay>1h</expiry-delay>
</local-scheme>
```

Note that while the first is somewhat more compact, the second offers the ability to easily reuse the `LocalSizeLimited` scheme within multiple schemes. Example D–3 demonstrates multiple schemes reusing the same `LocalSizeLimited` base definition, but the second imposes a different `expiry-delay`.

Example D–3 Multiple Schemes Reusing the Same Base Definition

```
<distributed-scheme>
  <scheme-name>DistributedInMemoryCache</scheme-name>
  <service-name>DistributedCache</service-name>
  <backing-map-scheme>
    <local-scheme>
      <scheme-ref>LocalSizeLimited</scheme-ref>
    </local-scheme>
  </backing-map-scheme>
</distributed-scheme>

<replicated-scheme>
  <scheme-name>ReplicatedInMemoryCache</scheme-name>
  <service-name>ReplicatedCache</service-name>
  <backing-map-scheme>
    <local-scheme>
      <scheme-ref>LocalSizeLimited</scheme-ref>
      <expiry-delay>10m</expiry-delay>
    </local-scheme>
  </backing-map-scheme>
```

```
</replicated-scheme>

<local-scheme>
  <scheme-name>LocalSizeLimited</scheme-name>
  <eviction-policy>LRU</eviction-policy>
  <high-units>1000</high-units>
  <expiry-delay>1h</expiry-delay>
</local-scheme>
```

Elements

Table D–13 describes the different types of schemes you can define within the `caching-schemes` element.

Table D–13 caching-schemes Subelements

Element	Required/ Optional	Description
<local-scheme>	Optional	Defines a cache scheme which provides on-heap cache storage.
<external-scheme>	Optional	Defines a cache scheme which provides off-heap cache storage, for instance on disk.
<paged-external-scheme>	Optional	Defines a cache scheme which provides off-heap cache storage, that is size-limited by using time based paging.
<distributed-scheme>	Optional	Defines a cache scheme where storage of cache entries is partitioned across the cluster nodes.
<replicated-scheme>	Optional	Defines a cache scheme where each cache entry is stored on all cluster nodes.
<optimistic-scheme>	Optional	Defines a replicated cache scheme which uses optimistic rather then pessimistic locking.
<near-scheme>	Optional	Defines a two tier cache scheme which consists of a fast local front-tier cache of a much larger back-tier cache.
<versioned-near-scheme>	Optional	Defines a near-scheme which uses object versioning to ensure coherence between the front and back tiers.
<overflow-scheme>	Optional	Defines a two tier cache scheme where entries evicted from a size-limited front-tier overflow and are stored in a much larger back-tier cache.
<invocation-scheme>	Optional	Defines an invocation service which can be used for performing custom operations in parallel across cluster nodes.
<read-write-backing-map-scheme>	Optional	Defines a backing map scheme which provides a cache of a persistent store.
<versioned-backing-map-scheme>	Optional	Defines a backing map scheme which uses object versioning to determine what updates need to be written to the persistent store.

Table D–13 (Cont.) caching-schemes Subelements

Element	Required/ Optional	Description
<remote-cache-scheme>	Optional	Defines a cache scheme that enables caches to be accessed from outside a Coherence cluster by using Coherence*Extend.
<class-scheme>	Optional	Defines a cache scheme using a custom cache implementation. Any custom implementation must implement the `java.util.Map` interface, and include a zero-parameter public constructor. Additionally if the contents of the Map can be modified by anything other than the `CacheService` itself (for example, if the `Map` automatically expires its entries periodically or size-limits its contents), then the returned object must implement the `com.tangosol.util.ObservableMap` interface.
<disk-scheme>	Optional	**Note:** As of Coherence 3.0, the disk-scheme configuration element has been deprecated and replaced by the external-scheme and paged-external-scheme configuration elements.

class-scheme

Used in: caching-schemes, local-scheme, distributed-scheme, replicated-scheme, optimistic-scheme, near-scheme, versioned-near-scheme, overflow-scheme, read-write-backing-map-scheme, versioned-backing-map-scheme, cachestore-scheme, listener

Description

Class schemes provide a mechanism for instantiating an arbitrary Java object for use by other schemes. The scheme which contains this element will dictate what class or interface(s) must be extended. See "Cache of a Database" on page F-4 for an example of using a class-scheme.

The class-scheme may be configured to either instantiate objects directly by using their class-name, or indirectly by using a class-factory-name and method-name. The class-scheme must be configured with either a class-name or class-factory-name and method-name.

Elements

Table D–14 describes the elements you can define within the class-scheme element.

Table D–14 class-scheme Subelements

Element	Required/ Optional	Description
<scheme-name>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<scheme-ref>	Optional	Specifies the name of another scheme to inherit from. See "Scheme Inheritance" on page D-19 for more information.
<class-name>	Optional	Contains a fully specified Java class name to instantiate. This class must extend an appropriate implementation class as dictated by the containing scheme and must declare the exact same set of public constructors as the superclass.
<class-factory-name>	Optional	Specifies a fully specified name of a Java class that will be used as a factory for object instantiation.
<method-name>	Optional	Specifies the name of a static factory method on the factory class which will perform object instantiation.
<init-params>	Optional	Specifies initialization parameters which are accessible by implementations which support the com.tangosol.run.xml.XmlConfigurable interface, or which include a public constructor with a matching signature.

custom-store-manager

Used in: external-scheme, paged-external-scheme, async-store-manager.

Description

Used to create and configure custom implementations of a store manager for use in external caches.

Elements

Table D–15 describes the elements you can define within the custom-store-manager element.

Table D–15 custom-store-manager Subelements

Element	Required/ Optional	Description
<class-name>	Required	Specifies the implementation of the store manager. The specified class must implement the com.tangosol.io.BinaryStoreManager interface.
<init-params>	Optional	Specifies initialization parameters, for use in custom store manager implementations which implement the com.tangosol.run.xml.XmlConfigurable interface.

disk-scheme

Note: As of Coherence 3.0, the disk-scheme configuration element has been deprecated and replaced with by the <external-scheme> and <paged-external-scheme> configuration elements.

distributed-scheme

Used in: caching-schemes, near-scheme, versioned-near-scheme, overflow-scheme, versioned-backing-map-scheme

Description

The distributed-scheme defines caches where the storage for entries is partitioned across cluster nodes. See "Partitioned Cache Service" for a more detailed description of partitioned caches. See "Partitioned Cache" on page F-6 examples of various distributed-scheme configurations.

Clustered Concurrency Control

Partitioned caches support cluster wide key-based locking so that data can be modified in a cluster without encountering the classic missing update problem. Note that any operation made without holding an explicit lock is still atomic but there is no guarantee that the value stored in the cache does not change between atomic operations.

Cache Clients

The partitioned cache service supports the concept of cluster nodes which do not contribute to the overall storage of the cluster. Nodes which are not storage enabled (see <local-storage> subelement) are considered "cache clients".

Cache Partitions

The cache entries are evenly segmented into several logical partitions (see <partition-count> subelement), and each storage enabled (see <local-storage> subelement) cluster node running the specified partitioned service (see <service-name> subelement) will be responsible for maintain a fair-share of these partitions.

Key Association

By default the specific set of entries assigned to each partition is transparent to the application. In some cases it may be advantageous to keep certain related entries within the same cluster node. A key-associator (see <key-associator> subelement) may be used to indicate related entries, the partitioned cache service will ensure that associated entries reside on the same partition, and thus on the same cluster node. Alternatively, key association may be specified from within the application code by using keys which implement the com.tangosol.net.cache.KeyAssociation interface.

Cache Storage (Backing Map)

Storage for the cache is specified by using the backing-map-scheme (see <backing-map-scheme> subelement). For instance a partitioned cache which uses a local-scheme for its backing map will result in cache entries being stored in-memory on the storage enabled cluster nodes.

Failover

For the purposes of failover a configurable number of backups (see <backup-count> subelement) of the cache may be maintained in backup-storage (see <backup-storage> subelement) across the cluster nodes. Each backup is also divided into

partitions, and when possible a backup partition will not reside on the same physical machine as the primary partition. If a cluster node abruptly leaves the cluster, responsibility for its partitions will automatically be reassigned to the existing backups, and new backups of those partitions will be created (on remote nodes) to maintain the configured backup count.

Partition Redistribution

When a node joins or leaves the cluster, a background redistribution of partitions occurs to ensure that all cluster nodes manage a fair-share of the total number of partitions. The amount of bandwidth consumed by the background transfer of partitions is governed by the transfer-threshold (see `<transfer-threshold>` subelement).

Elements

Table D–16 describes the elements you can define within the `distributed-scheme` element.

Table D–16 distributed-scheme Subelements

Element	Required/ Optional	Description
`<scheme-name>`	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
`<scheme-ref>`	Optional	Specifies the name of another scheme to inherit from. See "Scheme Inheritance" on page D-19 for more information.
`<service-name>`	Optional	Specifies the name of the service which will manage caches created from this scheme. Services are configured in the `<services>` element in the `tangosol-coherence.xml` descriptor. See Appendix H, "Operational Configuration Elements" for more information.

`<serializer>`	Optional	Specifies the class configuration info for a `com.tangosol.io.Serializer` implementation used by the Partitioned service to serialize and deserialize user types.
		
		For example, the following configures a `ConfigurablePofContext` that uses the default `coherence-pof-config.xml` configuration file to write objects to and read from a stream:
		
		`<serializer>`
		`  <class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>`
		`</serializer>`

`<listener>`	Optional	Specifies an implementation of a `com.tangosol.MapListener` which will be notified of events occurring on the cache.

Table D–16 (Cont.) distributed-scheme Subelements

Element	Required/ Optional	Description
<backing-map-scheme>	Optional	Specifies what type of cache will be used within the cache server to store the entries. Legal values are: ■ class-scheme ■ external-scheme ■ local-scheme ■ paged-external-scheme ■ overflow-scheme ■ read-write-backing-map-scheme ■ versioned-backing-map-scheme When using an overflow-based backing map it is important that the corresponding backup-storage be configured for overflow (potentially using the same scheme as the backing-map). See "Partitioned Cache with Overflow" on page F-6 for an example configuration.
<partition-count>	Optional	Specifies the number of partitions that a partitioned cache will be "chopped up" into. Each node running the partitioned cache service that has the local-storage (<local-storage> subelement) option set to true will manage a "fair" (balanced) number of partitions. The number of partitions should be larger than the square of the number of cluster members to achieve a good balance, and it is suggested that the number be prime. Good defaults include 257 and 1021 and prime numbers in-between, depending on the expected cluster size. For large clusters it is recommended that the partition count not exceeded 16,381, regardless of the number of storage enabled members. A list of first 1,000 primes can be found at http://www.utm.edu/research/primes/lists/small/1000.txt. Legal values are prime numbers. Default value is the partition-count value specified in the tangosol-coherence.xml descriptor. See the partition-count parameter "DistributedCache Service Parameters" on page I-3 for more information.
<key-associator>	Optional	Specifies a class that will be responsible for providing associations between keys and allowing associated keys to reside on the same partition. This implementation must have a zero-parameter public constructor.
<key-partitioning>	Optional	Specifies a class that implements the com.tangosol.net.partition.KeyPartitioningStrategy interface, which will be responsible for assigning keys to partitions. This implementation must have a zero-parameter public constructor. If unspecified, the default key partitioning algorithm will be used, which ensures that keys are evenly segmented across partitions.
<partition-listener>	Optional	Specifies a class that implements the com.tangosol.net.partition.PartitionListener interface.

Table D–16 (Cont.) distributed-scheme Subelements

Element	Required/ Optional	Description
<backup-count>	Optional	Specifies the number of members of the partitioned cache service that hold the backup data for each unit of storage in the cache. Value of 0 means that in the case of abnormal termination, some portion of the data in the cache will be lost. Value of N means that if up to N cluster nodes terminate immediately, the cache data will be preserved. To maintain the partitioned cache of size M, the total memory usage in the cluster does not depend on the number of cluster nodes and will be in the order of M*(N+1). Recommended values are 0, 1 or 2. Default value is the backup-count value specified in the tangosol-coherence.xml descriptor. See the backup-count parameter in value specified in the tangosol-coherence.xml descriptor. See "DistributedCache Service Parameters" on page I-3 for more information.
<backup-count-after-writebehind>	Optional	Specifies the number of members of the partitioned cache service that will hold the backup data for each unit of storage in the cache that does *not* require write-behind, that is, data that is not vulnerable to being lost even if the entire cluster were shut down. Specifically, if a unit of storage is marked as requiring write-behind, then it will be backed up on the number of members specified by the <backup-count> subelement, and if the unit of storage is not marked as requiring write-behind, then it will be backed up by the number of members specified by the <backup-count-after-writebehind> element. This value should be set to 0 or this setting should not be specified at all. The rationale is that since this data is being backed up to another data store, no in-memory backup is required, other than the data temporarily queued on the write-behind queue to be written. (Note that the setting also applies to write-through data, or any data that can be re-loaded from another data store by a CacheLoader or CacheStore.) The value of 0 means that when write-behind has occurred, the backup copies of that data will be discarded. However, until write-behind occurs, the data will be backed up in accordance with the <backup-count> setting. Recommended value is 0 or this element should be omitted altogether.
<backup-storage>	Optional	Specifies the type and configuration for the partitioned cache backup storage.
<thread-count>	Optional	Specifies the number of daemon threads used by the partitioned cache service. If zero, all relevant tasks are performed on the service thread. Legal values are positive integers or zero. Default value is the thread-count value specified in the tangosol-coherence.xml descriptor. See the lthread-count parameter in "DistributedCache Service Parameters" on page I-3 for more information.

Table D–16 (Cont.) distributed-scheme Subelements

Element	Required/ Optional	Description
<lease-granularity>	Optional	Specifies the lease ownership granularity. Available since release 2.3.Legal values are: ■ thread ■ member A value of thread means that locks are held by a thread that obtained them and can only be released by that thread. A value of member means that locks are held by a cluster node and any thread running on the cluster node that obtained the lock can release it. Default value is the lease-granularity value specified in the tangosol-coherence.xml descriptor. See the lease-granularity parameter in "DistributedCache Service Parameters" on page I-3 for more information.
<transfer-threshold>	Optional	Specifies the threshold for the primary buckets distribution in kilo-bytes. When a new node joins the partitioned cache service or when a member of the service leaves, the remaining nodes perform a task of bucket ownership re-distribution. During this process, the existing data gets re-balanced along with the ownership information. This parameter indicates a preferred message size for data transfer communications. Setting this value lower will make the distribution process take longer, but will reduce network bandwidth utilization during this activity. Legal values are integers greater then zero. Default value is the transfer-threshold value specified in the tangosol-coherence.xml descriptor. See the transfer-threshold parameter in "DistributedCache Service Parameters" on page I-3 for more information.
<local-storage>	Optional	Specifies whether a cluster node will contribute storage to the cluster, that is, maintain partitions. When disabled the node is considered a cache client. Normally this value should be left unspecified within the configuration file, and instead set on a per-process basis using the tangosol.coherence.distributed.localstorage system property. This allows cache clients and servers to use the same configuration descriptor. Legal values are true or false. Default value is the local-storage value specified in the tangosol-coherence.xml descriptor. See the local-storage parameter in "DistributedCache Service Parameters" on page I-3 for more information.
<autostart>	Optional	The autostart element is intended to be used by cache servers (that is, com.tangosol.net.DefaultCacheServer). It specifies whether the cache services associated with this cache scheme should be automatically started at a cluster node. Legal values are true or false. Default value is false.
<task-hung-threshold>	Optional	Specifies the amount of time in milliseconds that a task can execute before it is considered "hung". Note: a posted task that has not yet started is never considered as hung. This attribute is applied only if the Thread pool is used (the thread-count value is positive). Legal values are positive integers or zero (indicating no default timeout). Default value is the task-hung-threshold value specified in the tangosol-coherence.xml descriptor. See the task-hung-threshold parameter in "DistributedCache Service Parameters" on page I-3 for more information.

Table D–16 (Cont.) distributed-scheme Subelements

Element	Required/ Optional	Description
<task-timeout>	Optional	Specifies the timeout value in milliseconds for requests executing on the service worker threads. This attribute is applied only if the thread pool is used (the thread-count value is positive). Legal values are positive integers or zero (indicating no default timeout). Default value is the value specified in the tangosol-coherence.xml descriptor. See the task-timeout parameter in "DistributedCache Service Parameters" on page I-3.
<request-timeout>	Optional	Specifies the maximum amount of time a client will wait for a response before abandoning the original request. The request time is measured on the client side as the time elapsed from the moment a request is sent for execution to the corresponding server node(s) and includes the following: ■ the time it takes to deliver the request to an executing node (server) ■ the interval between the time the task is received and placed into a service queue until the execution starts ■ the task execution time ■ the time it takes to deliver a result back to the client Legal values are positive integers or zero (indicating no default timeout). Default value is the value specified in the tangosol-coherence.xml descriptor. See the request-timeout parameter in "DistributedCache Service Parameters" on page I-3 for more information.
<operation-bundling>	Optional	Specifies the configuration info for a bundling strategy.

external-scheme

Used in: caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme, near-scheme, versioned-near-scheme, overflow-scheme, read-write-backing-map-scheme, versioned-backing-map-scheme

Description

External schemes define caches which are not JVM heap based, allowing for greater storage capacity. See "Local Caches (accessible from a single JVM)" on page F-2 for examples of various external cache configurations.

Implementation

This scheme is implemented by:

- `com.tangosol.net.cache.SerializationMap`—for unlimited size caches
- `com.tangosol.net.cache.SerializationCache`—for size limited caches

The implementation type is chosen based on the following rule:

- if the `<high-units>` subelement is specified and not zero then `SerializationCache` is used;
- otherwise `SerializationMap` is used.

Pluggable Storage Manager

External schemes use a pluggable store manager to store and retrieve binary key value pairs. Supported store managers include:

- a wrapper providing asynchronous write capabilities for of other store manager implementations
- allows definition of custom implementations of store managers
- uses Berkeley Database JE to implement an on disk cache
- uses a Coherence LH on disk database cache
- uses NIO to implement memory-mapped file based cache
- uses NIO to implement an off JVM heap, in-memory cache

Size Limited Cache

The cache may be configured as size-limited, which means that when it reaches its maximum allowable size (that is, the `<high-units>` subelement) it prunes itself.

Note: Eviction against disk-based caches can be expensive, consider using a paged-external-scheme for such cases.

Entry Expiration

External schemes support automatic expiration of entries based on the age of the value, as configured by the `<expiry-delay>` subelement.

Persistence (long-term storage)

External caches are generally used for temporary storage of large data sets, for example as the back-tier of an `overflow-scheme`. Certain implementations do however support persistence for non-clustered caches, see the `<store-name>` subelement of `bdb-store-manager` and the `<manager-filename>` subelement of `lh-file-manager` for details. Clustered persistence should be configured by using a `read-write-backing-map-scheme` on a `distributed-scheme`.

Elements

Table D–17 describes the elements you can define within the `external-scheme` element.

Table D–17 external-scheme Subelements

Element	Required/ Optional	Description
`<scheme-name>`	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
`<scheme-ref>`	Optional	Specifies the name of another scheme to inherit from. See "Scheme Inheritance" on page D-19 for more information
`<class-name>`	Optional	Specifies a custom implementation of the external cache. Any custom implementation must extend one of the following classes: ▪ `com.tangosol.net.cache.SerializationCache`—for size limited caches ▪ `com.tangosol.net.cache.SerializationMap`—for unlimited size caches ▪ `com.tangosol.net.cache.SimpleSerializationMap`—for unlimited size caches and declare the exact same set of public constructors as the superclass.
`<init-params>`	Optional	Specifies initialization parameters, for use in custom external cache implementations which implement the `com.tangosol.run.xml.XmlConfigurable` interface.
`<listener>`	Optional	Specifies an implementation of a `com.tangosol.util.MapListener` which will be notified of events occurring on the cache.
`<high-units>`	Optional	Used to limit the size of the cache. Contains the maximum number of units that can be placed in the cache before pruning occurs. An entry is the unit of measurement. When this limit is exceeded, the cache will begin the pruning process, evicting the least recently used entries until the number of units is brought below this limit. The scheme's `class-name` element may be used to provide custom extensions to `SerializationCache`, which implement alternative eviction policies. Legal values are positive integers or zero. Zero implies no limit. Default value is zero.

Table D–17 (Cont.) external-scheme Subelements

Element	Required/ Optional	Description
<expiry-delay>	Optional	Specifies the amount of time from last update that entries will be kept by the cache before being expired. Entries that are expired will not be accessible and will be evicted.The value of this element must be in the following format: `[\d]+[[.][\d]+]?[MS|ms|S|s|M|m|H|h|D|d]?` where the first non-digits (from left to right) indicate the unit of time duration: ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) If the value does not contain a unit, a unit of seconds is assumed. A value of zero implies no expiry. Default value is zero.
<async-store-manager>	Optional	Configures the external cache to use an asynchronous storage manager wrapper for any other storage manager. See "Pluggable Storage Manager" on page D-32
<custom-store-manager>	Optional	Configures the external cache to use a custom storage manager implementation.
<bdb-store-manager>	Optional	Configures the external cache to use Berkeley Database JE on disk databases for cache storage.
<lh-file-manager>	Optional	Configures the external cache to use a Coherence LH on disk database for cache storage.
<nio-file-manager>	Optional	Configures the external cache to use a memory-mapped file for cache storage.
<nio-memory-manager>	Optional	Configures the external cache to use an off JVM heap, memory region for cache storage.

initiator-config

Used in: remote-cache-scheme, remote-invocation-scheme.

Description

The `initiator-config` element specifies the configuration info for a protocol-specific connection initiator. A connection initiator allows a Coherence*Extend client to connect to a cluster (by using a connection acceptor) and use the clustered services offered by the cluster without having to first join the cluster.

The `initiator-config` element must contain exactly one protocol-specific connection initiator configuration element (either `jms-initiator` or `tcp-initiator`).

Elements

Table D–18 describes the elements you can define within the `initiator-config` element.

Table D–18 initiator-config Subelements

Element	Required/ Optional	Description
`<jms-initiator>`	Optional	Specifies the configuration info for a connection initiator that connects to the cluster over JMS.
`<outgoing-message-handler>`	Optional	Specifies the configuration info used by the connection initiator to detect dropped client-to-cluster connections.

`<serializer>`	Optional	Specifies the class configuration info for a `Serializer` implementation used by the connection initiator to serialize and deserialize user types. For example, the following configures a `ConfigurablePofContext` that uses the `my-pof-types.xml` POF type configuration file to deserialize user types to and from a POF stream:
		`<serializer>`
		`  <class-name>com.tangosol.io.pof.`
		`ConfigurablePofContext</class-name>`
		`  <init-params>`
		`  <init-param>`
		`  <param-type>string</param-type>`
		`  <param-value>my-pof-types.xml</param-value>`
		`  </init-param>`
		`  </init-params>`
		`</serializer>`

`<tcp-initiator>`	Optional	Specifies the configuration info for a connection initiator that connects to the cluster over TCP/IP.

`<use-filters>`	Optional	Contains the list of `<filters>` names to be used by this connection initiator. In the following example, specifying `use-filter` will activate gzip compression for all network messages, which can help substantially with WAN and low-bandwidth networks.
		`<use-filters>`
		`  <filter-name>gzip</filter-name>`
		`</use-filters>`

init-param

Used in: init-params.

Defines an individual initialization parameter.

Elements

Table D–19 describes the elements you can define within the `init-param` element.

Table D–19 init-param Subelements

Element	Required/ Optional	Description

`<param-name>`	Optional	Contains the name of the initialization parameter. For example:
		`<class-name>com.mycompany.cache.CustomCacheLoader</class-name>`
		`<init-params>`
		`  <init-param>`
		`  <param-name>sTableName</param-name>`
		`  <param-value>EmployeeTable</param-value>`
		`  </init-param>`
		`  <init-param>`
		`  <param-name>iMaxSize</param-name>`
		`  <param-value>2000</param-value>`
		`  </init-param>`
		`</init-params>`

`<param-type>`	Optional	Contains the Java type of the initialization parameter.The following standard types are supported:
		▪ `java.lang.String` (a.k.a. `string`)
		▪ `java.lang.Boolean` (a.k.a. `boolean`)
		▪ `java.lang.Integer` (a.k.a. `int`)
		▪ `java.lang.Long` (a.k.a. `long`)
		▪ `java.lang.Double` (a.k.a. `double`)
		▪ `java.math.BigDecimal`
		▪ `java.io.File`
		▪ `java.sql.Date`
		▪ `java.sql.Time`
		▪ `java.sql.Timestamp`
		For example:
		`<class-name>com.mycompany.cache.CustomCacheLoader</class-name>`
		`<init-params>`
		`  <init-param>`
		`  <param-type>java.lang.String</param-type>`
		`  <param-value>EmployeeTable</param-value>`
		`  </init-param>`
		`  <init-param>`
		`  <param-type>int</param-type>`
		`  <param-value>2000</param-value>`
		`  </init-param>`
		`</init-params>`
		Please refer to the list of available Appendix E, "Cache Configuration Parameter Macros".

`<param-value>`	Optional	Contains the value of the initialization parameter. The value is in the format specific to the Java type of the parameter. See Appendix E, "Cache Configuration Parameter Macros" for the list of available macros.

init-params

Used in: class-scheme, cache-mapping.

Description

Defines a series of initialization parameters as name-value pairs. See "Cache of a Database" on page F-4 for an example of using `init-params`.

Elements

Table D–20 describes the elements you can define within the `init-params` element.

Table D–20 init-params Subelements

Element	Required/ Optional	Description
<init-param>	Optional	Defines an individual initialization parameter.

invocation-scheme

Used in: caching-schemes.

Description

Defines an Invocation Service. The invocation service may be used to perform custom operations in parallel on any number of cluster nodes. See the `com.tangosol.net.InvocationService` API for additional details.

Elements

The following table describes the elements you can define within the invocation-scheme element.

Table D–21 invocation-scheme Subelements

Element	Required/ Optional	Description
`<scheme-name>`	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
`<scheme-ref>`	Optional	Specifies the name of another scheme to inherit from from. See "Scheme Inheritance" on page D-19 for more information.
`<service-name>`	Optional	Specifies the name of the service which will manage invocations from this scheme.

`<serializer>`	Optional	Specifies the class configuration info for a `com.tangosol.io.Serializer` implementation used by the `Invocation` service to serialize and deserialize user types. For example, the following configures a `ConfigurablePofContext` that uses the default `coherence-pof-types.xml` configuration file to write objects to and read from a stream:
		`<serializer>`
		`  <class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>`
		`</serializer>`
		
		Services are configured from within the `<services>` element in the `tangosol-coherence.xml` descriptor. See Appendix H, "Operational Configuration Elements" for more information.

`<thread-count>`	Optional	Specifies the number of daemon threads used by the invocation service. If zero, all relevant tasks are performed on the service thread. Legal values are positive integers or zero. Default value is the `thread-count` value specified in the `tangosol-coherence.xml` descriptor. See the `thread-count` parameter in "InvocationService Parameters" on page I-8.
`<autostart>`	Optional	The `autostart` element is intended to be used by cache servers (that is, `com.tangosol.net.DefaultCacheServer`). It specifies whether this service should be automatically started at a cluster node. Legal values are `true` or `false`. Default value is `false`.

Table D–21 (Cont.) invocation-scheme Subelements

Element	Required/ Optional	Description
<task-hung-threshold>	Optional	Specifies the amount of time in milliseconds that a task can execute before it is considered "hung". Note: a posted task that has not yet started is never considered as hung. This attribute is applied only if the Thread pool is used (the `thread-count` value is positive). Legal values are positive integers or zero (indicating no default timeout). Default value is the `task-hung-threshold` value specified in the `tangosol-coherence.xml` descriptor. See the `task-hung-threshold` parameter in "InvocationService Parameters" on page I-8.
<task-timeout>	Optional	Specifies the default timeout value in milliseconds for tasks that can be timed-out (for example, implement the `com.tangosol.net.PriorityTask` interface), but don't explicitly specify the task execution timeout value. The task execution time is measured on the server side and does not include the time spent waiting in a service backlog queue before being started. This attribute is applied only if the thread pool is used (the `thread-count` value is positive). Legal values are positive integers or zero (indicating no default timeout). Default value is the `task-timeout` value specified in the `tangosol-coherence.xml` descriptor. See the `task-timeout` parameter in "InvocationService Parameters" on page I-8.
<request-timeout>	Optional	Specifies the default timeout value in milliseconds for requests that can time-out (for example, implement the `com.tangosol.net.PriorityTask` interface), but don't explicitly specify the request timeout value. The request time is measured on the client side as the time elapsed from the moment a request is sent for execution to the corresponding server node(s) and includes the following: (1) the time it takes to deliver the request to an executing node (server); (2) the interval between the time the task is received and placed into a service queue until the execution starts; (3) the task execution time; (4) the time it takes to deliver a result back to the client. Legal values are positive integers or zero (indicating no default timeout). Default value is the `request-timeout` value specified in the `tangosol-coherence.xml` descriptor. See the `request-timeout` parameter in "InvocationService Parameters" on page I-8.

invocation-service-proxy

Used in: proxy-config

Description

The `invocation-service-proxy` element contains the configuration info for an invocation service proxy managed by a proxy service.

Elements

Table D–22 describes the elements you can define within the `invocation-service-proxy` element.

Table D–22 invocation-service-proxy Subelement

Element	Required/ Optional	Description
`<enabled>`	Optional	Specifies whether the invocation service proxy is enabled. If disabled, clients will not be able to execute Invocable objects on the proxy service JVM. Legal values are `true` or `false`. Default value is `true`.

jms-acceptor

Used in: acceptor-config.

Description

The `jms-acceptor` element specifies the configuration info for a connection acceptor that accepts connections from Coherence*Extend clients over JMS. For additional details and example configurations see Chapter 18, "Configuring and Using Coherence*Extend."

Elements

Table D–23 describes the elements you can define within the `jms-acceptor` element.

Table D–23 jms-acceptor Subelements

Element	Required/ Optional	Description
`<queue-connection-factory-name>`	Required	Specifies the JNDI name of the JMS `QueueConnectionFactory` used by the connection acceptor.
`<queue-name>`	Required	Specifies the JNDI name of the JMS Queue used by the connection acceptor.

jms-initiator

Used in: initiator-config.

Description

The `jms-initiator` element specifies the configuration info for a connection initiator that enables Coherence*Extend clients to connect to a remote cluster by using JMS. For additional details and example configurations see Chapter 18, "Configuring and Using Coherence*Extend."

Elements

The following table describes the elements you can define within the jms-initiator element.

Table D–24 jms-initiator Subelements

Element	Required/ Optional	Description
`<queue-connection-factory-name>`	Required	Specifies the JNDI name of the JMS `QueueConnectionFactory` used by the connection initiator.
`<queue-name>`	Required	Specifies the JNDI name of the JMS Queue used by the connection initiator.
`<connect-timeout>`	Optional	Specifies the maximum amount of time to wait while establishing a connection with a connection acceptor. The value of this element must be in the following format: `[\d]+[[.][\d]+]?[MS|ms|S|s|M|m|H|h|D|d]?` where the first non-digits (from left to right) indicate the unit of time duration: ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) If the value does not contain a unit, a unit of milliseconds is assumed. Default value is an infinite timeout.

key-associator

Used in: distributed-scheme

Description

Specifies an implementation of a `com.tangosol.net.partition.KeyAssociator` which will be used to determine associations between keys, allowing related keys to reside on the same partition.

Alternatively the cache's keys may manage the association by implementing the `com.tangosol.net.cache.KeyAssociation` interface.

Elements

Table D–25 describes the elements you can define within the key-associator element.

Table D–25 key-associator Subelements

Element	Required/ Optional	Description
`<class-name>`	Required	The name of a class that implements the `com.tangosol.net.partition.KeyAssociator` interface. This implementation must have a zero-parameter public constructor. Default value is the value of the `key-associator/class-name` parameter specified in the `tangosol.coherence.xml` descriptor. See "DistributedCache Service Parameters" on page I-3 for more information.

key-partitioning

Used in: distributed-scheme

Description

Specifies an implementation of a `com.tangosol.net.partition.KeyPartitioningStrategy` which will be used to determine the partition in which a key will reside.

Elements

Table D–26 describes the elements you can define within the key-partitioning element.

Table D–26 key-partitioning Subelements

Element	Required/Optional	Description
`<class-name>`	Required	The name of a class that implements the `com.tangosol.net.partition.KeyPartitioningStrategy` interface. This implementation must have a zero-parameter public constructor. Default value is the value of the `key-partitioning/class-name` parameter specified in the `tangosol-coherence.xml` descriptor. See "DistributedCache Service Parameters" on page I-3 for more information.

lh-file-manager

Used in: external-scheme, paged-external-scheme, async-store-manager.

Description

Configures a store manager which will use a Coherence LH on disk embedded database for storage. See "Persistent Cache on Disk" on page F-3 and "In-memory Cache with Disk Based Overflow" on page F-4 for examples of LH-based store configurations.

Implementation

Implemented by the `com.tangosol.io.lh.LHBinaryStoreManager` class. The `BinaryStore` objects created by this class are instances of javadoc:`com.tangosol.io.lh.LHBinaryStore`.

Elements

Table D–27 describes the elements you can define within the `lh-file-manager` element.

Table D–27 lh-file-manager Subelements

Element	Required/ Optional	Description
`<class-name>`	Optional	Specifies a custom implementation of the LH `BinaryStoreManager`. Any custom implementation must extend the `com.tangosol.io.lh.LHBinaryStoreManager` class and declare the exact same set of public constructors.
`<init-params>`	Optional	Specifies initialization parameters, for use in custom LH file manager implementations which implement the `com.tangosol.run.xml.XmlConfigurable` interface.
`<directory>`	Optional	Specifies the path name for the root directory that the LH file manager will use to store files in. If not specified or specifies a non-existent directory, a temporary file in the default location will be used.
`<file-name>`	Optional	Specifies the name for a non-temporary (persistent) file that the LH file manager will use to store data in. Specifying this parameter will cause the lh-file-manager to use non-temporary database instances. Use this parameter **only** for local caches that are backed by a cache loader from a non-temporary file: this allows the local cache to be pre-populated from the disk file on startup. When specified it is recommended that it use the `{cache-name}` macro described in Appendix E, "Cache Configuration Parameter Macros" macro. Normally this parameter should be left unspecified, indicating that temporary storage is to be used.

listener

Used in: local-scheme, external-scheme, paged-external-scheme, distributed-scheme, replicated-scheme, optimistic-scheme, near-scheme, versioned-near-scheme, overflow-scheme, read-write-backing-map-scheme, versioned-backing-map-scheme.

Description

The Listener element specifies an implementation of a `com.tangosol.util.MapListener` which will be notified of events occurring on a cache.

Elements

The following table describes the elements you can define within the listener element.

Table D–28 listener Subelement

Element	Required/ Optional	Description
<class-scheme>	Required	Specifies the full class name of the listener implementation to use. The specified class must implement the `com.tangosol.util.MapListener` interface.

local-scheme

Used in: caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme, near-scheme, versioned-near-scheme, overflow-scheme, read-write-backing-map-scheme, versioned-backing-map-scheme

Description

Local cache schemes define in-memory "local" caches. Local caches are generally nested within other cache schemes, for instance as the front-tier of a near-scheme. See "Local Cache of a Partitioned Cache (Near cache)" on page F-7 for examples of various local cache configurations.

Implementation

Local caches are implemented by the `com.tangosol.net.cache.LocalCache` class.

Cache of an External Store

A local cache may be backed by an external cache store (see "cachestore-scheme" on page D-17). Cache misses will read-through to the back end store to retrieve the data. If a writable store is provided, cache writes will propagate to the cache store as well. For optimizing read/write access against a cache store, see the "read-write-backing-map-scheme" on page D-71.

Size Limited Cache

The cache may be configured as size-limited, which means that when it reaches its maximum allowable size (see `<allowable-size>` subelement) it prunes itself back to a specified smaller size (see `<low-units>` subelement), choosing which entries to evict according to its eviction-policy (see `<eviction-policy>` subelement). The entries and size limitations are measured in terms of units as calculated by the scheme's unit-calculator (see `<unit-calculator>` subelement).

Entry Expiration

The local cache supports automatic expiration of entries based on the age of the value, as configured by the expiry-delay (see `<expiry-delay>` subelement).

Elements

Table D–29 describes the elements you can define within the `local-scheme` element.

Table D–29 local-scheme Subelements

Element	Required/ Optional	Description
`<scheme-name>`	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
`<scheme-ref>`	Optional	Specifies the name of another scheme to inherit from. See "Scheme Inheritance" on page D-19 for more information.
`<service-name>`	Optional	Specifies the name of the service which will manage caches created from this scheme. Services are configured from within the `<services>` element in the `tangosol-coherence.xml` descriptor. See Appendix H, "Operational Configuration Elements" for more information.

Table D–29 (Cont.) local-scheme Subelements

Element	Required/ Optional	Description
<class-name>	Optional	Specifies a custom implementation of the local cache. Any custom implementation must extend the com.tangosol.net.cache.LocalCache class and declare the exact same set of public constructors.
<init-params>	Optional	Specifies initialization parameters, for use in custom local cache implementations which implement the com.tangosol.run.xml.XmlConfigurable interface.
<listener>	Optional	Specifies an implementation of a com.tangosol.util.MapListener which will be notified of events occurring on the cache.
<cachestore-scheme>	Optional	Specifies the store which is being cached. If unspecified the cached data will only reside in memory, and only reflect operations performed on the cache itself.
<eviction-policy>	Optional	Specifies the type of eviction policy to use.Legal values are: ▪ LRU - Least Recently Used eviction policy chooses which entries to evict based on how recently they were last accessed, evicting those that were not accessed the for the longest period first. ▪ LFU - Least Frequently Used eviction policy chooses which entries to evict based on how often they are being accessed, evicting those that are accessed least frequently first. ▪ HYBRID - Hybrid eviction policy chooses which entries to evict based the combination (weighted score) of how often and recently they were accessed, evicting those that are accessed least frequently and were not accessed for the longest period first. ▪ <class-scheme> - A custom eviction policy, specified as a class-scheme. The class specified within this scheme must implement the com/tangosol/net/cache/OldCache.EvictionPolicy interface. Default value is HYBRID.
<high-units>	Optional	Used to limit the size of the cache. Contains the maximum number of units that can be placed in the cache before pruning occurs. An entry is the unit of measurement, unless it is overridden by an alternate unit-calculator (see <unit-calculator> subelement). When this limit is exceeded, the cache will begin the pruning process, evicting entries according to the eviction policy until the low-units (see <low-units> subelement) size is reached. Legal values are positive integers or zero. Zero implies no limit. Default value is 0.
<low-units>	Optional	Contains the number of units that the cache will be pruned down to when pruning takes place. An entry is the unit of measurement, unless it is overridden by an alternate unit-calculator (see <unit-calculator> subelement). When pruning occurs entries will continue to be evicted according to the eviction policy until this size. Legal values are positive integers or zero. Zero implies the default. Default value is 75% of the high-units setting (that is, for a high-units setting of 1000 the default low-units will be 750).

Table D–29 (Cont.) local-scheme Subelements

Element	Required/ Optional	Description
`<unit-calculator>`	Optional	Specifies the type of unit calculator to use. A unit calculator is used to determine the cost (in "units") of a given object.Legal values are: ■ `FIXED` - A unit calculator that assigns an equal weight of 1 to all cached objects. ■ `BINARY` - A unit calculator that assigns an object a weight equal to the number of bytes of memory required to cache the object. This requires that the objects are `com.tangosol.util.Binary` instances, as in a Partitioned cache. See `com.tangosol.net.cache.BinaryMemoryCalculator` for additional details. ■ <class-scheme> - A custom unit calculator, specified as a class-scheme. The class specified within this scheme must implement the `com/tangosol/net/cache/OldCache.UnitCalculator` interface. Default value is `FIXED`.
`<expiry-delay>`	Optional	Specifies the amount of time from last update that entries will be kept by the cache before being marked as expired. Any attempt to read an expired entry will result in a reloading of the entry from the configured cache store (see `<cachestore-scheme>`. Expired values are periodically discarded from the cache based on the flush-delay (see `<flush-delay>` subelement). The value of this element must be in the following format: `[\d]+[[.][\d]+]?[MS|ms|S|s|M|m|H|h|D|d]?` where the first non-digits (from left to right) indicate the unit of time duration: ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) If the value does not contain a unit, a unit of seconds is assumed. A value of zero implies no expiry. Default value is zero.
`<flush-delay>`	Optional	Specifies the time interval between periodic cache flushes, which will discard expired entries from the cache, thus freeing resources.The value of this element must be in the following format: `[\d]+[[.][\d]+]?[MS|ms|S|s|M|m|H|h|D|d]?` where the first non-digits (from left to right) indicate the unit of time duration: ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) If the value does not contain a unit, a unit of seconds is assumed. If `<expiry-delay>` is enabled, the default flush-delay is 1m, otherwise a default of zero is used and automatic flushes are disabled.

near-scheme

Used in: caching-schemes.

Description

The `near-scheme` defines a two-tier cache consisting of a front-tier (see `<front-scheme>` subelement) which caches a subset of a back-tier cache (see `<back-scheme>` subelement). The front-tier is generally a fast, size limited cache, while the back-tier is slower, but much higher capacity. A typical deployment might use a local-scheme for the front-tier, and a distributed-scheme for the back-tier. The result is that a portion of a large partitioned cache will be cached locally in-memory allowing for very fast read access. See Appendix B, "Types of Caches in Coherence," for a more detailed description of near caches, and "Local Cache of a Partitioned Cache (Near cache)" on page F-7 for an example of near cache configurations.

Implementation

The near scheme is implemented by the `com.tangosol.net.cache.NearCache` class.

Front-tier Invalidation

Specifying an invalidation-strategy (see `<invalidation-strategy>` subelement) defines a strategy that is used to keep the front tier of the near cache in sync with the back tier. Depending on that strategy a near cache is configured to listen to certain events occurring on the back tier and automatically update (or invalidate) the front portion of the near cache.

Elements

Table D–30 describes the elements you can define within the `near-scheme` element.

Table D–30 near-scheme Subelements

Element	Required/ Optional	Description
`<scheme-name>`	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
`<scheme-ref>`	Optional	Specifies the name of another scheme to inherit from. See "Scheme Inheritance" on page D-19 for more information
`<class-name>`	Optional	Specifies a custom implementation of the near cache. Any custom implementation must extend the `com.tangosol.net.cache.NearCache` class and declare the exact same set of public constructors.
`<init-params>`	Optional	Specifies initialization parameters for custom near cache implementations which implement the `com.tangosol.run.xml.XmlConfigurable` interface.
`<listener>`	Optional	Specifies an implementation of a `com.tangosol.util.MapListener` which will be notified of events occurring on the cache.

Table D–30 (Cont.) near-scheme Subelements

Element	Required/ Optional	Description

`<front-scheme>`	Required	Specifies the `cache-scheme` to use in creating the front-tier cache.Legal values are:
		■ `local-scheme`
		■ `external-scheme`
		■ `paged-external-scheme`
		■ `class-scheme`
		The eviction policy of the front-scheme defines which entries will be cached locally. For example:
		`<front-scheme>`
		`  <local-scheme>`
		`  <eviction-policy>HYBRID</eviction-policy>`
		`  <high-units>1000</high-units>`
		`  </local-scheme>`
		`</front-scheme>`

Table D–30 (Cont.) near-scheme Subelements

Element	Required/ Optional	Description
<back-scheme>	Required	Specifies the cache-scheme to use in creating the back-tier cache. Legal values are: ■ distributed-scheme ■ replicated-scheme ■ optimistic-scheme ■ local-scheme ■ external-scheme ■ paged-external-scheme ■ class-scheme ■ remote-cache-scheme For example: <back-scheme> <distributed-scheme> <scheme-ref>default-distributed</scheme-ref> </distributed-scheme> </back-scheme>
<invalidation-strategy>	Optional	Specifies the strategy used keep the front-tier in-sync with the back-tier. Please see com.tangosol.net.cache.NearCache for more details.Legal values are: ■ none - instructs the cache not to listen for invalidation events at all. This is the best choice for raw performance and scalability when business requirements permit the use of data which might not be absolutely current. Freshness of data can be guaranteed by use of a sufficiently brief eviction policy. The worst case performance is identical to a standard Distributed cache. ■ present - instructs the near cache to listen to the back map events related **only** to the items currently present in the front map. This strategy works best when cluster nodes have sticky data access patterns (for example, HTTP session management with a sticky load balancer). ■ all - instructs the near cache to listen to **all** back map events. This strategy is optimal for read-heavy access patterns where there is significant overlap between the front caches on each cluster member. ■ auto - instructs the near cache to switch between present and all strategies automatically based on the cache statistics. Default value is auto.
<autostart>	Optional	The autostart element is intended to be used by cache servers (that is, com.tangosol.net.DefaultCacheServer). It specifies whether the cache services associated with this cache scheme should be automatically started at a cluster node. Legal values are true or false. Default value is false.

nio-file-manager

Used in: external-scheme, paged-external-scheme, async-store-manager.

Description

Configures an external store which uses memory-mapped file for storage.

Implementation

This store manager is implemented by the com.tangosol.io.nio.MappedStoreManager class. The BinaryStore objects created by this class are instances of the com.tangosol.io.nio.BinaryMapStore.

Elements

Table D–31 describes the elements you can define within the nio-file-manager element.

Table D–31 nio-file-manager Subelements

Element	Required/ Optional	Description
<class-name>	Optional	Specifies a custom implementation of the local cache. Any custom implementation must extend the com.tangosol.io.nio.MappedStoreManager class and declare the exact same set of public constructors.
<init-params>	Optional	Specifies initialization parameters, for use in custom nio-file-manager implementations which implement the com.tangosol.run.xml.XmlConfigurable interface.
<initial-size>	Optional	Specifies the initial buffer size in megabytes.The value of this element must be in the following format: [\d]+[[.][\d]+]?[K|k|M|m|G|g]?[B|b]? where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied: ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) If the value does not contain a factor, a factor of mega is assumed. Legal values are positive integers between 1 and Integer.MAX_VALUE - 1023 (that is, 2,147,482,624 bytes). Default value is 1MB.
<maximum-size>	Optional	Specifies the maximum buffer size in bytes.The value of this element must be in the following format: [\d]+[[.][\d]+]?[K|k|M|m|G|g]?[B|b]? where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied: ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) If the value does not contain a factor, a factor of mega is assumed. Legal values are positive integers between 1 and Integer.MAX_VALUE - 1023 (that is, 2,147,482,624 bytes). Default value is 1024MB.
<directory>	Optional	Specifies the path name for the root directory that the manager will use to store files in. If not specified or specifies a non-existent directory, a temporary file in the default location will be used.

nio-memory-manager

Used in: external-scheme, paged-external-scheme, async-store-manager.

Description

Configures a store-manager which uses an off JVM heap, memory region for storage, which means that it does not affect the Java heap size and the related JVM garbage-collection performance that can be responsible for application pauses. See "NIO In-memory Cache" on page F-2 for an example of an NIO cache configuration.

> **Note:** Some JVMs (starting with 1.4) require the use of a command line parameter if the total NIO buffers will be greater than 64MB. For example: `-XX:MaxDirectMemorySize=512M`

Implementation

Implemented by the `com.tangosol.io.nio.DirectStoreManager` class. The `BinaryStore` objects created by this class are instances of the `com.tangosol.io.nio.BinaryMapStore`.

Elements

Table D–32 describes the elements you can define within the `nio-memory-manager` element.

Table D–32 nio-memory-manager Subelements

Element	Required/ Optional	Description
`<class-name>`	Optional	Specifies a custom implementation of the local cache. Any custom implementation must extend the `com.tangosol.io.nio.DirectStoreManager` class and declare the exact same set of public constructors.

Table D–32 (Cont.) nio-memory-manager Subelements

Element	Required/ Optional	Description
<init-params>	Optional	Specifies initialization parameters, for use in custom `nio-memory-manager` implementations which implement the `com.tangosol.run.xml.XmlConfigurable` interface.
<initial-size>	Optional	Specifies the initial buffer size in bytes. The value of this element must be in the following format: `[\d]+[[.][\d]+]?[K|k|M|m|G|g]?[B|b]?` where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied: ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) If the value does not contain a factor, a factor of mega is assumed. Legal values are positive integers between 1 and `Integer.MAX_VALUE` - 1023 (that is, 2,147,482,624 bytes). Default value is 1MB.
<maximum-size>	Optional	Specifies the maximum buffer size in bytes. The value of this element must be in the following format: `[\d]+[[.][\d]+]?[K|k|M|m|G|g]?[B|b]?` where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied: ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) If the value does not contain a factor, a factor of mega is assumed. Legal values are positive integers between 1 and `Integer.MAX_VALUE` - 1023 (that is, 2,147,482,624 bytes). Default value is 1024MB.

operation-bundling

Used in: cachestore-scheme, distributed-scheme, remote-cache-scheme.

Description

The operation-bundling element specifies the configuration info for a particular bundling strategy.

Bundling is a process of coalescing multiple individual operations into "bundles". It could be beneficial when

- there is a continuous stream of operations on multiple threads in parallel;
- individual operations have relatively high latency (network or database-related); and
- there are functionally analogous "bulk" operations that take a collection of arguments instead of a single one without causing the latency to grow linearly (as a function of the collection size).

> **Note:** As with any bundling algorithm, there is a natural trade-off between the resource utilization and average request latency. Depending on a particular application usage pattern, enabling this feature may either help or hurt the overall application performance.

See com.tangosol.net.cache.AbstractBundler for additional implementation details.

Elements

Table D–33 describes the subelement for the operation-bundling element.

Table D–33 operation-bundling Subelement

Element	Required/ Optional	Description
<bundle-config>	Required	Describes one or more bundle-able operations.

optimistic-scheme

Used in: caching-schemes, near-scheme, versioned-near-scheme, overflow-scheme

The optimistic scheme defines a cache which fully replicates all of its data to all cluster nodes that run the service (see `<service-name>` subelement). See Appendix B, "Types of Caches in Coherence" for a more detailed description of optimistic caches.

Optimistic Locking

Unlike the replicated-scheme and distributed-scheme caches, optimistic caches do not support concurrency control (locking). Individual operations against entries are atomic but there is no guarantee that the value stored in the cache does not change between atomic operations. The lack of concurrency control allows optimistic caches to support very fast write operations.

Cache Storage (Backing Map)

Storage for the cache is specified by using the backing-map-scheme (see `<backing-map-scheme>` subelement). For instance an optimistic cache which uses a `local-scheme` for its backing map will result in cache entries being stored in-memory.

Elements

Table D–34 describes the elements you can define within the `optimistic-scheme` element.

Table D–34 optimistic-scheme Subelements

Element	Required/ Optional	Description
`<scheme-name>`	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<scheme-ref>	Optional	Specifies the name of another scheme to inherit from. See "Scheme Inheritance" on page D-19 for more information.
<service-name>	Optional	Specifies the name of the service which will manage caches created from this scheme. Services are configured from within the `<services>` parameter in `tangosol-coherence.xml`. See Appendix H, "Operational Configuration Elements" for more information.

Table D–34 (Cont.) optimistic-scheme Subelements

Element	Required/ Optional	Description
<listener>	Optional	Specifies an implementation of a com.tangosol.util.MapListener which will be notified of events occurring on the cache.
<backing-map-scheme>	Optional	Specifies what type of cache will be used within the cache server to store the entries.Legal values are: ■ local-scheme ■ external-scheme ■ paged-external-scheme ■ overflow-scheme ■ class-scheme To ensure cache coherence, the backing-map of an optimistic cache must not use a read-through pattern to load cache entries. Either use a cache-aside pattern from outside the cache service, or switch to the distributed-scheme, which supports read-through clustered caching.
<autostart>	Optional	The autostart element is intended to be used by cache servers (that is, com.tangosol.net.DefaultCacheServer). It specifies whether the cache services associated with this cache scheme should be automatically started at a cluster node. Legal values are true or false. Default value is false.

outgoing-message-handler

Used in: `acceptor-config`, `initiator-config`.

Description

The `outgoing-message-handler` specifies the configuration info used to detect dropped client-to-cluster connections. For connection initiators and acceptors that use connectionless protocols (for example, JMS), this information is necessary to proactively detect and release resources allocated to dropped connections. Connection-oriented initiators and acceptors can also use this information as an additional mechanism to detect dropped connections.

Elements

Table D–35 describes the elements you can define within the outgoing-message-handler element.

Table D–35 outgoing-message-handler Subelements

Element	Required/ Optional	Description
`<heartbeat-interval>`	Optional	Specifies the interval between ping requests. A ping request is used to ensure the integrity of a connection.The value of this element must be in the following format: `[\d]+[[.][\d]+]?[MS|ms|S|s|M|m|H|h|D|d]?` where the first non-digits (from left to right) indicate the unit of time duration: ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) If the value does not contain a unit, a unit of milliseconds is assumed. A value of zero disables ping requests. The default value is zero.
`<heartbeat-timeout>`	Optional	Specifies the maximum amount of time to wait for a response to a ping request before declaring the underlying connection unusable.The value of this element must be in the following format: `[\d]+[[.][\d]+]?[MS|ms|S|s|M|m|H|h|D|d]?` where the first non-digits (from left to right) indicate the unit of time duration: ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) If the value does not contain a unit, a unit of milliseconds is assumed. The default value is the value of the `request-timeout` element.
`<request-timeout>`	Optional	Specifies the maximum amount of time to wait for a response message before declaring the underlying connection unusable. The value of this element must be in the following format: `[\d]+[[.][\d]+]?[MS|ms|S|s|M|m|H|h|D|d]?` where the first non-digits (from left to right) indicate the unit of time duration: ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) If the value does not contain a unit, a unit of milliseconds is assumed. The default value is an infinite timeout.

overflow-scheme

Used in: caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme, read-write-backing-map-scheme, versioned-backing-map-scheme.

Description

The overflow-scheme defines a two-tier cache consisting of a fast, size limited front-tier, and slower but much higher capacity back-tier cache. When the size limited front fills up, evicted entries are transparently moved to the back. In the event of a cache miss, entries may move from the back to the front. A typical deployment might use a local-scheme for the front-tier, and a external-scheme for the back-tier, allowing for fast local caches with capacities larger the JVM heap would allow. In such a deployment the local-scheme element's high-units and eviction-policy will control the transfer (eviction) of entries from the front to back caches.

Note: Relying on overflow for normal cache storage is not recommended. It should only be used to help avoid eviction-related data loss in the case where the storage requirements temporarily exceed the configured capacity. In general, the overflow's on disk storage should remain empty.

Implementation

Implemented by either com.tangosol.net.cache.OverflowMap or com.tangosol.net.cache.SimpleOverflowMap, see expiry-enabled for details.

Entry Expiration

Overflow supports automatic expiration of entries based on the age of the value, as configured by the expiry-delay (see <expiry-delay> subelement).

Elements

Table D–36 describes the elements you can define within the overflow-scheme element.

Table D–36 overflow-scheme Subelements

Element	Required/ Optional	Description
<scheme-name>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<scheme-ref>	Optional	Specifies the name of another scheme to inherit from. See "Scheme Inheritance" on page D-19 for more information.
<class-name>	Optional	Specifies a custom implementation of the overflow cache. Any custom implementation must extend either the com.tangosol.net.cache.OverflowMap or com.tangosol.net.cache.SimpleOverflowMap class, and declare the exact same set of public constructors.
<init-params>	Optional	Specifies initialization parameters, for use in custom overflow cache implementations which implement the com.tangosol.run.xml.XmlConfigurable interface.
<listener>	Optional	Specifies an implementation of a com.tangosol.util.MapListener which will be notified of events occurring on the cache.

Table D–36 (Cont.) overflow-scheme Subelements

Element	Required/ Optional	Description
<front-scheme>	Required	Specifies the cache-scheme to use in creating the front-tier cache. Legal values are: ■ local-scheme ■ external-scheme ■ paged-external-scheme ■ class-scheme The eviction policy of the front-scheme defines which entries which items are in the front versus back tiers. For example: <front-scheme> <local-scheme> <eviction-policy>HYBRID</eviction-policy> <high-units>1000</high-units> </local-scheme> </front-scheme>
<back-scheme>	Required	Specifies the cache-scheme to use in creating the back-tier cache.Legal values are: ■ local-scheme ■ external-scheme ■ paged-external-scheme ■ class-scheme For Example: <back-scheme> <external-scheme> <lh-file-manager/> </external-scheme> </back-scheme>
<miss-cache-scheme>	Optional	Specifies a cache-scheme for maintaining information on cache misses. For caches which are not expiry-enabled (see <expiry-enabled> subelement), the miss-cache is used track keys which resulted in both a front and back tier cache miss. The knowledge that a key is not in either tier allows some operations to perform faster, as they can avoid querying the potentially slow back-tier. A size limited scheme may be used to control how many misses are tracked. If unspecified no cache-miss data will be maintained. Legal values are: ■ local-scheme

Table D–36 (Cont.) overflow-scheme Subelements

Element	Required/ Optional	Description
`<expiry-enabled>`	Optional	Turns on support for automatically-expiring data, as provided by the `com.tangosol.net.cache.CacheMap` API. When enabled the overflow-scheme will be implemented using `com.tangosol.net.cache.OverflowMap`, rather then `com.tangosol.net.cache.SimpleOverflowMap`. Legal values are `true` or `false`. Default value is `false`.
`<expiry-delay>`	Optional	Specifies the amount of time from last update that entries will be kept by the cache before being expired. Entries that are expired will not be accessible and will be evicted.The value of this element must be in the following format: `[\d]+[[.][\d]+]?[MS|ms|S|s|M|m|H|h|D|d]?` where the first non-digits (from left to right) indicate the unit of time duration: ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) If the value does not contain a unit, a unit of seconds is assumed. A value of zero implies no expiry. Default value is zero.
`<autostart>`	Optional	The `autostart` element is intended to be used by cache servers (that is, `com.tangosol.net.DefaultCacheServer`). It specifies whether the cache services associated with this cache scheme should be automatically started at a cluster node. Legal values are `true` or `false`. Default value is `false`.

paged-external-scheme

Used in: caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme, near-scheme, versioned-near-scheme, overflow-scheme, read-write-backing-map-scheme, versioned-backing-map-scheme

Description

As with external-scheme, paged-external-schemes define caches which are not JVM heap based, allowing for greater storage capacity. The paged-external-scheme optimizes LRU eviction by using a paging approach (see <paging> subelement). See Chapter 12, "Serialization Paged Cache," for a detailed description of the paged cache functionality.

Implementation

This scheme is implemented by the com.tangosol.net.cache.SerializationPagedCache class.

Paging

Cache entries are maintained over a series of pages, where each page is a separate com.tangosol.io.BinaryStore, obtained from the configured storage manager (see "Pluggable Storage Manager"). When a page is created it is considered to be the "current" page, and all write operations are performed against this page. On a configurable interval (see <page-duration> subelement) the current page is closed and a new current page is created. Read operations for a given key are performed against the last page in which the key was stored. When the number of pages exceeds a configured maximum (see <page-limit> subelement), the oldest page is destroyed and those items which were not updated since the page was closed are be evicted. For example configuring a cache with a duration of ten minutes per page, and a maximum of six pages, will result in entries being cached for at most an hour. Paging improves performance by avoiding individual delete operations against the storage manager as cache entries are removed or evicted. Instead the cache simply releases its references to those entries, and relies on the eventual destruction of an entire page to free the associated storage of all page entries in a single stroke.

Pluggable Storage Manager

External schemes use a pluggable store manager to create and destroy pages, and to access entries within those pages. Supported store-managers include:

- async-store-manager—a wrapper providing asynchronous write capabilities for of other store-manager implementations
- custom-store-manager—allows definition of custom implementations of store-managers
- bdb-store-manager—uses Berkeley Database JE to implement an on disk cache
- lh-file-manager—uses a Coherence LH on disk database cache
- nio-file-manager—uses NIO to implement memory-mapped file based cache
- nio-memory-manager—uses NIO to implement an off JVM heap, in-memory cache

Persistence (long-term storage)

Paged external caches are used for temporary storage of large data sets, for example as the back-tier of an overflow-scheme. These caches are not usable as for long-term storage (persistence), and will not survive beyond the life of the JVM. Clustered persistence should be configured by using a read-write-backing-map-scheme on a distributed-scheme. If a non-clustered persistent cache is what is needed, refer to "Persistence (long-term storage)" on page D-33.

Elements

Table D–37 describes the elements you can define within the `paged-external-scheme` element.

Table D–37 paged-external-scheme Subelements

Element	Required/ Optional	Description
`<scheme-name>`	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
`<scheme-ref>`	Optional	Specifies the name of another scheme to inherit from. See "Scheme Inheritance" on page D-19 for more information.
`<class-name>`	Optional	Specifies a custom implementation of the external paged cache. Any custom implementation must extend the com.tangosol.net.cache. SerializationPagedCache class and declare the exact same set of public constructors.
`<init-params>`	Optional	Specifies initialization parameters, for use in custom external paged cache implementations which implement the `com.tangosol.run.xml.XmlConfigurable` interface.
<listener>	Optional	Specifies an implementation of a `com.tangosol.util.MapListener` which will be notified of events occurring on the cache.
`<page-limit>`	Required	Specifies the maximum number of active pages for the paged cache. Legal values are positive integers between 2 and 3600.
`<page-duration>`	Optional	Specifies the length of time, in seconds, that a page in the paged cache is current.The value of this element must be in the following format: `[\d]+[[.][\d]+]?[MS|ms|S|s|M|m|H|h|D|d]?` where the first non-digits (from left to right) indicate the unit of time duration: ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) If the value does not contain a unit, a unit of seconds is assumed. Legal values are between 5 and 604800 seconds (one week) and zero (no expiry). Default value is zero
`<async-store-manager>`	Optional	Configures the paged external cache to use an asynchronous storage manager wrapper for any other storage manager. See "Pluggable Storage Manager" on page D-32 for more information.
`<custom-store-manager>`	Optional	Configures the paged external cache to use a custom storage manager implementation.

Table D–37 (Cont.) paged-external-scheme Subelements

Element	Required/ Optional	Description
<bdb-store-manager>	Optional	Configures the paged external cache to use Berkeley Database JE on disk databases for cache storage.
<lh-file-manager>	Optional	Configures the paged external cache to use a Coherence LH on disk database for cache storage.
<nio-file-manager>	Optional	Configures the paged external cache to use a memory-mapped file for cache storage.
<nio-memory-manager>	Optional	Configures the paged external cache to use an off JVM heap, memory region for cache storage.

partition-listener

Used in: distributed-scheme

Description

Specifies an implementation of a `com.tangosol.net.partition.PartitionListener` interface, which allows receiving partition distribution events.

Elements

Table D–38 describes the elements you can define within the `partition-listener` element.

Table D–38 partition-listener Subelements

Element	Required/ Optional	Description
`<class-name>`	Required	The name of a class that implements the `com.tangosol.net.partition.PartitionListener` interface. This implementation must have a zero-parameter public constructor. Default value is the value specified in the `partition-listener/class-name` parameter in the `tangosol-coherence.xml` descriptor. See "DistributedCache Service Parameters" on page I-3 for more information.

proxy-config

Used in: proxy-scheme.

Description

The `proxy-config` element specifies the configuration info for the clustered service proxies managed by a proxy service. A service proxy is an intermediary between a remote client (connected to the cluster by using a connection acceptor) and a clustered service used by the remote client.

Elements

Table D–39 describes the elements you can define within the `proxy-config` element.

Table D–39 proxy-config Subelements

Element	Required/ Optional	Description
<cache-service-proxy>	Optional	Specifies the configuration info for a cache service proxy managed by the proxy service.
<invocation-service-proxy>	Optional	Specifies the configuration info for an invocation service proxy managed by the proxy service.

proxy-scheme

Used in: caching-schemes.

Description

The proxy-scheme element contains the configuration info for a clustered service that allows Coherence*Extend clients to connect to the cluster and use clustered services without having to join the cluster.

Elements

Table D–40 describes the subelements you can define within the proxy-scheme element.

Table D–40 proxy-scheme Subelements

Element	Required/Optional	Description
<scheme-name>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<scheme-ref>	Optional	Specifies the name of another scheme to inherit from. See "Scheme Inheritance" on page D-19 for more information.
<service-name>	Optional	Specifies the name of the service.
<serializer>	Optional	Specifies the class configuration info for a com.tangosol.io.Serializer implementation used by the Proxy service to serialize and deserialize user types. For example, the following configures a ConfigurablePofContext that uses the default coherence-pof-types.xml configuration file to write objects to and read from a stream: <serializer> <class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name> </serializer>
<thread-count>	Optional	Specifies the number of daemon threads used by the service. If zero, all relevant tasks are performed on the service thread. Legal values are positive integers or zero. Default value is the value specified in the thread-count parameter of the tangosol-coherence.xml descriptor. See "ProxyService Parameters" on page I-9 for more information.
<acceptor-config>	Required	Contains the configuration of the connection acceptor used by the service to accept connections from Coherence*Extend clients and to allow them to use the services offered by the cluster without having to join the cluster.
<proxy-config>	Optional	Contains the configuration of the clustered service proxies managed by this service.
<autostart>	Optional	The autostart element is intended to be used by cache servers (that is, com.tangosol.net.DefaultCacheServer). It specifies whether this service should be automatically started at a cluster node. Legal values are true or false. Default value is false.

read-write-backing-map-scheme

Used in: caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme.

Description

The read-write-backing-map-scheme defines a backing map which provides a size limited cache of a persistent store. See "Read-Through, Write-Through, Write-Behind Caching and Refresh-Ahead" for more details.

Implementation

The `read-write-backing-map-scheme` is implemented by the `com.tangosol.net.cache.ReadWriteBackingMap` class.

Cache of an External Store

A read write backing map maintains a cache backed by an external persistent cache store (see `<cachestore-scheme>` subelement), cache misses will read-through to the back-end store to retrieve the data. If a writable store is provided, cache writes will propagate to the cache store as well.

Refresh-Ahead Caching

When enabled (see `<refreshahead-factor>` subelement) the cache will watch for recently accessed entries which are about to expire, and asynchronously reload them from the cache store. This insulates the application from potentially slow reads against the cache store, as items periodically expire.

Write-Behind Caching

When enabled (see `<write-delay>` subelement) the cache will delay writes to the back-end cache store. This allows for the writes to be batched (see `<write-batch-factor>` subelement) into more efficient update blocks, which occur asynchronously from the client thread.

Elements

The following table describes the elements you can define within the `read-write-backing-map-scheme` element.

Table D–41 read-write-backing-map-scheme Subelements

Element	Required/ Optional	Description
`<scheme-name>`	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
`<scheme-ref>`	Optional	Specifies the name of another scheme to inherit from. See "Scheme Inheritance" on page D-19 for more information.
`<class-name>`	Optional	Specifies a custom implementation of the read write backing map. Any custom implementation must extend the `com.tangosol.net.cache.ReadWriteBackingMap` class and declare the exact same set of public constructors.

Table D–41 (Cont.) read-write-backing-map-scheme Subelements

Element	Required/ Optional	Description
<init-params>	Optional	Specifies initialization parameters, for use in custom read write backing map implementations which implement the com.tangosol.run.xml.XmlConfigurable interface.
<listener>	Optional	Specifies an implementation of a com.tangosol.util.MapListener which will be notified of events occurring on the cache.
<cachestore-scheme>	Optional	Specifies the store to cache. If unspecified the cached data will only reside within the internal cache (see <internal-cache-scheme> subelement), and only reflect operations performed on the cache itself.
<internal-cache-scheme>	Required	Specifies a cache-scheme which will be used to cache entries. Legal values are: ■ local-scheme ■ disk-scheme ■ external-scheme ■ paged-external-scheme ■ overflow-scheme ■ class-scheme
<miss-cache-scheme>	Optional	Specifies a cache-scheme for maintaining information on cache misses. The miss-cache is used track keys which were not found in the cache store. The knowledge that a key is not in the cache store allows some operations to perform faster, as they can avoid querying the potentially slow cache store. A size-limited scheme may be used to control how many misses are cached. If unspecified no cache-miss data will be maintained. Legal values are: ■ local-scheme
<read-only>	Optional	Specifies if the cache is read only. If true the cache will load data from cachestore for read operations and will not perform any writing to the cachestore when the cache is updated. Legal values are true or false. Default value is false.

Table D–41 (Cont.) read-write-backing-map-scheme Subelements

Element	Required/ Optional	Description
<write-delay>	Optional	Specifies the time interval for a write-behind queue to defer asynchronous writes to the cachestore by.The value of this element must be in the following format: `[\d]+[[.][\d]+]?[MS|ms|S|s|M|m|H|h|D|d]?` where the first non-digits (from left to right) indicate the unit of time duration: ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) If the value does not contain a unit, a unit of seconds is assumed. If zero, synchronous writes to the cachestore (without queuing) will take place, otherwise the writes will be asynchronous and deferred by specified time interval after the last update to the value in the cache. Default is zero.
`<write-batch-factor>`	Optional	The `write-batch-factor` element is used to calculate the "soft-ripe" time for write-behind queue entries.A queue entry is considered to be "ripe" for a write operation if it has been in the write-behind queue for no less than the write-delay interval. The "soft-ripe" time is the point in time before the actual ripe time after which an entry will be included in a batched asynchronous write operation to the `CacheStore` (along with all other ripe and soft-ripe entries). In other words, a soft-ripe entry is an entry that has been in the write-behind queue for at least the following duration: D' = (1.0 - F)*Dwhere:D = write-delay intervalF = write-batch-factor Conceptually, the write-behind thread uses the following logic when performing a batched update: **1.** The thread waits for a queued entry to become ripe. **2.** When an entry becomes ripe, the thread dequeues all ripe and soft-ripe entries in the queue. **3.** The thread then writes all ripe and soft-ripe entries either by using `store()` (if there is only the single ripe entry) or `storeAll()` (if there are multiple ripe/soft-ripe entries). **4.** The thread then repeats (1). This element is only applicable if asynchronous writes are enabled (that is, the value of the write-delay element is greater than zero) and the `CacheStore` implements the `storeAll()` method. The value of the element is expressed as a percentage of the write-delay interval. Legal values are nonnegative doubles less than or equal to 1.0. Default is zero.

Table D–41 (Cont.) read-write-backing-map-scheme Subelements

Element	Required/ Optional	Description
`<write-requeue-threshold>`	Optional	Specifies the maximum size of the write-behind queue for which failed cachestore write operations are requeued. The purpose of this setting is to prevent flooding of the write-behind queue with failed cachestore operations. This can happened in situations where a large number of successive write operations fail. If zero, write-behind requeuing is disabled. Legal values are positive integers or zero. Default is zero.
`<refresh-ahead-factor>`	Optional	The refresh-ahead-factor element is used to calculate the "soft-expiration" time for cache entries. Soft-expiration is the point in time before the actual expiration after which any access request for an entry will schedule an asynchronous load request for the entry. This attribute is only applicable if the internal cache is a `local-scheme`, configured with the `<expiry-delay>` subelement. The value is expressed as a percentage of the internal `LocalCache` expiration interval. If zero, refresh-ahead scheduling will be disabled. If 1.0, then any get operation will immediately trigger an asynchronous reload. Legal values are nonnegative doubles less than or equal to 1.0. Default value is zero.
`<rollback-cachestore-failures>`	Optional	Specifies whether exceptions caught during synchronous cachestore operations are rethrown to the calling thread (possibly over the network to a remote member). If the value of this element is false, an exception caught during a synchronous cachestore operation is logged locally and the internal cache is updated. If the value is true, the exception is rethrown to the calling thread and the internal cache is not changed. If the operation was called within a transactional context, this would have the effect of rolling back the current transaction. Legal values are `true` or `false`. Default value is `false`.

remote-cache-scheme

Used in: cachestore-scheme, caching-schemes, near-scheme.

Description

The remote-cache-scheme element contains the configuration info necessary to use a clustered cache from outside the cluster by using Coherence*Extend.

Elements

The following table describes the elements you can define within the remote-cache-scheme element.

Table D–42 remote-cache-scheme Subelements

Element	Required/ Optional	Description
<scheme-name>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<scheme-ref>	Optional	Specifies the name of another scheme to inherit from. See "Scheme Inheritance" on page D-19 for more information.
<service-name>	Optional	Specifies the name of the service which will manage caches created from this scheme.
<operation-bundling>	Optional	Specifies the configuration info for a bundling strategy.
<initiator-config>	Required	Contains the configuration of the connection initiator used by the service to establish a connection with the cluster.

remote-invocation-scheme

Used in: caching-schemes

Description

The remote-invocation-scheme element contains the configuration info necessary to execute tasks within the context of a cluster without having to first join the cluster. This scheme uses Coherence*Extend to connect to the cluster.

Elements

Table D–43 describes the elements you can define within the remote-invocation-scheme element.

Table D–43 remote-invocation-scheme Subelements

Element	Required/ Optional	Description
<scheme-name>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<scheme-ref>	Optional	Specifies the name of another scheme to inherit from. See "Scheme Inheritance" on page D-19 for more information.
<service-name>	Optional	Specifies the name of the service.
<initiator-config>	Required	Contains the configuration of the connection initiator used by the service to establish a connection with the cluster.

replicated-scheme

Used in: caching-schemes, near-scheme, versioned-near-scheme, overflow-scheme, versioned-backing-map-scheme

Description

The replicated scheme defines caches which fully replicate all their cache entries on each cluster nodes running the specified service. See "Replicated Cache Service" for a more detailed description of replicated caches.

Clustered Concurrency Control

Replicated caches support cluster wide key-based locking so that data can be modified in a cluster without encountering the classic missing update problem. Note that any operation made without holding an explicit lock is still atomic but there is no guarantee that the value stored in the cache does not change between atomic operations.

Cache Storage (Backing Map)

Storage for the cache is specified by using the backing-map scheme (see <backing-map> subelement). For instance, a replicated cache which uses a local-scheme for its backing map will result in cache entries being stored in-memory.

Elements

Table D–44 describes the elements you can define within the replicated-scheme element.

Table D–44 replicated-scheme Subelements

Element	Required/ Optional	Description
<scheme-name>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<scheme-ref>	Optional	Specifies the name of another scheme to inherit from. See "Scheme Inheritance" on page D-19 for more information.
<service-name>	Optional	Specifies the name of the service which will manage caches created from this scheme. Services are configured from within the <services> element in the tangosol-coherence.xml file. See Appendix H, "Operational Configuration Elements" for more information.
<listener>	Optional	Specifies an implementation of a com.tangosol.util.MapListener which will be notified of events occurring on the cache.

Table D–44 (Cont.) replicated-scheme Subelements

Element	Required/ Optional	Description
<backing-map-scheme>	Optional	Specifies what type of cache will be used within the cache server to store the entries.Legal values are: ■ local-scheme ■ external-scheme ■ paged-external-scheme ■ overflow-scheme ■ class-scheme To ensure cache coherence, the backing-map of an replicated cache must not use a read-through pattern to load cache entries. Either use a cache-aside pattern from outside the cache service, or switch to the distributed-scheme, which supports read-through clustered caching.
<standard-lease-milliseconds>	Optional	Specifies the duration of the standard lease in milliseconds. When a lease has aged past this number of milliseconds, the lock will automatically be released. Set this value to zero to specify a lease that never expires. The purpose of this setting is to avoid deadlocks or blocks caused by stuck threads; the value should be set higher than the longest expected lock duration (for example, higher than a transaction timeout). It's also recommended to set this value higher than packet-delivery/timeout-milliseconds value. Legal values are from positive long numbers or zero. Default value is the value specified for packet-delivery/timeout-milliseconds in the tangosol-coherence.xml descriptor. See "ReplicatedCache Service Parameters" on page I-7 for more information.
<lease-granularity>	Optional	Specifies the lease ownership granularity. Available since release 2.3.Legal values are: ■ thread ■ member A value of thread means that locks are held by a thread that obtained them and can only be released by that thread. A value of member means that locks are held by a cluster node and any thread running on the cluster node that obtained the lock can release it. Default value is the lease-granularity value specified in the tangosol-coherence.xml descriptor. See "ReplicatedCache Service Parameters" on page I-7 for more information.
<mobile-issues>	Optional	Specifies whether the lease issues should be transferred to the most recent lock holders. Legal values are true or false. Default value is the mobile-issue value specified in the tangosol-coherence.xml descriptor. See "ReplicatedCache Service Parameters" on page I-7 for more information.
<autostart>	Optional	The autostart element is intended to be used by cache servers (that is, com.tangosol.net.DefaultCacheServer). It specifies whether the cache services associated with this cache scheme should be automatically started at a cluster node. Legal values are true or false. Default value is false.

tcp-acceptor

Used in: acceptor-config.

Description

The `tcp-initiator` element specifies the configuration info for a connection acceptor that accepts connections from Coherence*Extend clients over TCP/IP. For additional details and example configurations see Chapter 18, "Configuring and Using Coherence*Extend."

Elements

Table D–45 describes the elements you can define within the `tcp-acceptor` element.

Table D–45 tcp-acceptor Subelements

Element	Required/ Optional	Description

`<local-address>`	Required	Specifies the local address (IP or DNS name) and port that the TCP/IP ServerSocket opened by the connection acceptor will listen on.For example, the following will instruct the connection acceptor to bind the TCP/IP ServerSocket to 192.168.0.2:9099:
		`<local-address>`
		`  <address>192.168.0.2</address>`
		`  <port>9099</port>`
		`  <reusable>true</reusable>`
		`</local-address>`
		The `<reusable>` child element specifies whether a TCP/IP socket can be bound to an address if a previous connection is in a timeout state. When a TCP/IP connection is closed the connection may remain in a timeout state for a period after the connection is closed (typically known as the `TIME_WAIT` state or 2MSL wait state). For applications using a well known socket address or port it may not be possible to bind a socket to a required address if there is a connection in the timeout state involving the socket address or port.

`<keep-alive-enabled>`	Optional	Indicates whether keep alive (`SO_KEEPALIVE`) is enabled on a TCP/IP socket. Valid values are true and false. Keep alive is enabled by default.
`<tcp-delay-enabled>`	Optional	Indicates whether TCP delay (Nagle's algorithm) is enabled on a TCP/IP socket. Valid values are `true` and `false`. TCP delay is disabled by default.
`<receive-buffer-size>`	Optional	Configures the size of the underlying TCP/IP socket network receive buffer.Increasing the receive buffer size can increase the performance of network I/O for high-volume connections, while decreasing it can help reduce the backlog of incoming data.The value of this element must be in the following format: `[\d]+[[.][\d]+]?[K|k|M|m|G|g]?[B|b]?` where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied: ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) If the value does not contain a factor, a factor of one is assumed. Default value is O/S dependent.

Table D–45 (Cont.) tcp-acceptor Subelements

Element	Required/ Optional	Description
`<send-buffer-size>`	Optional	Configures the size of the underlying TCP/IP socket network send buffer.The value of this element must be in the following format: `[\d]+[[.][\d]+]?[K|k|M|m|G|g]?[B|b]?` where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied: ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) If the value does not contain a factor, a factor of one is assumed. Default value is O/S dependent.
`<listen-backlog>`	Optional	Configures the size of the TCP/IP server socket backlog queue. Valid values are positive integers. Default value is O/S dependent.
`<linger-timeout>`	Optional	Enables `SO_LINGER` on a TCP/IP socket with the specified linger time.The value of this element must be in the following format: `[\d]+[[.][\d]+]?[MS|ms|S|s|M|m|H|h|D|d]?` where the first non-digits (from left to right) indicate the unit of time duration: ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) If the value does not contain a unit, a unit of milliseconds is assumed. Linger is disabled by default.
`<authorized-hosts>`	Optional	A collection of IP addresses of TCP/IP initiator hosts that are allowed to connect to this TCP/IP acceptor.

tcp-initiator

Used in: initiator-config.

Description

The tcp-initiator element specifies the configuration info for a connection initiator that enables Coherence*Extend clients to connect to a remote cluster by using TCP/IP. For additional details and example configurations see Chapter 18, "Configuring and Using Coherence*Extend."

Elements

Table D–46 describes the elements you can define within the tcp-initiator element.

Table D–46 tcp-initiator Subelements

Element	Required/ Optional	Description

<local-address>	Optional	Specifies the local address (IP or DNS name) that the TCP/IP socket opened by the connection initiator will be bound to. For example, the following will instruct the connection initiator to bind the TCP/IP socket to the IP address 192.168.0.1:
		`<local-address>`
		`  <address>192.168.0.1</address>`
		`</local-address>`

<remote-addresses>	Required	Contains the <socket-address> of one or more TCP/IP connection acceptors. The TCP/IP connection initiator uses this information to establish a TCP/IP connection with a remote cluster. The TCP/IP connection initiator will attempt to connect to the addresses in a random order, until either the list is exhausted or a TCP/IP connection is established.For example, the following will instruct the connection initiator to attempt to connect to 192.168.0.2:9099 and 192.168.0.3:9099 in a random order:
		`<remote-addresses>`
		`  <socket-address>`
		`  <address>192.168.0.2</address>`
		`  <port>9099</port>`
		`  </socket-address>`
		`  <socket-address>`
		`  <address>192.168.0.3</address>`
		`  <port>9099</port>`
		`  </socket-address>`
		`</remote-addresses>`
		Alternatively, the set of remote addresses may be specified using a <address-provider> element instead of the list of <socket-address> elements. This approach may be used to implement custom load balancing algorithms and/or dynamic discovery of TCP/IP connection acceptors.

<keep-alive-enabled>	Optional	Indicates whether keep alive (SO_KEEPALIVE) is enabled on a TCP/IP socket. Valid values are true and false. Keep alive is enabled by default.
<tcp-delay-enabled>	Optional	Indicates whether TCP delay (Nagle's algorithm) is enabled on a TCP/IP socket. Valid values are true and false. TCP delay is disabled by default.

Table D–46 (Cont.) tcp-initiator Subelements

Element	Required/ Optional	Description
`<receive-buffer-size>`	Optional	Configures the size of the underlying TCP/IP socket network receive buffer.Increasing the receive buffer size can increase the performance of network I/O for high-volume connections, while decreasing it can help reduce the backlog of incoming data.The value of this element must be in the following format: `[\d]+[[.][\d]+]?[K|k|M|m|G|g]?[B|b]?` where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied: ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) If the value does not contain a factor, a factor of one is assumed. Default value is O/S dependent.

Table D–46 (Cont.) tcp-initiator Subelements

Element	Required/ Optional	Description
<send-buffer-size>	Optional	Configures the size of the underlying TCP/IP socket network send buffer.The value of this element must be in the following format: [\d]+[[.][\d]+]?[K|k|M|m|G|g]?[B|b]? where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied: ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) If the value does not contain a factor, a factor of one is assumed. Default value is O/S dependent.
<connect-timeout>	Optional	Specifies the maximum amount of time to wait while establishing a connection with a connection acceptor.The value of this element must be in the following format: [\d]+[[.][\d]+]?[MS|ms|S|s|M|m|H|h|D|d]? where the first non-digits (from left to right) indicate the unit of time duration: ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) If the value does not contain a unit, a unit of milliseconds is assumed. Default value is an infinite timeout.
<linger-timeout>	Optional	Enables SO_LINGER on a TCP/IP socket with the specified linger time. The value of this element must be in the following format: [\d]+[[.][\d]+]?[MS|ms|S|s|M|m|H|h|D|d]? where the first non-digits (from left to right) indicate the unit of time duration: ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) If the value does not contain a unit, a unit of milliseconds is assumed. Linger is disabled by default.

version-persistent-scheme

Used in: `versioned-backing-map-scheme`.

Description

The `version-persistent-scheme` defines a cache for storing object versioning information in a clustered cache. Specifying a size limit on the specified scheme's backing-map allows control over how many version identifiers are tracked.

Elements

Table D–47 describes the elements you can define within the version-`persistent-scheme` element.

Table D–47 persistent-scheme Subelements

Element	Required/ Optional	Description
`<cache-name-suffix>`	Optional	Specifies the name modifier that is used to create a cache of version objects associated with a given cache. The value of this element is appended to the base cache name. Legal value is a string. Default value is `-persist`. For example, if the base case is named `Sessions` and this name modifier is set to `-persist`, the associated version cache will be named `Sessions-persist`.
`<replicated-scheme>` or `<distributed-scheme>`	Required	Specifies the scheme for the cache used to maintain the versioning information. Legal values are: ■ `replicated-scheme` ■ `distributed-scheme`

version-transient-scheme

Used in: versioned-near-scheme, versioned-backing-map-scheme.

Description

The version-transient-scheme defines a cache for storing object versioning information for use in versioned near-caches. Specifying a size limit on the specified scheme's backing-map allows control over how many version identifiers are tracked.

Elements

The following table describes the elements you can define within the version-transient-scheme element.

Table D–48 transient-scheme Subelements

Element	Required/ Optional	Description
<cache-name-suffix>	Optional	Specifies the name modifier that is used to create a cache of version objects associated with a given cache. The value of this element is appended to the base cache name. Legal value is a string. Default value is "-version". For example, if the base case is named Sessions and this name modifier is set to -version, the associated version cache will be named Sessions-version.
<replicated-scheme> or <distributed-scheme>	Required	Specifies the scheme for the cache used to maintain the versioning information. Legal values are: ■ replicated-scheme ■ distributed-scheme

versioned-backing-map-scheme

Used in: caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme.

Description

The versioned-backing-map-scheme is an extension of a read-write-backing-map-scheme, defining a size limited cache of a persistent store. It uses object versioning to determine what updates need to be written to the persistent store. See "Versioning" for more information.

Implementation

The versioned-backing-map-scheme scheme is implemented by the com.tangosol.net.cache.VersionedBackingMap class.

Cache of an External Store

As with the read-write-backing-map-scheme, a versioned backing map maintains a cache backed by an external persistent cache store (see <cachestore-scheme> subelement), cache misses will read-through to the back-end store to retrieve the data. Cache stores may also support updates to the back-end data store.

Refresh-Ahead and Write-Behind Caching

As with the read-write-backing-map-scheme both the refresh-ahead (see <refresh-ahead> subelement) and write-behind (see <write-behind> subelement) caching optimizations are supported. See "Read-Through, Write-Through, Write-Behind Caching and Refresh-Ahead" for more details.

Versioning

For entries whose values implement the com.tangosol.util.Versionable interface, the versioned backing map will use the version identifier to determine if an update must be written to the persistent store. The primary benefit of this feature is that in the event of cluster node failover, the backup node can determine if the most recent version of an entry has already been written to the persistent store, and if so it can avoid an extraneous write.

Elements

Table D–49 describes the elements you can define within the versioned-backing-map-scheme element.

Table D–49 versioned-backing-map-scheme Subelement

Element	Required/ Optional	Description
<scheme-name>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<scheme-ref>	Optional	Specifies the name of another scheme to inherit from. See "Scheme Inheritance" on page D-19 for more information.
<class-name>	Optional	Specifies a custom implementation of the versioned backing map. Any custom implementation must extend the com.tangosol.net.cache.VersionedBackingMap class and declare the exact same set of public constructors.

Table D–49 (Cont.) versioned-backing-map-scheme Subelement

Element	Required/ Optional	Description
<init-params>	Optional	Specifies initialization parameters, for use in custom versioned backing map implementations which implement the com.tangosol.run.xml.XmlConfigurable interface.
<listener>	Optional	Specifies an implementation of a com.tangosol.util.MapListener which will be notified of events occurring on the cache.
<cachestore-scheme>	Optional	Specifies the store to cache. If unspecified the cached data will only reside within the (see <internal-cache-scheme> subelement), and only reflect operations performed on the cache itself.
<internal-cache-scheme>	Required	Specifies a cache-scheme which will be used to cache entries. Legal values are: ■ local-scheme ■ external-scheme ■ paged-external-scheme ■ overflow-scheme ■ class-scheme
<miss-cache-scheme>	Optional	Specifies a cache-scheme for maintaining information on cache misses. The miss-cache is used track keys which were not found in the cache store. The knowledge that a key is not in the cache store allows some operations to perform faster, as they can avoid querying the potentially slow cache store. A size-limited scheme may be used to control how many misses are cached. If unspecified no cache-miss data will be maintained. Legal values are: ■ local-scheme
<read-only>	Optional	Specifies if the cache is read only. If true the cache will load data from cachestore for read operations and will not perform any writing to the cachestore when the cache is updated. Legal values are true or false. Default value is false.
<write-delay>	Optional	Specifies the time interval for a write-behind queue to defer asynchronous writes to the cachestore by.The value of this element must be in the following format: [\d]+[[.][\d]+]?[MS|ms|S|s|M|m|H|h|D|d]? where the first non-digits (from left to right) indicate the unit of time duration: ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) If the value does not contain a unit, a unit of seconds is assumed. If zero, synchronous writes to the cachestore (without queuing) will take place, otherwise the writes will be asynchronous and deferred by the number of seconds after the last update to the value in the cache. Default is zero.

Table D–49 (Cont.) versioned-backing-map-scheme Subelement

Element	Required/ Optional	Description
<write-batch-factor>	Optional	The write-batch-factor element is used to calculate the "soft-ripe" time for write-behind queue entries. A queue entry is considered to be "ripe" for a write operation if it has been in the write-behind queue for no less than the write-delay interval. The "soft-ripe" time is the point in time before the actual "ripe" time after which an entry will be included in a batched asynchronous write operation to the CacheStore (along with all other "ripe" and "soft-ripe" entries). This element is only applicable if asynchronous writes are enabled (that is, the value of the write-delay element is greater than zero) and the CacheStore implements the storeAll() method. The value of the element is expressed as a percentage of the write-delay interval. For example, if the value is zero, only "ripe" entries from the write-behind queue will be batched. On the other hand, if the value is 1.0, all currently queued entries will be batched and the value of the write-delay element will be effectively ignored. Legal values are nonnegative doubles less than or equal to 1.0. Default is zero.
<write-requeue-threshold>	Optional	Specifies the maximum size of the write-behind queue for which failed cachestore write operations are requeued. The purpose of this setting is to prevent flooding of the write-behind queue with failed cachestore operations. This can happened in situations where a large number of successive write operations fail. If zero, write-behind requeuing is disabled. Legal values are positive integers or zero. Default is zero.
<refresh-ahead-factor>	Optional	The refresh-ahead-factor element is used to calculate the "soft-expiration" time for cache entries. Soft-expiration is the point in time before the actual expiration after which any access request for an entry will schedule an asynchronous load request for the entry. This attribute is only applicable if the internal cache (see <internal-cache-scheme> subelement) is a local-scheme, configured with the <location> subelement. The value is expressed as a percentage of the internal LocalCache expiration interval. If zero, refresh-ahead scheduling will be disabled. If 1.0, then any get operation will immediately trigger an asynchronous reload. Legal values are nonnegative doubles less than or equal to 1.0. Default value is zero.
<rollback-cachestore-failures>	Optional	Specifies whether exceptions caught during synchronous cachestore operations are rethrown to the calling thread (possibly over the network to a remote member). If the value of this element is false, an exception caught during a synchronous cachestore operation is logged locally and the internal cache is updated. If the value is true, the exception is rethrown to the calling thread and the internal cache is not changed. If the operation was called within a transactional context, this would have the effect of rolling back the current transaction. Legal values are true or false. Default value is false.
<version-persistent-scheme>	Optional	Specifies a cache-scheme for tracking the version identifier for entries in the persistent cachestore (see cachestore-scheme).
<version-transient-scheme>	Optional	Specifies a cache-scheme for tracking the version identifier for entries in the transient internal cache (see <internal-cache-scheme> subelement).
<manage-transient>	Optional	Specifies if the backing map is responsible for keeping the transient version cache up to date. If disabled the backing map manages the transient version cache only for operations for which no other party is aware (such as entry expiry). This is used when there is already a transient version cache of the same name being maintained at a higher level, for instance within a versioned-near-scheme. Legal values are true or false. Default value is false.

versioned-near-scheme

Used in: caching-schemes.

> **Note:** As of Coherence release 2.3, it is suggested that a near-scheme be used instead of versioned-near-scheme. Legacy Coherence applications use versioned-near-scheme to ensure Coherence through object versioning. As of Coherence 2.3 the near-scheme includes a better alternative, in the form of reliable and efficient front cache invalidation.

Description

As with the near-scheme, the versioned-near-scheme defines a two tier cache consisting of a small and fast front-end, and higher-capacity but slower back-end cache. The front-end (see <front-end> subelement) and back-end (see <back-end> subelement) are expressed as normal cache-schemes. A typical deployment might use a local-scheme for the front-end, and a distributed-scheme for the back-end. See Appendix B, "Types of Caches in Coherence" for a more detailed description of versioned near caches.

Implementation

The versioned near scheme is implemented by the com.tangosol.net.cache.VersionedNearCache class.

Versioning

Object versioning is used to ensure coherence between the front and back tiers. See the <version-transient-scheme> subelement for more information

Elements

Table D–50 describes the elements you can define within the near-scheme element.

Table D–50 near-scheme Subelements

Element	Required/ Optional	Description
<scheme-name>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<scheme-ref>	Optional	Specifies the name of another scheme to inherit from. See "Scheme Inheritance" on page D-19 for more information.
<class-name>	Optional	Specifies a custom implementation of the versioned near cache. The specified class must extend the com.tangosol.net.cache.VersionedNearCache class and declare the exact same set of public constructors.
<init-params>	Optional	Specifies initialization parameters, for use in custom versioned near cache implementations which implement the com.tangosol.run.xml.XmlConfigurable interface.
<listener>	Optional	Specifies an implementation of a com.tangosol.util.MapListener which will be notified of events occurring on the cache.

Table D–50 (Cont.) near-scheme Subelements

Element	Required/ Optional	Description
<front-scheme>	Required	Specifies the cache-scheme to use in creating the front-tier cache.Legal values are: ■ local-scheme ■ external-scheme ■ paged-external-scheme ■ class-scheme For example: <front-scheme> <local-scheme> <scheme-ref>default-eviction</scheme-ref> </local-scheme> </front-scheme> or <front-scheme> <class-scheme> <class-name>com.tangosol.util.SafeHashMap</class-name> <init-params></init-params> </class-scheme> </front-scheme>
<back-scheme>	Required	Specifies the cache-scheme to use in creating the back-tier cache.Legal values are: ■ distributed-scheme ■ replicated-scheme ■ optimistic-scheme ■ local-scheme ■ external-scheme ■ paged-external-scheme ■ class-scheme For Example: <back-scheme> <distributed-scheme> <scheme-ref>default-distributed</scheme-ref> </distributed-scheme> </back-scheme>
<version-transient-scheme>	Optional	Specifies a scheme for versioning cache entries, which ensures coherence between the front and back tiers.
<autostart>	Optional	The autostart element is intended to be used by cache servers (that is, com.tangosol.net.DefaultCacheServer). It specifies whether the cache services associated with this cache scheme should be automatically started at a cluster node. Legal values are true or false. Default value is false.

E

Cache Configuration Parameter Macros

The Cache Configuration Deployment Descriptor (`coherence-cache-config.xml`) supports parameter *macros* to minimize custom coding and enable specification of commonly used attributes when configuring class constructor parameters. The macros should be entered enclosed in curly braces as shown below, without any quotes or spaces.

Table E–1 describes the parameter macros that may be specified:

Table E–1 Parameter Macros for Cache Configuration

<param-type>	<param-value>	Description

`java.lang.String`	`{cache-name}`	Used to pass the current cache name as a constructor parameter For example:
		`<class-name>com.mycompany.cache.CustomCacheLoader</class-name>`
		`<init-params>`
		`  <init-param>`
		`  <param-type>`**`java.lang.String`**`</param-type>`
		`  <param-value>`**`{cache-name}`**`</param-value>`
		`  </init-param>`
		`</init-params>`

`java.lang.ClassLoader`	`{class-loader}`	Used to pass the current classloader as a constructor parameter. For example:
		`<class-name>com.mycompany.cache.CustomCacheLoader</class-name>`
		`<init-params>`
		`  <init-param>`
		`  <param-type>`**`java.lang.ClassLoader`**`</param-type>`
		`  <param-value>`**`{class-loader}`**`</param-value>`
		`  </init-param>`
		`</init-params>`

Table E–1 (Cont.) Parameter Macros for Cache Configuration

<param-type>	<param-value>	Description

com.tangosol.net.BackingMapManagerContext	{manager-context}	Used to pass the current BackingMapManagerContext object as a constructor parameter. For example:
		`<class-name>com.mycompany.cache.CustomCacheLoader</class-name>`
		`<init-params>`
		`  <init-param>`
		`  <param-type>`**`com.tangosol.net.BackingMapManagerContext`**
		`  </param-type>`
		`  <param-value>`**`{manager-context}`**`</param-value>`
		`  </init-param>`
		`</init-params>`

{scheme-ref}	local-scheme	Instantiates an object defined by the <class-scheme>, <local-scheme> or <file-scheme> with the specified <scheme-name> value and uses it as a constructor parameter. For example:
		`<class-scheme>`
		`  `**`<scheme-name>dbconnection</scheme-name>`**
		`  <class-name>com.mycompany.dbConnection</class-name>`
		`  <init-params>`
		`  <init-param>`
		`  <param-name>driver</param-name>`
		`  <param-type>String</param-type>`
		`  <param-value>org.gjt.mm.mysql.Driver</param-value>`
		`  </init-param>`
		`  <init-param>`
		`  <param-name>url</param-name>`
		`  <param-type>String</param-type>`
		`  <param-value>jdbc:mysql://dbserver:3306/companydb`
		`  </param-value>`
		`  </init-param>`
		`  <init-param>`
		`  <param-name>user</param-name>`
		`  <param-type>String</param-type>`
		`  <param-value>default</param-value>`
		`  </init-param>`
		`  <init-param>`
		`  <param-name>password</param-name>`
		`  <param-type>String</param-type>`
		`  <param-value>default</param-value>`
		`  </init-param>`
		`  </init-params>`
		`</class-scheme>`
		`...`
		`<class-name>com.mycompany.cache.CustomCacheLoader</class-name>`
		`  <init-params>`
		`  <init-param>`
		`  <param-type>`**`{scheme-ref}`**`</param-type>`
		`  <param-value>`**`dbconnection`**`</param-value>`
		`  </init-param>`
		`  </init-params>`

Table E–1 (Cont.) Parameter Macros for Cache Configuration

<param-type>	<param-value>	Description

{cache-ref}	cache name	Used to obtain a NamedCache reference for the specified cache name. Consider the following configuration example:
		
		`<cache-config>`
		`  <caching-scheme-mapping>`
		`  <cache-mapping>`
		`  <cache-name>boston-*</cache-name>`
		`  <scheme-name>wrapper</scheme-name>`
		`  <init-params>`
		`  <init-param>`
		`  <param-name>delegate-cache-name</param-name>`
		`  <param-value>london-*</param-value>`
		`  </init-param>`
		`  </init-params>`
		`  </cache-mapping>`
		`  <cache-mapping>`
		`  <cache-name>london-*</cache-name>`
		`  <scheme-name>partitioned</scheme-name>`
		`  </cache-mapping>`
		`  </caching-scheme-mapping>`
		
		`  <caching-schemes>`
		`  <class-scheme>`
		`  <scheme-name>wrapper</scheme-name>`
		`  <class-name>com.tangosol.net.cache.WrapperNamedCache`
		`  </class-name>`
		`  <init-params>`
		`  <init-param>`
		`  <param-type>cache-ref</param-type>`
		`  <param-value>delegate-cache-name</param-value>`
		`  </init-param>`
		`  <init-param>`
		`  <param-type>string</param-type>`
		`  <param-value>cache-name</param-value>`
		`  </init-param>`
		`  </init-params>`
		`  </class-scheme>`
		
		`  <distributed-scheme>`
		`  <scheme-name>partitioned</scheme-name>`
		`  <service-name>partitioned</service-name>`
		`  <backing-map-scheme>`
		`  <local-scheme>`
		`  <unit-calculator>BINARY</unit-calculator>`
		`  </local-scheme>`
		`  </backing-map-scheme>`
		`  <autostart>true</autostart>`
		`  </distributed-scheme>`
		`  </caching-schemes>`
		`</cache-config>`
		
		The CacheFactory.getCache("london-test") call would result in a standard partitioned cache reference. Conversely, the CacheFactory.getCache("boston-test") call would resolve the value of the delegate-cache-name parameter to london-test and would construct an instance of the WrapperNamedCache delegating to the NamedCache returned by the CacheFactory.getCache("london-test") call.

F

Sample Cache Configurations

This section provides a series of simple cache scheme configurations. The samples build upon one another and will often use a `scheme-ref` element to reuse other samples as nested schemes. See "Scheme Inheritance" on page D-19 for an example of `<scheme-ref>`.

Cache schemes are specified in the `caching-schemes` element of the cache configuration descriptor `coherence-cache-config.xml` which is described in Appendix D, "Cache Configuration Elements". These samples only specify a minimum number of settings, follow the embedded links to the scheme's documentation to see the full set of options.

This section describes configurations for the following caching scenarios:

- Local Caches (accessible from a single JVM)
  - In-memory Cache
  - NIO In-memory Cache
  - Size Limited In-memory Cache
  - In-memory Cache with Expiring Entries
  - Cache on Disk
  - Size Limited Cache on Disk
  - Persistent Cache on Disk
  - In-memory Cache with Disk Based Overflow
  - Cache of a Database
- Clustered Caches (accessible from multiple JVMs)
  - Replicated Cache
  - Replicated Cache with Overflow
  - Partitioned Cache
  - Partitioned Cache with Overflow
  - Partitioned Cache of a Database
  - Partitioned Cache with a Serializer
  - Local Cache of a Partitioned Cache (Near cache)

Local Caches (accessible from a single JVM)

This section defines a series of local cache schemes. In this context "local" means that the cache is only directly accessible by a single JVM. Later in this document local caches will be used as building blocks for clustered caches. See "Clustered Caches (accessible from multiple JVMs)" on page F-5.

In-memory Cache

Example F–1 uses a `local-scheme` to define an in-memory cache. The cache will store as much as the JVM heap will allow.

Example F–1 Configuration for a Local, In-memory Cache

```
<local-scheme>
   <scheme-name>SampleMemoryScheme</scheme-name>
</local-scheme>
```

NIO In-memory Cache

Example F–2 uses an `external-scheme` to define an in-memory cache using an `nio-memory-manager`. The advantage of an NIO memory based cache is that it allows for large in-memory cache storage while not negatively impacting the JVM's GC times. The size of the cache is limited by the maximum size of the NIO memory region. See the `<maximum-size>` subelement of `nio-memory-manager`.

Example F–2 Configuration for a NIO In-memory Cache

```
<external-scheme>
  <scheme-name>SampleNioMemoryScheme</scheme-name>
  <nio-memory-manager/>
</external-scheme>
```

Size Limited In-memory Cache

Adding a `<high-units>` sub element to `<local-scheme>` limits the size of the cache. Here the cache is size limited to one thousand entries. When the limit is exceeded, the scheme's `<eviction-policy>` will determine which elements to evict from the cache.

Example F–3 Configuration for a Size Limited, In-memory, Local Cache

```
<local-scheme>
   <scheme-name>SampleMemoryLimitedScheme</scheme-name>
   <high-units>1000</high-units>
</local-scheme>
```

In-memory Cache with Expiring Entries

Adding an `<expiry-delay>` subelement to `<local-scheme>` will cause cache entries to automatically expire if they are not updated for a given time interval. When expired the cache will invalidate the entry, and remove it from the cache.

Example F–4 Configuration for an In-memory Cache with Expiring Entries

```
<local-scheme>
   <scheme-name>SampleMemoryExpirationScheme</scheme-name>
   <expiry-delay>5m</expiry-delay>
</local-scheme>
```

Cache on Disk

Example F–5 uses an `external-scheme` to define an on disk cache. The cache will store as much as the file system will allow.

> **Note:** This example uses the `lh-file-manager` for its on disk storage implementation. See "external-scheme" on page D-32 for additional external storage options.

Example F–5 Configuration to Define a Cache on Disk

```
<external-scheme>
  <scheme-name>SampleDiskScheme</scheme-name>
  <lh-file-manager/>
</external-scheme>
```

Size Limited Cache on Disk

Adding a `<high-units>` sub- element to `external-scheme` limits the size of the cache. The cache is size limited to one million entries. When the limit is exceeded, LRU eviction is used determine which elements to evict from the cache. Refer to "paged-external-scheme" on page D-65 for an alternate size limited external caching approach.

Example F–6 Configuration for a Size Limited Cache on Disk

```
<external-scheme>
  <scheme-name>SampleDiskLimitedScheme</scheme-name>
  <lh-file-manager/>
  <high-units>1000000</high-units>
</external-scheme>
```

Persistent Cache on Disk

Example F–7 uses an `external-scheme` to implement a cache suitable for use as long-term storage for a single JVM.

External caches are generally used for temporary storage of large data sets, and are automatically deleted on JVM shutdown. An external-cache can be used for long term storage (see "Persistence (long-term storage)" on page D-33) in non-clustered caches when using either the `lh-file-manager` or `bdb-store-manager` storage managers. For clustered persistence see the "Partitioned Cache of a Database" on page F-6 sample.

The `{cache-name}` macro is used to specify the name of the file the data will be stored in. See Appendix E, "Cache Configuration Parameter Macros" for more information on this macro.

Example F–7 Configuration for Persistent cache on disk

```
<external-scheme>
  <scheme-name>SampleDiskPersistentScheme</scheme-name>
  <lh-file-manager>
    <directory>/my/storage/directory</directory>
    <file-name>{cache-name}.store</file-name>
  </lh-file-manager>
</external-scheme>
```

Example F–8 illustrates using Berkeley DB rather then LH.

Example F–8 Configuration for Persistent cache on disk with Berkeley DB

```
<external-scheme>
  <scheme-name>SampleDiskPersistentScheme</scheme-name>
  <bdb-store-manager>
    <directory>/my/storage/directory</directory>
    <store-name>{cache-name}.store</store-name>
  </bdb-store-manager>
</external-scheme>
```

In-memory Cache with Disk Based Overflow

Example F–9 uses an `overflow-scheme` to define a size limited in-memory cache, when the in-memory (`<front-scheme>`) size limit is reached, a portion of the cache contents will be moved to the on disk (`<back-scheme>`). The front-scheme's `<eviction-policy>` will determine which elements to move from the front to the back.

Note that this example reuses the examples in "Size Limited Cache on Disk" and "Cache on Disk" on page F-3. to implement the front and back of the cache.

Example F–9 Configuration for In-memory Cache with Disk Based Overflow

```
<overflow-scheme>
  <scheme-name>SampleOverflowScheme</scheme-name>
  <front-scheme>
    <local-scheme>
      <scheme-ref>SampleMemoryLimitedScheme</scheme-ref>
    </local-scheme>
  </front-scheme>
  <back-scheme>
    <external-scheme>
      <scheme-ref>SampleDiskScheme</scheme-ref>
    </external-scheme>
  </back-scheme>
</overflow-scheme>
```

Cache of a Database

Example F–10 uses a `read-write-backing-map-scheme` to define a cache of a database. This scheme maintains local cache of a portion of the database contents. Cache misses will read-through to the database, and cache writes will be written back to the database.

The `cachestore-scheme` element is configured with a custom class implementing either the `com.tangosol.net.cache.CacheLoader` or `com.tangosol.net.cache.CacheStore` interface. This class is responsible for all operations against the database, such as reading and writing cache entries. See Appendix G, "Sample CacheStores" implementations for examples of writing a cache store.

The `{cache-name}` macro is used to inform the cache store implementation of the name of the cache it will back.See Appendix E, "Cache Configuration Parameter Macros" for more information on this macro.

Example F–10 Configuration for the Cache of a Database

```
<read-write-backing-map-scheme>
  <scheme-name>SampleDatabaseScheme</scheme-name>
  <internal-cache-scheme>
    <local-scheme>
      <scheme-ref>SampleMemoryScheme</scheme-ref>
```

```
    </local-scheme>
  </internal-cache-scheme>
  <cachestore-scheme>
    <class-scheme>
      <class-name>com.tangosol.examples.coherence.DBCacheStore</class-name>
      <init-params>
        <init-param>
          <param-type>java.lang.String</param-type>
          <param-value>{cache-name}</param-value>
        </init-param>
      </init-params>
    </class-scheme>
  </cachestore-scheme>
</read-write-backing-map-scheme>
```

Clustered Caches (accessible from multiple JVMs)

This section defines a series of clustered cache examples. Clustered caches are accessible from multiple JVMs (any cluster node running the same cache service). The internal cache storage (backing-map) on each cluster node is defined using local caches (see "Local Caches (accessible from a single JVM)" on page F-2). The cache service provides the capability to access local caches from other cluster nodes.

Replicated Cache

Example F–11 uses the `replicated-scheme` element to define a clustered cache in which a copy of each cache entry will be stored on all cluster nodes.

The sample in "In-memory Cache" on page F-2 is used to define the cache storage on each cluster node. The size of the cache is only limited by the cluster node with the smallest JVM heap.

Example F–11 Configuration for a Replicated Cache

```
<replicated-scheme>
  <scheme-name>SampleReplicatedScheme</scheme-name>
  <backing-map-scheme>
    <local-scheme>
      <scheme-ref>SampleMemoryScheme</scheme-ref>
    </local-scheme>
  </backing-map-scheme>
</replicated-scheme>
```

Replicated Cache with Overflow

The `backing-map-scheme` element could just as easily specify any of the other local cache samples. For instance, if it had used the "In-memory Cache with Disk Based Overflow" on page F-4, each cluster node would have a local overflow cache allowing for much greater storage capacity.

Example F–12 Configuration for a Replicated Cache with Overflow

```
<replicated-scheme>
  <scheme-name>SampleReplicatedOverflowScheme</scheme-name>
  <backing-map-scheme>
    <overflow-scheme>
      <scheme-ref>SampleOverflowScheme</scheme-ref>
    </overflow-scheme>
  </backing-map-scheme>
```

```
</replicated-scheme>
```

Partitioned Cache

Example F–13 uses the `distributed-scheme` to define a clustered cache in which cache storage is partitioned across all cluster nodes.

The "In-memory Cache" on page F-2 is used to define the cache storage on each cluster node. The total storage capacity of the cache is the sum of all storage enabled cluster nodes running the partitioned cache service. See the `<local-storage>` subelement of "distributed-scheme" on page D-26.

Example F–13 Configuration for a Partitioned Cache

```
<distributed-scheme>
  <scheme-name>SamplePartitionedScheme</scheme-name>
  <backing-map-scheme>
    <local-scheme>
      <scheme-ref>SampleMemoryScheme</scheme-ref>
    </local-scheme>
  </backing-map-scheme>
</distributed-scheme>
```

Partitioned Cache with Overflow

The `backing-map-scheme` element could just as easily specify any of the other local cache samples. For instance if it had used the "In-memory Cache with Disk Based Overflow" on page F-4, each storage-enabled cluster node would have a local overflow cache allowing for much greater storage capacity. Note that the cache's backup storage also uses the same overflow scheme which allows for backup data to be overflowed to disk.

Example F–14 Configuration for a Partitioned Cache with Overflow

```
<distributed-scheme>
  <scheme-name>SamplePartitionedOverflowScheme</scheme-name>
  <backing-map-scheme>
    <overflow-scheme>
      <scheme-ref>SampleOverflowScheme</scheme-ref>
    </overflow-scheme>
  </backing-map-scheme>
  <backup-storage>
    <type>scheme</type>
    <scheme-name>SampleOverflowScheme</scheme-name>
  </backup-storage>
</distributed-scheme>
```

Partitioned Cache of a Database

Switching the `backing-map-scheme` element to use a `read-write-backing-map-scheme` allows the cache to load and store entries against an external source such as a database.

Example F–15 reuses the "Cache of a Database" on page F-4 to define the database access.

Example F–15 Configuration for a Partitioned Cache of a Database

```
<distributed-scheme>
  <scheme-name>SamplePartitionedDatabaseScheme</scheme-name>
```

```
  <backing-map-scheme>
    <read-write-backing-map-scheme>
      <scheme-ref>SampleDatabaseScheme</scheme-ref>
    </read-write-backing-map-scheme>
  </backing-map-scheme>
</distributed-scheme>
```

Partitioned Cache with a Serializer

Example F–16 uses the serializer element in `distributed-scheme` to define a serializer that will be used to serialize and deserialize user types. In this case, the partitioned cache will use POF (`ConfigurablePofContext`) as its serialization format. Note that if you use POF and your application uses any custom user type classes, then you must also define a custom POF configuration for them. See Appendix J, "POF User Type Configuration Elements" for more information on POF elements.

Example F–16 Configuration for a Partitioned Cache with a Serializer

```
<distributed-scheme>
  <scheme-name>SamplePartitionedPofScheme</scheme-name>
  <service-name>PartitionedPofCache</service-name>
  <serializer>
    <class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>
  </serializer>
  <backing-map-scheme>
    <local-scheme/>
  </backing-map-scheme>
  <autostart>true</autostart>
</distributed-scheme>
```

Local Cache of a Partitioned Cache (Near cache)

Example F–17 uses the `near-scheme` to define a local in-memory cache of a subset of a partitioned cache. The result is that any cluster node accessing the partitioned cache will maintain a local copy of the elements it frequently accesses. This offers read performance close to the `replicated-scheme`-based caches, while offering the high scalability of a `distributed-scheme`-based cache.

The "Size Limited In-memory Cache" on page F-2 sample is reused to define the "near" (`<front-scheme>`) cache, while the "Partitioned Cache" on page F-6 sample is reused to define the `near-scheme`.

Note that the size limited configuration of the front-scheme specifies the limit on how much of the back-scheme cache is locally cached.

Example F–17 Configuration for a Local Cache of a Partitioned Cache

```
<near-scheme>
  <scheme-name>SampleNearScheme</scheme-name>
  <front-scheme>
    <local-scheme>
       <scheme-ref>SampleLimitedMemoryScheme</scheme-ref>
    </local-scheme>
  </front-scheme>
  <back-scheme>
    <distributed-scheme>
      <scheme-ref>SamplePartitionedScheme</scheme-ref>
    </distributed-scheme>
  </back-scheme>
```

```
    </near-scheme>
```

G

Sample CacheStores

Cache stores are used by caches to read and write cache entries to external stores such as a database. The examples on this page illustrate different ways in which you can interact with a cache store.

> **Note:** Save processing effort by bulk loading the cache. The following examples use the `put` method to write values to the cache store. Often, performing bulk loads with the `putAll` method will result in a savings in processing effort and network traffic. For more information on bulk loading, see Chapter 13, "Pre-Loading the Cache."

Sample CacheStore

This section provides a very basic implementation of the `com.tangosol.net.cache.CacheStore` interface. The implementation in Example G–1 uses a single database connection by using JDBC, and does not use bulk operations. A complete implementation would use a connection pool, and, if write-behind is used, implement `CacheStore.storeAll()` for bulk JDBC inserts and updates. "Cache of a Database" on page F-4 provides an example of a database cache configuration.

Example G–1 Implementation of the CacheStore Interface

```
package com.tangosol.examples.coherence;


import com.tangosol.net.cache.CacheStore;
import com.tangosol.util.Base;

import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import java.util.Collection;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;


/**
* An example implementation of CacheStore
```

```
* interface.
*
* @author erm 2003.05.01
*/
public class DBCacheStore
        extends Base
        implements CacheStore
    {
    // ----- constructors ---------------------------------------------------
    /**
    * Constructs DBCacheStore for a given database table.
    *
    * @param sTableName the db table name
    */
    public DBCacheStore(String sTableName)
        {
        m_sTableName = sTableName;
        configureConnection();
        }

        /**
        * Set up the DB connection.
        */
        protected void configureConnection()
                {
                try
                        {
                        Class.forName("org.gjt.mm.mysql.Driver");
                        m_con = DriverManager.getConnection(DB_URL, DB_USERNAME,
DB_PASSWORD);
                        m_con.setAutoCommit(true);
                        }
                catch (Exception e)
                        {
                        throw ensureRuntimeException(e, "Connection failed");
                        }
                }


        // ---- accessors -------------------------------------------------------

    /**
    * Obtain the name of the table this CacheStore is persisting to.
    *
    * @return the name of the table this CacheStore is persisting to
    */
    public String getTableName()
        {
        return m_sTableName;
        }

    /**
    * Obtain the connection being used to connect to the database.
    *
    * @return the connection used to connect to the database
    */
    public Connection getConnection()
        {
        return m_con;
        }
```

```
// ----- CacheStore Interface ------------------------------------------

/**
* Return the value associated with the specified key, or null if the
* key does not have an associated value in the underlying store.
*
* @param oKey  key whose associated value is to be returned
*
* @return the value associated with the specified key, or
*         <tt>null</tt> if no value is available for that key
*/
public Object load(Object oKey)
    {
    Object     oValue = null;
    Connection con    = getConnection();
    String     sSQL   = "SELECT id, value FROM " + getTableName()
                      + " WHERE id = ?";
    try
        {
        PreparedStatement stmt = con.prepareStatement(sSQL);

        stmt.setString(1, String.valueOf(oKey));

        ResultSet rslt = stmt.executeQuery();
        if (rslt.next())
            {
            oValue = rslt.getString(2);
            if (rslt.next())
                {
                throw new SQLException("Not a unique key: " + oKey);
                }
            }
        stmt.close();
        }
    catch (SQLException e)
        {
        throw ensureRuntimeException(e, "Load failed: key=" + oKey);
        }
    return oValue;
    }

/**
* Store the specified value under the specific key in the underlying
* store. This method is intended to support both key/value creation
* and value update for a specific key.
*
* @param oKey    key to store the value under
* @param oValue  value to be stored
*
* @throws UnsupportedOperationException  if this implementation or the
*         underlying store is read-only
*/
public void store(Object oKey, Object oValue)
    {
    Connection con     = getConnection();
    String     sTable  = getTableName();
    String     sSQL;
```

```
        // the following is very inefficient; it is recommended to use DB
        // specific functionality that is, REPLACE for MySQL or MERGE for Oracle
if (load(oKey) != null)
                {
                // key exists - update
         sSQL = "UPDATE " + sTable + " SET value = ? where id = ?";
                }
        else
                {
                // new key - insert
         sSQL = "INSERT INTO " + sTable + " (value, id) VALUES (?,?)";
                }
        try
                {
                PreparedStatement stmt = con.prepareStatement(sSQL);
                int i = 0;
                stmt.setString(++i, String.valueOf(oValue));
                stmt.setString(++i, String.valueOf(oKey));
                stmt.executeUpdate();
                stmt.close();
                }
        catch (SQLException e)
                {
                throw ensureRuntimeException(e, "Store failed: key=" + oKey);
                }
        }

    /**
    * Remove the specified key from the underlying store if present.
    *
    * @param oKey key whose mapping is to be removed from the map
    *
    * @throws UnsupportedOperationException  if this implementation or the
    *         underlying store is read-only
    */
    public void erase(Object oKey)
        {
        Connection con  = getConnection();
        String     sSQL = "DELETE FROM " + getTableName() + " WHERE id=?";
        try
            {
            PreparedStatement stmt = con.prepareStatement(sSQL);

            stmt.setString(1, String.valueOf(oKey));
            stmt.executeUpdate();
            stmt.close();
            }
        catch (SQLException e)
            {
            throw ensureRuntimeException(e, "Erase failed: key=" + oKey);
            }
        }

        /**
        * Remove the specified keys from the underlying store if present.
        *
        * @param colKeys  keys whose mappings are being removed from the cache
        *
        * @throws UnsupportedOperationException  if this implementation or the
        *         underlying store is read-only
```

```
    */
    public void eraseAll(Collection colKeys)
            {
            throw new UnsupportedOperationException();
            }

    /**
    * Return the values associated with each the specified keys in the
    * passed collection. If a key does not have an associated value in
    * the underlying store, then the return map will not have an entry
    * for that key.
    *
    * @param colKeys  a collection of keys to load
    *
    * @return a Map of keys to associated values for the specified keys
    */
    public Map loadAll(Collection colKeys)
            {
            throw new UnsupportedOperationException();
            }

    /**
    * Store the specified values under the specified keys in the underlying
    * store. This method is intended to support both key/value creation
    * and value update for the specified keys.
    *
    * @param mapEntries   a Map of any number of keys and values to store
    *
    * @throws UnsupportedOperationException  if this implementation or the
    *         underlying store is read-only
    */
    public void storeAll(Map mapEntries)
            {
            throw new UnsupportedOperationException();
            }

/**
* Iterate all keys in the underlying store.
*
* @return a read-only iterator of the keys in the underlying store
*/
public Iterator keys()
    {
    Connection con  = getConnection();
    String     sSQL = "SELECT id FROM " + getTableName();
    List       list = new LinkedList();

    try
        {
        PreparedStatement stmt = con.prepareStatement(sSQL);
        ResultSet         rslt = stmt.executeQuery();
        while (rslt.next())
            {
            Object oKey = rslt.getString(1);
            list.add(oKey);
            }
        stmt.close();
        }
    catch (SQLException e)
        {
```

```
                throw ensureRuntimeException(e, "Iterator failed");
                }

            return list.iterator();
            }


    // ----- data members ---------------------------------------------------

    /**
    * The connection.
    */
    protected Connection m_con;

    /**
    * The db table name.
    */
    protected String m_sTableName;

        /**
        * Driver class name.
        */
        private static final String DB_DRIVER   = "org.gjt.mm.mysql.Driver";

        /**
        * Connection URL.
        */
        private static final String DB_URL      =
"jdbc:mysql://localhost:3306/CacheStore";

        /**
        * User name.
        */
        private static final String DB_USERNAME = "root";

        /**
        * Password.
        */
        private static final String DB_PASSWORD = null;
    }
```

Sample Controllable CacheStore

This section illustrates the implementation of a controllable cache store. In this scenario, the application can control when updated values in the cache are written to the data store. The most common use case for this scenario is during the initial population of the cache from the data store at startup. At startup, there is no need to write values in the cache back to the data store. Any attempt to do so would be a waste of resources.

The `Main.java` file in Example G–2 illustrates two different approaches to interacting with a controllable cache store:

- Use a controllable cache (note that it must be on a different service) to enable or disable the cache store. This is illustrated by the `ControllableCacheStore1` class.
- Use the `CacheStoreAware` interface to indicate that objects added to the cache do not need to be stored. This is illustrated by the `ControllableCacheStore2` class.

Both `ControllableCacheStore1` and `ControllableCacheStore2` extend the `com.tangosol.net.cache.AbstractCacheStore` class. This helper class provides unoptimized implementations of the `storeAll` and `eraseAll` operations.

The `CacheStoreAware.java` file is an interface which can be used to indicate that an object added to the cache should not be stored in the database.

See "Cache of a Database" on page F-4 for a sample cache configurations.

Example G–2 provides a listing of the `Main.java` interface.

Example G–2 Main.java - Interacting with a Controllable CacheStore

```
import com.tangosol.net.CacheFactory;
import com.tangosol.net.NamedCache;
import com.tangosol.net.cache.AbstractCacheStore;
import com.tangosol.util.Base;

import java.io.Serializable;
import java.util.Date;

public class Main extends Base
    {

    /**
     * CacheStore implementation which is controlled by a control cache
     */
    public static class ControllableCacheStore1 extends AbstractCacheStore
        {
        public static final String CONTROL_CACHE = "cachestorecontrol";

        String m_sName;

        public static void enable(String sName)
            {
            CacheFactory.getCache(CONTROL_CACHE).put(sName, Boolean.TRUE);
            }

        public static void disable(String sName)
            {
            CacheFactory.getCache(CONTROL_CACHE).put(sName, Boolean.FALSE);
            }

        public void store(Object oKey, Object oValue)
            {
            Boolean isEnabled = (Boolean) CacheFactory.getCache(CONTROL_
CACHE).get(m_sName);
            if (isEnabled != null && isEnabled.booleanValue())
                {
                log("controllablecachestore1: enabled " + oKey + " = " + oValue);
                }
            else
                {
                log("controllablecachestore1: disabled " + oKey + " = " + oValue);
                }
            }

        public Object load(Object oKey)
            {
            log("controllablecachestore1: load:" + oKey);
            return new MyValue1(oKey);
```

```
            }

        public ControllableCacheStore1(String sName)
            {
            m_sName = sName;
            }

        }

    /**
     * CacheStore implementation which is controlled by values
     * implementing the CacheStoreAware interface
     */
    public static class ControllableCacheStore2 extends AbstractCacheStore
        {

        public void store(Object oKey, Object oValue)
            {
            boolean isEnabled = oValue instanceof CacheStoreAware ?
!((CacheStoreAware) oValue).isSkipStore() : true;
            if (isEnabled)
                {
                log("controllablecachestore2: enabled " + oKey + " = " + oValue);
                }
            else
                {
                log("controllablecachestore2: disabled " + oKey + " = " + oValue);
                }
            }

        public Object load(Object oKey)
            {
            log("controllablecachestore2: load:" + oKey);
            return new MyValue2(oKey);
            }

        }

    public static class MyValue1 implements Serializable
        {
        String m_sValue;

        public String getValue()
            {
            return m_sValue;
            }

        public String toString()
            {
            return "MyValue1[" + getValue() + "]";
            }

        public MyValue1(Object obj)
            {
            m_sValue = "value:" + obj;
            }
        }

    public static class MyValue2 extends MyValue1 implements CacheStoreAware
        {
```

```
    boolean m_isSkipStore = false;

    public boolean isSkipStore()
        {
        return m_isSkipStore;
        }

    public void skipStore()
        {
        m_isSkipStore = true;
        }

    public String toString()
        {
        return "MyValue2[" + getValue() + "]";
        }

    public MyValue2(Object obj)
        {
        super(obj);
        }

    }

public static void main(String[] args)
    {
    try
        {

        // example 1

        NamedCache cache1 = CacheFactory.getCache("cache1");

        // disable cachestore
        ControllableCacheStore1.disable("cache1");
        for(int i = 0; i < 5; i++)
            {
            cache1.put(new Integer(i), new MyValue1(new Date()));
            }

        // enable cachestore
        ControllableCacheStore1.enable("cache1");
        for(int i = 0; i < 5; i++)
            {
            cache1.put(new Integer(i), new MyValue1(new Date()));
            }

        // example 2

        NamedCache cache2 = CacheFactory.getCache("cache2");

        // add some values with cachestore disabled
        for(int i = 0; i < 5; i++)
            {
            MyValue2 value = new MyValue2(new Date());
            value.skipStore();
            cache2.put(new Integer(i), value);
            }

        // add some values with cachestore enabled
```

```
            for(int i = 0; i < 5; i++)
                {
                cache2.put(new Integer(i), new MyValue2(new Date()));
                }


            }
        catch(Throwable oops)
            {
            err(oops);
            }
        finally
            {
            CacheFactory.shutdown();
            }
        }

    }
```

Example G–3 provides a listing of the `CacheStoreAware.java` interface.

Example G–3 CacheStoreAware.java interface

```
public interface CacheStoreAware
    {
    public boolean isSkipStore();
    }
```

H

Operational Configuration Elements

This section describes the elements that control the operational and runtime settings used by Oracle Coherence. These settings are used to create, configure and maintain Coherence clustering, communication, and data management services. This section also describes the deployment descriptor files in which these elements can appear.

Operational Configuration Deployment Descriptors

The elements that control the operational and runtime settings to create and configure clustering, communication, and data management services can be specified in either of two deployment descriptors.

The `tangosol-coherence.xml` descriptor is where you specify the operational and runtime elements that control clustering, communication, and data management services. The optional `tangosol-coherence-override.xml` override file is where you specify only the subset of the operational descriptor which you want to adjust. See "Operational Override File (tangosol-coherence-override.xml)" on page H-2 for more information.

For information on configuring caches see Appendix D, "Cache Configuration Elements."

Document Location

When deploying Coherence, it is important to make sure that the `tangosol-coherence.xml` descriptor is present and situated in the application classpath (like with any other resource, Coherence will use the first one it finds in the classpath). By default (as Oracle ships the software) `tangosol-coherence.xml` is packaged into in the `coherence.jar`.

Document Root

The root element of the operational descriptor is <coherence>, this is where you may begin configuring your cluster and services.

Document Format

Coherence Operational Configuration deployment descriptor should begin with the following `DOCTYPE` declaration:

Example H–1 Operational Configuration Deployment Descriptor DOCTYPE Declaration

```
<!DOCTYPE coherence PUBLIC "-//Oracle, Inc.//DTD Oracle Coherence
3.4//EN""http://www.tangosol.com/dtd/coherence_3_3.dtd">
```

Note: When deploying Coherence into environments where the default character set is EBCDIC rather than ASCII, please make sure that this descriptor file is in ASCII format and is deployed into its runtime environment in the binary format.

Operational Override File (tangosol-coherence-override.xml)

Though it is acceptable to supply an alternate definition of the default `tangosol-coherence.xml` file, the preferred approach to operational configuration is to specify an override file. The override file contains only the subset of the operational descriptor which you want to adjust. The default name for the override file is `tangosol-coherence-override.xml`, and the first instance found in the classpath will be used. The format of the override file is the same as for the operational descriptor, except that all elements are optional, any missing element will simply be loaded from the operational descriptor.

Multiple levels of override files may also be configured, allowing for additional fine tuning between similar deployment environments such as staging and production. For example Coherence 3.2 and above use this feature to provide alternate configurations such as the logging verbosity based on the deployment type (evaluation, development, production). For more information on logging verbosity, see the `<severity-level>` subelement in "logging-config" on page H-19. See also the `tangosol-coherence-override-eval.xml`, `tangosol-coherence-override-dev.xml`, and `tangosol-coherence-override-prod.xml` files, within `coherence.jar` for the specific customizations.

Note: It is recommended that you supply an override file rather then a custom operational descriptor, thus specifying only the settings you want to adjust.

Command Line Override

Oracle Coherence provides a very powerful command line override feature which allows for any element defined in this descriptor to be overridden from the Java command line if it has a system-property attribute defined in the descriptor. This feature enables you to use the same operational descriptor (and override file) across all cluster nodes, and provide per-node customizations as system properties. See Appendix L, "Command Line Overrides" for more information on this feature.

Element Index

Table H–1 lists all non-terminal elements which may be used from within the operational configuration.

Table H–1 Non-Terminal Operational Configuration Elements

Element	Used in:
access-controller	security-config
authorized-hosts	cluster-config
burst-mode	packet-publisher
callback-handler	security-config
cluster-config	coherence
coherence	*root element*
configurable-cache-factory-config	coherence
filters	cluster-config
flow-control	packet-delivery
host-range	authorized-hosts
incoming-message-handler	cluster-config
init-param	init-params
init-params	access-controller, callback-handler, configurable-cache-factory-config, filters, services
license-config	coherence
logging-config	coherence
management-config	coherence
member-identity	cluster-config
multicast-listener	cluster-config
notification-queueing	packet-publisher
outgoing-message-handler	cluster-config
outstanding-packets	flow-control
packet-buffer	multicast-listener, packet-publisher, unicast-listener
packet-bundling	packet-delivery
packet-delivery	packet-publisher
packet-pool	incoming-message-handler, packet-publisher
packet-publisher	cluster-config
packet-size	packet-publisher
packet-speaker	cluster-config
pause-detection	flow-control
security-config	coherence
services	cluster-config
shutdown-listener	cluster-config

Table H–1 (Cont.) Non-Terminal Operational Configuration Elements

Element	Used in:
socket-address	well-known-addresses
tcp-ring-listener	cluster-config
traffic-jam	packet-publisher
unicast-listener	cluster-config
volume-threshold	packet-speaker
well-known-addresses	unicast-listener

access-controller

Used in: `security-config`.

Table H–2 describes the subelements you can define within the `access-controller` element.

Table H–2 access-controller Subelements

Element	Required/ Optional	Description
`<class-name>`	Required	Specifies the name of a Java class that implements `com.tangosol.net.security.AccessController` interface, which will be used by the security framework to check access rights for clustered resources and encrypt/decrypt node-to-node communications regarding those rights. See Chapter 7, "Security Framework" for more information. Default value is `com.tangosol.net.security.DefaultController`.

`<init-params>`	Optional	Contains one or more initialization parameter(s) for a class that implements the `AccessController` interface.For the default `AccessController` implementation the parameters are the paths to the key store file and permissions description file, specified as follows:
		`<init-params>`
		`  <init-param id="1">`
		`  <param-type>java.io.File</param-type>`
		`  <param-value system-property="tangosol.coherence.security.keystore"></param-value>`
		`  </init-param>`
		`  <init-param id="2">`
		`  <param-type>java.io.File</param-type>`
		`  <param-value system-property="tangosol.coherence.security.permissions"></param-value>`
		`  </init-param>`
		`</init-params>`
		
		Preconfigured value based on the default `AccessController` implementation and the default parameters as specified above are `tangosol.coherence.security.keystore` and `tangosol.coherence.security.permissions`. For more information on preconfigured overrides, see Appendix L, "Command Line Overrides." For more information on the elements you can define within the `init-param` element, see "init-param" on page H-16.

authorized-hosts

Used in: `cluster-config`.

Description

If specified, restricts cluster membership to the cluster nodes specified in the collection of unicast addresses, or address range. The unicast address is the address value from the authorized cluster nodes' `unicast-listener` element. Any number of `host-address` and `host-range` elements may be specified.

Elements

Table H–3 describes the subelements you can define within the `authorized-hosts` element.

Table H–3 authorized-hosts Subelements

Element	Required/ Optional	Description
`<host-address>`	Optional	Specifies an IP address or hostname. If any are specified, only hosts with specified host-addresses or within the specified host-ranges will be allowed to join the cluster. The content override attributes `id` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document.
`<host-range>`	Optional	Specifies a range of IP addresses. If any are specified, only hosts with specified host-addresses or within the specified host-ranges will be allowed to join the cluster. The content override attributes `id` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document.

The content override attributes `xml-override` and `id` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "Element Attributes" on page H-53.

burst-mode

Used in: packet-publisher.

Description

The burst-mode element is used to control the rate at which packets will transmitted on the network, by specifying the maximum number of packets to transmit without pausing. By default this feature is disabled and is typically only needed when flow-control is disabled, or when operating with heavy loads on a half-duplex network link. This setting only effects packets which are sent by the packet-speaker.

Elements

Table H–4 describes the subelements you can define within the burst-mode element.

Table H–4 burst-mode Subelements

Element	Required/ Optional	Description
<maximum-packets>	Required	Specifies the maximum number of packets that the will be sent in a row without pausing. Zero indicates no limit. By setting this value relatively low, Coherence is forced to hold back when sending a large number of packets, which may reduce collisions in some instances or allow incoming traffic to be more quickly processed. Default value is 0.
<pause-milliseconds>	Required	Specifies the minimum number of milliseconds to delay between long bursts of packets. By increasing this value, Coherence is forced to hold back when sending a large number of packets, which may reduce collisions in some instances or allow incoming traffic to be more quickly processed. Default value is 10.

callback-handler

Used in: security-config.

Table H–5 describes the elements you can define within the `callback-handler` element.

Table H–5 callback-handler Subelement

Element	Required/ Optional	Description
`<class-name>`	Required	Specifies the name of a Java class that provides the implementation for the `javax.security.auth.callback.CallbackHandler` interface.
`<init-params>`	Optional	Contains one or more initialization parameter(s) for a `CallbackHandler` implementation. For more information on the elements you can define within the init-param element, refer to "init-param" on page H-16.

cluster-config

Used in: <coherence>

Description

Contains the cluster configuration information, including communication and service parameters.

Elements

Table H–6 describes the subelements you can define within the `cluster-config` element.

Table H–6 cluster-config Subelement

Element	Required/Optional	Description
<authorized-hosts>	Optional	Specifies the hosts which are allowed to join the cluster.
<filters>	Optional	Specifies data transformation filters, which can be used to perform custom transformations on data being transferred between cluster nodes.
<incoming-message-handler>	Required	Specifies configuration information for the Incoming message handler, used for dispatching incoming cluster communications.
<member-identity>	Optional	Specifies detailed identity information that is useful for defining the location and role of the cluster member.
<multicast-listener>	Required	Specifies the configuration information for the Multicast listener, used for receiving point-to-multipoint network communications.
<outgoing-message-handler>	Required	Specifies configuration information for the Outgoing message handler, used for dispatching outgoing cluster communications.
<packet-publisher>	Required	Specifies configuration information for the Packet publisher, used for managing network data transmission.
<packet-speaker>	Required	Specifies configuration information for the Packet speaker, used for network data transmission.
<services>	Required	Specifies the declarative data for all available Coherence services.
<shutdown-listener>	Required	Specifies the action to take upon receiving an external shutdown request.
<tcp-ring-listener>	Required	Specifies configuration information for the TCP Ring listener, used to death detection.
<unicast-listener>	Required	Specifies the configuration information for the Unicast listener, used for receiving point-to-point network communications.

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "Element Attributes" on page H-53 for more information on this attribute.

coherence

root element

Description

The `coherence` element is the root element of the operational deployment descriptor `tangosol-coherence.xml`.

Elements

Table H–7 describes the elements you can define within the `coherence` element.

Table H–7 coherence Subelements

Element	Required/ Optional	Description
<cluster-config>	Required	Contains the cluster configuration information. This element is where most communication and service parameters are defined.
<logging-config>	Required	Contains the configuration information for the logging facility.
<configurable-cache-factory-config>	Required	Contains configuration information for the configurable cache factory. It controls where, from, and how the cache configuration settings are loaded.
<management-config>	Required	Contains the configuration information for the coherence Management Framework. See Chapter 22, "How to Manage Coherence Using JMX" for more information.
<security-config>	Optional	Contains the configuration information for the Coherence Security Framework.
<license-config>	Optional	Contains the edition and operational mode configuration.

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "Element Attributes" on page H-53 for more information on this attribute

configurable-cache-factory-config

Used in: coherence.

Elements

Table H–8 describes the elements you can define within the configurable-cache-factory-config element.

Table H–8 configurable-cache-factory-config Subelements

Element	Required/ Optional	Description
<class-name>	Required	Specifies the name of a Java class that provides the cache configuration factory. Default value is com.tangosol.net.DefaultConfigurableCacheFactory.
<init-params>	Optional	Contains one or more initialization parameter(s) for a cache configuration factory class which implements the com.tangosol.run.xml.XmlConfigurable interface.For the default cache configuration factory class (DefaultConfigurableCacheFactory) the parameters are specified as follows: <init-param> <param-type>java.lang.String</param-type> <param-value system-property="tangosol.coherence.cacheconfig"> coherence-cache-config.xml </param-value> </init-param> Preconfigured is tangosol.coherence.cacheconfig. Unless an absolute or relative path is specified, such as with ./path/to/config.xml, the application's classpath will be used to find the specified descriptor. See Appendix L, "Command Line Overrides" for more information on overrides.

The content override attribute xml-override can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "Element Attributes" on page H-53 for more information on this attribute.

filters

Used in: cluster-config.

Description

Data transformation filters can be used by services to apply a custom transformation on data being transferred between cluster nodes. This can be used for instance to compress or encrypt Coherence network traffic. See the <filter-class> element for more information.

Implementation

Data transformation filters are implementations of the com.tangosol.util.WrapperStreamFactory interface.

> **Note:** Data transformation filters are not related to com.tangosol.util.Filter, which is part of the Coherence API for querying caches.

Elements

Table H–9 describes the elements you can define within each filters element.

Table H–9 filters Subelements

Element	Required/ Optional	Description
<filter-name>	Required	Specifies the canonical name of the filter. This name is unique within the cluster. For example: gzip. The content override attribute id can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document.
<filter-class>	Required	Specifies the class name of the filter implementation. This class must have a zero-parameter public constructor and must implement the com.tangosol.util.WrapperStreamFactory interface.

<init-params>	Optional	Specifies initialization parameters, for configuring filters which implement the com.tangosol.run.xml.XmlConfigurable interface.For example when using a com.tangosol.net.CompressionFilter the parameters are specified as follows:
		`<init-param>`
		`  <param-name>strategy</param-name>`
		`  <param-value>gzip</param-value>`
		`</init-param>`
		`<init-param>`
		`  <param-name>level</param-name>`
		`  <param-value>default</param-value>`
		`</init-param>`
		For more information on the parameter values for the standard filters refer to, refer to Chapter 8, "Network Filters."

The content override attributes id and xml-override can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "Element Attributes" on page H-53 for more information on these attributes.

flow-control

Used in: packet-delivery.

Description

The flow-control element contains configuration information related to packet throttling and remote GC detection.

Remote GC Detection

Remote Pause detection allows Coherence to detect and react to a cluster node becoming unresponsive (likely due to a long GC). When a node is marked as paused, packets addressed to it will be sent at a lower rate until the node resumes responding. This remote GC detection is used to avoid flooding a node while it is incapable of responding.

Packet Throttling

Flow control allows Coherence to dynamically adjust the rate at which packets are transmitted to a given cluster node based on point to point transmission statistics.

Elements

Table H–10 describes the elements you can define within the `flow-control` subelement.

Table H–10 flow-control Subelements

Element	Required/ Optional	Description
<enabled>	Optional	Specifies if flow control is enabled. Default is true
<pause-detection>	Optional	Defines the number of packets that will be resent to an unresponsive cluster node before assuming that the node is paused.
<outstanding-packets>	Optional	Defines the number of unconfirmed packets that will be sent to a cluster node before packets addressed to that node will be deferred.

host-range

Used in: `authorized-hosts`.

Description

Specifies a range of unicast addresses of nodes which are allowed to join the cluster.

Elements

Table H–11 describes the elements you can define within each `host-range` element.

Table H–11 host-range Subelements

Element	Required/ Optional	Description
`<from-address>`	Required	Specifies the starting IP address for a range of host addresses. For example: 198.168.1.1.
`<to-address>`	Required	Specifies to-address element specifies the ending IP address (inclusive) for a range of hosts. For example: 198.168.2.255.

The content override attribute `id` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "Element Attributes" on page H-53 for more information on this attribute.

incoming-message-handler

Used in: `cluster-config`.

Description

The `incoming-message-handler` assembles UDP packets into logical messages and dispatches them to the appropriate Coherence service for processing.

Elements

Table H–12 describes the subelements you can define within the `incoming-message-handler` element.

Table H–12 incoming-message-handler Subelements

Element	Required/ Optional	Description
`<maximum-time-variance>`	Required	Specifies the maximum time variance between sending and receiving broadcast Messages when trying to determine the difference between a new cluster Member's system time and the cluster time. The smaller the variance, the more certain one can be that the cluster time will be closer between multiple systems running in the cluster; however, the process of joining the cluster will be extended until an exchange of Messages can occur within the specified variance. Normally, a value as small as 20 milliseconds is sufficient, but with heavily loaded clusters and multiple network hops it is possible that a larger value would be necessary. Default value is 16.
`<use-nack-packets>`	Required	Specifies whether the packet receiver will use negative acknowledgments (packet requests) to pro-actively respond to known missing packets. See "notification-queueing" on page H-28 for additional details and configuration. Legal values are `true` or `false`. Default value is `true`.
`<priority>`	Required	Specifies a priority of the incoming message handler execution thread. Legal values are from 1 to 10. Default value is 7.
`<packet-pool>`	Required	Specifies how many incoming packets Coherence will buffer before blocking.

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "Element Attributes" on page H-53 for more information on this attribute.

init-param

Used in: `init-params`.

Description

Defines an individual initialization parameter.

Elements

Table H–13 describes the elements you can define within the `init-param` element.

Table H–13 init-param Subelement

Element	Required/ Optional	Description
`<param-name>`	Optional	Specifies the name of the parameter passed to the class. The `param-type` or `param-name` must be specified. For example: `thread-count`. For more information on the pre-defined parameter values available for the specific elements, refer to Appendix I, "Initialization Parameter Settings".
`<param-type>`	Optional	Specifies the data type of the parameter passed to the class. The `param-type` or `param-name` must be specified. For example: `int`
`<param-value>`	Required	Specifies the value passed in the parameter. For example: `8`. For more information on the pre-defined parameter values available for the specific elements, refer to Appendix I, "Initialization Parameter Settings".

The content override attribute `id` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "Element Attributes" on page H-53 for more information no this attribute.

init-params

Used in: filters, services, configurable-cache-factory-config, access-controller and callback-handler.

Description

Defines a series of initialization parameters.

Elements

Table H–14 describes the elements you can define within the init-params element.

Table H–14 init-params Subelement

Element	Required/ Optional	Description
<init-param>	Optional	Defines an individual initialization parameter.

license-config

Used in: coherence.

Table H–15 describes the elements you can define within the license-config element.

Table H–15 license-config Subelements

Element	Required/ Optional	Description
<edition-name>	Optional	Specifies the product edition that the member will use. This allows multiple product editions to be used within the same cluster, with each member specifying the edition that it will be using. Valid values are: GE (Grid Edition), EE (Enterprise Edition), SE (Standard Edition), RTC (Real-Time Client), DC (Data Client). Default value is GE.
<license-mode>	Optional	Specifies whether the product is being used in an development or production mode. Valid values are prod (Production), and dev (Development). **Note:** This value cannot be overridden in tangosol-coherence-override.xml. It must be specified in tangosol-coherence.xml or (preferably) supplied as system property tangosol.coherence.mode on the Java command line. Default value is dev.

logging-config

Used in: coherence.

Elements

The following table describes the elements you can define within the logging-config element.

Table H–16 logging-config Subelements

Element	Required/ Optional	Description
<destination>	Required	Specifies the output device used by the logging system. Legal values are: ■ `stdout` ■ `stderr` (default) ■ `jdk` ■ `log4j` ■ *a file name* If `jdk` is specified as the destination, Coherence must be run using JDK 1.4 or later; likewise, if `log4j` is specified, the Log4j libraries must be in the classpath. In both cases, the appropriate logging configuration mechanism (system properties, property files, and so on) are necessary to configure the `JDK/Log4j` logging libraries. Preconfigured value is `tangosol.coherence.log`. See Appendix L, "Command Line Overrides" for more information.

Table H–16 (Cont.) logging-config Subelements

Element	Required/ Optional	Description
<severity-level>	Required	Specifies which logged messages will be output to the log destination. Legal values are: ■ 0—only output without a logging severity level specified will be logged ■ 1—all the above plus errors ■ 2—all the above plus warnings ■ 3—all the above plus informational messages ■ 4-9—all the above plus internal debugging messages (the higher the number, the more the messages) ■ -1—no messages Default value is 3. Preconfigured value is tangosol.coherence.log.level. See Appendix L, "Command Line Overrides" for more information.
<message-format>	Required	Specifies how messages that have a logging level specified will be formatted before passing them to the log destination. The value of the message-format element is static text with the following replaceable parameters: ■ {date}—the date/time format (to a millisecond) at which the message was logged ■ {version}—the Oracle Coherence exact version and build details ■ {level}—the logging severity level of the message ■ {thread}—the thread name that logged the message ■ {member}—the cluster member id (if the cluster is currently running) ■ {location}—the fully qualified cluster member id: cluster-name, site-name, rack-name, machine-name, process-name and member-name (if the cluster is currently running) ■ {role}—the specified role of the cluster member ■ {text}—the text of the message Default value is: {date} Oracle Coherence {version} <{level}> (thread={thread}, member={member}): {text}

Table H–16 (Cont.) logging-config Subelements

Element	Required/ Optional	Description
`<character-limit>`	Required	Specifies the maximum number of characters that the logger daemon will process from the message queue before discarding all remaining messages in the queue. Note that the message that caused the total number of characters to exceed the maximum will **NOT** be truncated, and all messages that are discarded will be summarized by the logging system with a single log entry detailing the number of messages that were discarded and their total size. The truncation of the logging is only temporary, since when the queue is processed (emptied), the logger is reset so that subsequent messages will be logged. The purpose of this setting is to avoid a situation where logging can itself prevent recovery from a failing condition. For example, with tight timings, logging can actually change the timings, causing more failures and probably more logging, which becomes a vicious cycle. A limit on the logging being done at any one point in time is a "pressure valve" that prevents such a vicious cycle from occurring. Note that logging occurs on a dedicated low-priority thread to even further reduce its impact on the critical portions of the system. Legal values are positive integers or zero. Zero implies no limit. Default value is 4096. Preconfigured value is `tangosol.coherence.log.limit`. For more information, see Appendix L, "Command Line Overrides."

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "Element Attributes" on page H-53 for more information on this attribute.

management-config

Used in: coherence.

Elements

Table H–17 describes the elements you can define within the management-config element.

Table H–17 management-config Subelements

Element	Optional/ Required	Description
<default-domain-name>	Required	Specifies the name of the JMX domain used to register MBeans exposed by the Coherence Management Framework. See Chapter 22, "How to Manage Coherence Using JMX" for more information.
<managed-nodes>	Required	Specifies whether a cluster node's JVM has an [in-process] MBeanServer and if so, whether this node allows management of other nodes' managed objects.Legal values are: ▪ none—No MBeanServer is instantiated. ▪ local-only—Manage only MBeans which are local to the cluster node (that is, within the same JVM). ▪ remote-only—Manage MBeans on other remotely manageable cluster nodes. See <allowed-remote-management> subelement. **Requires** Coherence Enterprise Edition or higher ▪ all—Manage both local and remotely manageable cluster nodes. See <allowed-remote-management> subelement. **Requires** Coherence Enterprise Edition or higher. Default value is none. Preconfigured value is tangosol.coherence.management. See Appendix L, "Command Line Overrides" for more information.
<allow-remote-management>	Required	Specifies whether this cluster node exposes its managed objects to remote MBeanServer(s). Legal values are: true or false. Default value is false. Preconfigured value is tangosol.coherence.management.remote. See Appendix L, "Command Line Overrides" for more information.
<read-only>	Required	Specifies whether the managed objects exposed by this cluster node allow operations that modify run-time attributes. Legal values are: true or false. Default value is false. Preconfigured value is tangosol.coherence.management.readonly. See Appendix L, "Command Line Overrides"
<service-name>	Required	Specifies the name of the Invocation Service used for remote management. This element is used only if allow-remote-management is set to true.

The content override attribute xml-override can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "Element Attributes" on page H-53 for more information..

member-identity

Used in: `cluster-config`.

The `member-identity` element contains detailed identity information that is useful for defining the location and role of the cluster member.

Elements

Table H–18 describes the elements you can define within the member-identity element.

Table H–18 member-identity Subelements

Element	Required/ Optional	Description
`<cluster-name>`	Optional	The `cluster-name` element contains the name of the cluster. To join the cluster all members must specify the same cluster name. It is strongly suggested that cluster-name be specified for production systems, thus preventing accidental cluster discovery among applications. Preconfigured value is `tangosol.coherence.cluster`. See Appendix L, "Command Line Overrides" for more information.
`<site-name>`	Optional	The site-name element contains the name of the geographic site that the member is hosted at. For WAN clustering, this value identifies the datacenter within which the member is located, and can be used as the basis for intelligent routing, load balancing and disaster recovery planning (that is, the explicit backing up of data on separate geographic sites). The name is also useful for displaying management information (for example, JMX) and interpreting log entries. It is optional to provide a value for this element. Deployments that spread across more than one geographic site should specify a site-name value. Preconfigured value is `tangosol.coherence.site`. See Appendix L, "Command Line Overrides" for more information.
`<rack-name>`	Optional	The rack-name element contains the name of the location within a geographic site that the member is hosted at. This is often a cage, rack or bladeframe identifier, and can be used as the basis for intelligent routing, load balancing and disaster recovery planning (that is, the explicit backing up of data on separate bladeframes). The name is also useful for displaying management information (for example, JMX) and interpreting log entries. It is optional to provide a value for this element. Large scale deployments should always specify a rack-name value. Preconfigured value is `tangosol.coherence.rack`. See Appendix L, "Command Line Overrides" for more information.
`<machine-name>`	Optional	The machine-name element contains the name of the physical server that the member is hosted on. This is often the same name as the server identifies itself as (for example, its `HOSTNAME`, or its name as it appears in a DNS entry). If provided, the machine-name is used as the basis for creating a machine-id, which in turn is used to guarantee that data are backed up on different physical machines to prevent single points of failure (SPOFs). The name is also useful for displaying management information (for example, JMX) and interpreting log entries. It is optional to provide a value for this element. However, it is strongly encouraged that a name always be provided. Preconfigured value is tangosol.coherence.machine. See Appendix L, "Command Line Overrides" for more information.

Table H–18 (Cont.) member-identity Subelements

Element	Required/ Optional	Description
<process-name>	Optional	The process-name element contains the name of the process (JVM) that the member is hosted on. This name makes it possible to easily differentiate among multiple JVMs running on the same machine. The name is also useful for displaying management information (for example, JMX) and interpreting log entries. It is optional to provide a value for this element. Often, a single member will exist per JVM, and in that situation this name would be redundant. Preconfigured value is tangosol.coherence.process. See Appendix L, "Command Line Overrides" for more information.
<member-name>	Optional	The member-name element contains the name of the member itself. This name makes it possible to easily differentiate among members, such as when multiple members run on the same machine (or even within the same JVM).The name is also useful for displaying management information (for example, JMX) and interpreting log entries. It is optional to provide a value for this element. However, it is strongly encouraged that a name always be provided. Preconfigured value is tangosol.coherence.member. see Appendix L, "Command Line Overrides" for more information.
<role-name>	Optional	The role-name element contains the name of the member role. This name allows an application to organize members into specialized roles, such as cache servers and cache clients. The name is also useful for displaying management information (for example, JMX) and interpreting log entries. It is optional to provide a value for this element. However, it is strongly encouraged that a name always be provided. Preconfigured value is tangosol.coherence.role. See Appendix L, "Command Line Overrides" for more information.
<priority>	Optional	The priority element specifies a priority of the corresponding member. The priority is used as the basis for determining tie-breakers between members. If a condition occurs in which one of two members will be ejected from the cluster, and in the rare case that it is not possible to objectively determine which of the two is at fault and should be ejected, then the member with the lower priority will be ejected. Valid values are from 1 to 10. Preconfigured value is tangosol.coherence.priority. See Appendix L, "Command Line Overrides" for more information.

multicast-listener

Used in: `cluster-config`.

Description

Specifies the configuration information for the Multicast listener. This element is used to specify the address (see `<address>` subelement) and port (see `<port>` subelement) that a cluster will use for cluster wide and point-to-multipoint communications. All nodes in a cluster must use the same multicast address and port, whereas distinct clusters on the same network should use different multicast addresses.

Multicast-Free Clustering

By default, Coherence uses a multicast protocol to discover other nodes when forming a cluster. If multicast networking is undesirable, or unavailable in your environment, the `well-known-addresses` feature may be used to eliminate the need for multicast traffic. If you are having difficulties in establishing a cluster by using multicast, see Chapter 16, "Performing a Multicast Connectivity Test."

Elements

Table H–19 describes the elements you can define within the `multicast-listener` element.

Table H–19 multicast-listener Subelements

Element	Required /Optional	Description
`<address>`	Required	Specifies the multicast IP address that a Socket will listen or publish on. Legal values are from 224.0.0.0 to 239.255.255.255. Default value depends on the release and build level and typically follows the convention of `{build}.{major version}.{minor version}.{patch}`. For example, for Coherence Release 2.2 build 255 it is 225.2.2.0. Preconfigured is `tangosol.coherence.clusteraddress`. See Appendix L, "Command Line Overrides" for more information.
`<port>`	Required	Specifies the port that the Socket will listen or publish on. Legal values are from 1 to 65535. Default value depends on the release and build level and typically follows the convention of `{version}+{{{build}}}`. For example, for Coherence Release 2.2 build 255 it is 22255. Preconfigured value is `tangosol.coherence.clusterport`. See Appendix L, "Command Line Overrides" for more information.
`<time-to-live>`	Required	Specifies the time-to-live setting for the multicast. This determines the maximum number of "hops" a packet may traverse, where a hop is measured as a traversal from one network segment to another by using a router. Legal values are from 0 to 255. Default value is 4. Preconfigured value is `tangosol.coherence.ttl`. See Appendix L, "Command Line Overrides" for more information.
`<packet-buffer>`	Required	Specifies how many incoming packets the operating system will be requested to buffer.

Table H–19 (Cont.) multicast-listener Subelements

Element	Required /Optional	Description
<priority>	Required	Specifies a priority of the multicast listener execution thread. Legal values are from 1 to 10. Default value is 8.
<join-timeout-milliseconds>	Required	Specifies the number of milliseconds that a new member will wait without finding any evidence of a cluster before starting its own cluster and electing itself as the senior cluster member. Legal values are from 1 to 1000000. **Note:** For production use, the recommended value is 30000. Default value is 6000.
<multicast-threshold-percent>	Required	Specifies the threshold percentage value used to determine whether a packet will be sent by using unicast or multicast. It is a percentage value and is in the range of 1% to 100%. In a cluster of "n" nodes, a particular node sending a packet to a set of other (that is, not counting self) destination nodes of size "d" (in the range of 0 to n-1), the packet will be sent multicast if and only if the following both hold true: 1. The packet is being sent over the network to more than one other node, that is, (d > 1). 2. The number of nodes is greater than the threshold, that is, (d > (n-1) * (threshold/100)). Setting this value to 1 will allow the implementation to use multicast for basically all multi-point traffic. Setting it to 100 will force the implementation to use unicast for all multi-point traffic except for explicit broadcast traffic (for example, cluster heartbeat and discovery) because the 100% threshold will never be exceeded. With the setting of 25 the implementation will send the packet using unicast if it is destined for less than one-fourth of all nodes, and send it using multicast if it is destined for the one-fourth or more of all nodes. **Note:** This element is only used if the well-known-addresses element is empty. Legal values are from 1 to 100. Default value is 25.

The content override attribute xml-override can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "Element Attributes" on page H-53 for more information on this attribute.

notification-queueing

Used in: packet-publisher.

Description

The notification-queueing element is used to specify the timing of notifications packets sent to other cluster nodes. Notification packets are used to acknowledge the receipt of packets which require confirmation.

Batched Acknowledgments

Rather then sending an individual ACK for each received packet which requires confirmation, Coherence will batch a series of acknowledgments for a given sender into a single ACK. The <ack-delay-milliseconds> specifies the maximum amount of time that an acknowledgment will be delayed before an ACK notification is sent. By batching the acknowledgments Coherence avoids wasting network bandwidth with many small ACK packets.

Negative Acknowledgments

When enabled cluster nodes will use packet ordering to perform early packet loss detection (see the <use-nack-packets> subelement of <incoming-message-handler>). This allows Coherence to identify a packet as likely being lost and retransmit it well before the packets scheduled (see the <resend-milliseconds> subelement of <packet-delivery>).

Elements

The following table describes the elements you can define within the notification-queuing element.

Table H–20 notification-queuing Subelements

Element	Required/ Optional	Description
<ack-delay-milliseconds>	Required	Specifies the maximum number of milliseconds that the packet publisher will delay before sending an ACK packet. The ACK packet may be transmitted earlier if number of batched acknowledgments fills the ACK packet. This value should be substantially lower then the remote node's packet-delivery resend timeout, to allow ample time for the ACK to be received and processed by the remote node before the resend timeout expires. Default value is 16.
<nack-delay-milliseconds>	Required	Specifies the number of milliseconds that the packet publisher will delay before sending a NACK packet. Default value is 1.

outgoing-message-handler

Used in: `acceptor-config`, `initiator-config`.

Description

The `outgoing-message-handler` specifies the configuration info used to detect dropped client-to-cluster connections. For connection initiators and acceptors that use connectionless protocols (for example, JMS), this information is necessary to proactively detect and release resources allocated to dropped connections. Connection-oriented initiators and acceptors can also use this information as an additional mechanism to detect dropped connections.

Elements

Table H–21 describes the elements you can define within the `outgoing-message-handler` element.

Table H–21 outgoing-message-handler Subelement

Element	Required/ Optional	Description
`<heartbeat-interval>`	Optional	Specifies the interval between ping requests. A ping request is used to ensure the integrity of a connection.The value of this element must be in the following format: `[\d]+[[.][\d]+]?[MS|ms|S|s|M|m|H|h|D|d]?` where the first non-digits (from left to right) indicate the unit of time duration: ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) If the value does not contain a unit, a unit of milliseconds is assumed. A value of zero disables ping requests. The default value is zero.
`<heartbeat-timeout>`	Optional	Specifies the maximum amount of time to wait for a response to a ping request before declaring the underlying connection unusable.The value of this element must be in the following format: `[\d]+[[.][\d]+]?[MS|ms|S|s|M|m|H|h|D|d]?` where the first non-digits (from left to right) indicate the unit of time duration: ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) If the value does not contain a unit, a unit of milliseconds is assumed. The default value is the value of the request-timeout element.
`<request-timeout>`	Optional	Specifies the maximum amount of time to wait for a response message before declaring the underlying connection unusable.The value of this element must be in the following format: `[\d]+[[.][\d]+]?[MS|ms|S|s|M|m|H|h|D|d]?` where the first non-digits (from left to right) indicate the unit of time duration: ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) If the value does not contain a unit, a unit of milliseconds is assumed. The default value is an infinite timeout.

outstanding-packets

Used in: `flow-control`.

Description

Defines the number of unconfirmed packets that will be sent to a cluster node before packets addressed to that node will be deferred. This helps to prevent the sender from flooding the recipient's network buffers.

Auto Tuning

The value may be specified as either an explicit number by using the `maximum-packets` element, or as a range by using both the `maximum-packets` and `minimum-packets` elements. When a range is specified, this setting will be dynamically adjusted based on network statistics.

Elements

Table H–22 describes the elements you can define within the outstanding-packets element.

Table H–22 outstanding-packets Subelements

Element	Required/ Optional	Description
`<maximum-packets>`	Optional	The maximum number of unconfirmed packets that will be sent to a cluster node before packets addressed to that node will be deferred. It is recommended that this value not be set below 256. Default is 4096.
`<minimum-packets>`	Optional	The lower bound on the range for the number of unconfirmed packets that will be sent to a cluster node before packets addressed to that node will be deferred. It is recommended that this value not be set below 16. Default is 64.

packet-buffer

Used in: unicast-listener, multicast-listener, packet-publisher.

Description

Specifies the size of the operating system buffer for datagram sockets.

Performance Impact

Large inbound buffers help insulate the Coherence network layer from JVM pauses caused by the Java Garbage Collector. While the JVM is paused, Coherence is unable to dequeue packets from any inbound socket. If the pause is long enough to cause the packet buffer to overflow, the packet reception will be delayed as the originating node will need to detect the packet loss and retransmit the packet(s).

It's just a hint

The operating system will only treat the specified value as a hint, and is not required to allocate the specified amount. In the event that less space is allocated then requested Coherence will issue a warning and continue to operate with the constrained buffer, which may degrade performance. See `http://forums.oracle.com/forums/forum.jspa?forumID=480&start=0` for details on configuring your operating system to allow larger buffers.

Elements

Table H–23 describes the elements you can define within the `packet-buffer` element.

Table H–23 packet-buffer Subelements

Element	Required/ Optional	Description
<maximum-packets>	Required	For unicast-listener, multicast-listener and packet-publisher: Specifies the number of packets of packet-size that the datagram socket will be asked to size itself to buffer. See `SO_SNDBUF` and `SO_RCVBUF`. Actual buffer sizes may be smaller if the underlying socket implementation cannot support more than a certain size. Defaults are 32 for publishing, 64 for multicast listening, and 1428 for unicast listening.

packet-bundling

Used in: packet-delivery.

Description

The packet-bundling element contains configuration information related to the bundling of multiple small packets into a single larger packet to reduce the load on the network switching infrastructure.

Default Configuration

The default packet-bundling settings are minimally aggressive allowing for bundling to occur without adding a measurable delay. The benefits of more aggressive bundling will be based on the network infrastructure and the application object's typical data sizes and access patterns.

Elements

Table H–24 describes the elements you can define within the packet-bundling element.

Table H–24 packet-bundling Subelements

Element	Required/Optional	Description
<maximum-deferral-time>	Optional	The maximum amount of time to defer a packet while waiting for additional packets to bundle. A value of zero will result in the algorithm not waiting, and only bundling the readily accessible packets. A value greater than zero will cause some transmission deferral while waiting for additional packets to become available. This value is typically set below 250 microseconds to avoid a detrimental throughput impact. If the units are not specified, nanoseconds are assumed. Default value is 1us (microsecond).
<aggression-factor>	Optional	Specifies the aggressiveness of the packet deferral algorithm. Where as the maximum-deferral-time element defines the upper limit on the deferral time, the aggression-factor influences the average deferral time. The higher the aggression value, the longer the Publisher may wait for additional packets. The factor may be expressed as a real number, and often times values between 0.0 and 1.0 will be allow for high packet utilization while keeping latency to a minimum. Default value is zero.

packet-pool

Used in: `incoming-message-handler`, `packet-publisher`.

Description

Specifies the number of packets which Coherence will internally maintain for use in transmitting and receiving UDP packets. Unlike the `packet-buffer` these buffers are managed by Coherence rather then the operating system, and allocated on the JVM's heap.

Performance Impact

The packet pools are used as a reusable buffer between Coherence network services. For packet transmission, this defines the maximum number of packets which can be queued on the `packet-speaker` before the `packet-publisher` must block. For packet reception, this defines the number of packets which can be queued on the `incoming-message-handler` before the `unicast-listener`, and `multicast-listener` must block.

Elements

Table H–25 describes the subelements you can define within the `packet-pool` element.

Table H–25 packet-pool Subelements

Element	Required/ Optional	Description
`<maximum-packets>`	Required	The maximum number of reusable packets to be used by the services responsible for publishing and receiving. The pools are initially small, and will grow on demand up to the specified limits. Defaults are 2048 for transmitting and receiving.

packet-delivery

Used in: packet-publisher.

Description

Specifies timing and transmission rate parameters related to packet delivery.

Death Detection

The <timeout-milliseconds> and <heartbeat-milliseconds> subelements are used in detecting the death of other cluster nodes.

Elements

Table H–26 describes the elements you can define within the packet-delivery element.

Table H–26 packet-delivery Subelements

Element	Required/ Optional	Description
<resend-milliseconds>	Required	For packets which require confirmation, specifies the minimum amount of time in milliseconds to wait for a corresponding ACK packet, before resending a packet. Default value is 200.
<timeout-milliseconds>	Required	For packets which require confirmation, specifies the maximum amount of time, in milliseconds, that a packet will be resent. After this timeout expires Coherence will make a determination if the recipient is to be considered "dead". This determination takes additional data into account, such as if other nodes are still able to communicate with the recipient. Default value is 60000.**Note:** For production use, the recommended value is the greater of 60000 and two times the maximum expected full GC duration.
<heartbeat-milliseconds>	Required	Specifies the interval between heartbeats. Each member issues a unicast heartbeat, and the most senior member issues the cluster heartbeat, which is a broadcast message. The heartbeat is used by the tcp-ring-listener as part of fast death detection. Default value is 1000.
<flow-control>	Optional	Configures per-node packet throttling and remote GC detection.
<packet-bundling>	Optional	Configures how aggressively Coherence will attempt to maximize packet utilization.

packet-publisher

Used in: `cluster-config`.

Description

Specifies configuration information for the Packet publisher, which manages network data transmission.

Reliable packet delivery

The Packet publisher is responsible for ensuring that transmitted packets reach the destination cluster node. The publisher maintains a set of packets which are waiting to be acknowledged, and if the ACK does not arrive by the `packet-delivery` resend timeout, the packet will be retransmitted (see `<packet-delivery>` subelement). The recipient node will delay the ACK, to batch a series of ACKs into a single response (see `<notification-queuing>` subelement).

Throttling

The rate at which the publisher will accept and transmit packet may be controlled by using the `traffic-jam` and `flow-control` settings. Throttling may be necessary when dealing with slow networks, or small `packet-buffer`.

Elements

Table H–27 describes the elements you can define within the `packet-publisher` element.

Table H–27 packet-publisher Subelements

Element	Required/ Optional	Description
<enabled>	Required	Specifies if TCMP clustering is enabled. For Coherence editions which support both Coherence Extend and Coherence TCMP based clustering, this feature allows TCMP to be disabled to ensure that a node only connects by using the Extend protocol. Default value is `true`. Preconfigured value is `tangosol.coherence.tcmp.enabled`. See Appendix L, "Command Line Overrides" for more information.
<packet-size>	Required	Specifies the UDP packet sizes to use.
<packet-delivery>	Required	Specifies timing parameters related to reliable packet delivery.
<notification-queueing>	Required	Contains the notification queue related configuration info.
<burst-mode>	Required	Specifies the maximum number of packets the publisher may transmit without pausing.
<traffic-jam>	Required	Specifies the maximum number of packets which can be enqueued on the publisher before client threads block.
<packet-buffer>	Required	Specifies how many outgoing packets the operating system will be requested to buffer.
<packet-pool>	Required	Specifies how many outgoing packets Coherence will buffer before blocking.
<priority>	Required	Specifies a priority of the packet publisher execution thread. Legal values are from 1 to 10. Default value is 6.

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "Element Attributes" on page H-53 for more information.

packet-size

Used in: `packet-publisher`.

Description

The packet-size element specifies the maximum and preferred UDP packet sizes (see the `<maximum-length>` and `<preferred-length>` subelements). All cluster nodes must use identical maximum packet sizes. For optimal network utilization this value should be 32 bytes less then the network MTU.

Note: When specifying a UDP packet size larger then 1024 bytes on Microsoft Windows a registry setting must be adjusted to allow for optimal transmission rates. See "Datagram size (Microsoft Windows)" on page 20-3 for details.

Elements

Table H–28 describes the subelements you can define within the `packet-size` element.

Table H–28 packet-size Subelement

Element	Required/ Optional	Description
`<maximum-length>`	Required	Specifies the maximum size, in bytes, of the UDP packets that will be sent and received on the unicast and multicast sockets. This value should be at least 512; recommended value is 1468 for 100Mb, and 1Gb Ethernet. This value must be identical on all cluster nodes. **Note:** Some network equipment cannot handle packets larger than 1472 bytes (IPv4) or 1468 bytes (IPv6), particularly under heavy load. If you encounter this situation on your network, this value should be set to 1472 or 1468 respectively. The recommended values is 32 bytes less then the network MTU setting. Default value is 1468.
`<preferred-length>`	Required	Specifies the preferred size, in bytes, of UDP packets that will be sent and received on the unicast and multicast sockets. This value should be at least 512 and cannot be greater than the maximum-length value; it is recommended to set the value to the same as the maximum-length value. Default value is 1468.

packet-speaker

Used in: `cluster-config`.

Description

Specifies configuration information for the Packet speaker, used for network data transmission.

Offloaded Transmission

The Packet speaker is responsible for sending packets on the network. The speaker is used when the `packet-publisher` detects that a network send operation is likely to block. This allows the Packet publisher to avoid blocking on IO and continue to prepare outgoing packets. The Publisher will dynamically choose whether to use the speaker as the packet load changes.

Elements

Table H–29 describes the subelements you can define within the `packet-speaker` element.

Table H–29 packet-speaker Subelements

Element	Required/ Optional	Description
`<volume-threshold>`	Optional	Specifies the packet load which must be present for the speaker to be activated.
`<priority>`	Required	Specifies a priority of the packet speaker execution thread. Legal values are from 1 to 10. Default value is 8.

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "Element Attributes" on page H-53 for more information on this attribute.

pause-detection

Used in: `flow-control`.

Description

Remote Pause detection allows Coherence to detect and react to a cluster node becoming unresponsive (likely due to a long GC). When a node is marked as paused, packets addressed to it will be sent at a lower rate until the node resumes responding. This remote GC detection is used to avoid flooding a node while it is incapable of responding.

Elements

Table H–30 describes the subelements you can define within the `pause-detection` element.

Table H–30 pause-detection Subelements

Element	Required/ Optional	Description
`<maximum-packets>`	Optional	The maximum number of packets that will be resent to an unresponsive cluster node before assuming that the node is paused. Specifying a value of 0 will disable pause detection. Default is 16.

security-config

Used in: coherence.

Elements

Table H–31 describes the subelements you can define within the security-config element.

Table H–31 security-config Subelements

Element	Required/ Optional	Description
<enabled>	Required	Specifies whether the security features are enabled. All other configuration elements in the security-config group will be verified for validity and used if and only if the value of this element is true. Legal values are true or false. Default value is false. Preconfigured value is tangosol.coherence.security. See Appendix L, "Command Line Overrides" for more information.
<login-module-name>	Required	Specifies the name of the JAAS LoginModule that should be used to authenticate the caller. This name should match a module in a configuration file will be used by the JAAS (for example specified by using the -Djava.security.auth.login.config Java command line attribute). For details please refer to the Sun Login Module Developer's Guide.
<access-controller>	Required	Contains the configuration information for the class that implements com.tangosol.net.security.AccessController interface, which will be used by the security framework to check access rights for clustered resources and encrypt/decrypt node-to-node communications regarding those rights. See Chapter 7, "Security Framework" for more information.
<callback-handler>	Optional	Contains the configuration information for the class that implements javax.security.auth.callback.CallbackHandler interlace which will be called if an attempt is made to access a protected clustered resource when there is no identity associated with the caller.

The content override attribute xml-override can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "Element Attributes" on page H-53 for more information.

services

Used in: `cluster-config`.

Description

Specifies the configuration for Coherence services.

Service Components

The types of services which can be configured includes:

- `ReplicatedCache`—A cache service which maintains copies of all cache entries on all cluster nodes which run the service.
- `ReplicatedCache.Optimistic`—A version of the `ReplicatedCache` which uses optimistic locking.
- `DistributedCache`—A cache service which evenly partitions cache entries across the cluster nodes which run the service.
- `SimpleCache` —A version of the `ReplicatedCache` which lacks concurrent control.
- `LocalCache`—A cache service for caches where all cache entries reside in a single cluster node.
- `InvocationService`—A service used for performing custom operations on remote cluster nodes.

Elements

Table H–32 describes the subelements you can define for each `services` element.

Table H–32 services Subelements

Element	Required/ Optional	Description
<service-type>	Required	Specifies the canonical name for a service, allowing the service to be referenced from the service-name element in cache configuration caching schemes. See "caching-schemes" on page D-19 for more information.
<service-component>	Required	Specifies either the fully qualified class name of the service or the relocatable component name relative to the base Service component. Legal values are: ■ ReplicatedCache ■ ReplicatedCache.Optimistic ■ DistributedCache ■ SimpleCache ■ LocalCache ■ InvocationService
<use-filters>	Optional	Contains the list of filters names to be used by this service.For example, specify use-filter as follows <use-filters> <filter-name>gzip</filter-name> </use-filters> will activate gzip compression for the network messages used by this service, which can help substantially with WAN and low-bandwidth networks.
<init-params>	Optional	Specifies the initialization parameters that are specific to each service-component. For more service specific parameter information see: ■ "DistributedCache Service Parameters" on page I-3 ■ "ReplicatedCache Service Parameters" on page I-7 ■ "InvocationService Parameters" on page I-8

The content override attributes xml-override and id can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document.

shutdown-listener

Used in: `cluster-config`.

Description

Specifies the action a cluster node should take upon receiving an external shutdown request. External shutdown includes the "kill" command on UNIX and `Ctrl-C` on Windows and UNIX.

Elements

Table H–33 describes the elements you can define within the `shutdown-listener` element.

Table H–33 shutdown-listener Subelements

Element	Required/ Optional	Description
`<enabled>`	Required	Specifies the type of action to take upon an external JVM shutdown. Legal values: ▪ `none`—perform no explicit shutdown actions ▪ `force`—perform "hard-stop" the node by calling `Cluster.stop()` ▪ `graceful`—perform a "normal" shutdown by calling `Cluster.shutdown()` ▪ `true`—same as force ▪ `false`—same as none **Note:** For production use, the suggested value is none unless testing has verified that the behavior on external shutdown is exactly what is desired. Default value is force. Preconfigured value is `tangosol.coherence.shutdownhook`. See Appendix L, "Command Line Overrides" for more information.

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "Element Attributes" on page H-53 for more information.

socket-address

Used in: `well-known-addresses`, `tcp-initiator`.

Elements

Table H–34 describes the subelements you can define within the `socket-address` element.

Table H–34 socket-address Subelements

Element	Required/ Optional	Description
`<address>`	Required	Specifies the IP address that a Socket will listen or publish on. **Note:** The localhost setting may not work on systems that define localhost as the loopback address; in that case, specify the machine name or the specific IP address.
`<port>`	Required	Specifies the port that the Socket will listen or publish on. Legal values are from 1 to 65535.

tcp-ring-listener

Used in: `cluster-config`.

Description

The TCP-ring provides a means for fast death detection of another node within the cluster. When enabled the cluster nodes form a single "ring" of TCP connections spanning the entire cluster. A cluster node is able to use the TCP connection to detect the death of another node within a heartbeat interval (default is one second; see the `<heartbeat-milliseconds>` subelement of `packet-delivery`). If disabled, the cluster node must rely on detecting that another node has stopped responding to UDP packets for a considerately longer interval (see the `<timeout-milliseconds>` subelement of `packet-delivery`). When the death has been detected it is communicated to all other cluster nodes.

Elements

Table H–35 describes the subelements you can define within the `tcp-ring-listener` element.

Table H–35 tcp-ring-listener Subelements

Element	Required/ Optional	Description
<enabled>	Required	Specifies whether the tcp ring listener should be enabled to defect node failures faster. Legal values are `true` and `false`. Default value is `true`. Preconfigured value is `tangosol.coherence.tcpring`. see Appendix L, "Command Line Overrides" for more information.
`<maximum-socket-closed-exceptions>`	Required	Specifies the maximum number of tcp ring listener exceptions that will be tolerated before a particular member is considered really gone and is removed from the cluster. This value is used only if the value of `tcp-ring-listener/enabled` is `true`. Legal values are integers greater than zero. Default value is 2.
`<priority>`	Required	Specifies a priority of the tcp ring listener execution thread. Legal values are from 1 to 10. Default value is 6.

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "Element Attributes" on page H-53 for more information on this attribute.

traffic-jam

Used in: packet-publisher.

Description

The traffic-jam element is used to control the rate at which client threads enqueue packets for the Packet publisher to transmit on the network. When the limit is exceeded any client thread will be forced to pause until the number of outstanding packets drops below the specified limit. To limit the rate at which the Publisher transmits packets see the flow-control, and burst-mode elements.

Tuning

Specifying a limit which is to low, or a pause which is to long may result in the publisher transmitting all pending packets, and being left without packets to send. An ideal value will ensure that the publisher is never left without work to do, but at the same time prevent the queue from growing uncontrollably. It is therefore recommended that the pause remain quite short (10ms or under), and that the limit on the number of packets be kept high (that is, greater than 5000). As of Coherence 3.2 a warning will be periodically logged if this condition is detected.

Traffic Jam and Flow Control

When flow-control is enabled the traffic-jam operates in a point-to-point mode, only blocking a send if the recipient has too many packets outstanding. It is recommended that the traffic-jam/maximum-packets value be greater than the value (see the <maximum-packets> subelement of outstanding-packets). When flow-control is disabled, the traffic-jam will take all outstanding packets into account.

Elements

Table H–36 describes the subelements you can define within the traffic-jam element.

Table H–36 traffic-jam Subelements

Element	Required/ Optional	Description
<maximum-packets>	Required	Specifies the maximum number of pending packets that the Publisher will tolerate before determining that it is clogged and must slow down client requests (requests from local non-system threads). Zero means no limit. This property prevents most unexpected out-of-memory conditions by limiting the size of the resend queue. Default value is 8192.
<pause-milliseconds>	Required	Number of milliseconds that the Publisher will pause a client thread that is trying to send a message when the Publisher is clogged. The Publisher will not allow the message to go through until the clog is gone, and will repeatedly sleep the thread for the duration specified by this property. Default value is 10.

unicast-listener

Used in: `cluster-config`.

Description

Specifies the configuration information for the Unicast listener. This element is used to specify the address and port that a cluster node will bind to, to listen for point-to-point cluster communications.

Automatic Address Settings

By default Coherence will attempt to obtain the IP to bind to using the `java.net.InetAddress.getLocalHost()` call. On machines with multiple IPs or NICs you may need to explicitly specify the address (see the `<address>` subelement). Additionally if the specified port is already in use, Coherence will by default auto increment the port number until the binding succeeds (see the `<port>` and `<auto>` subelements).

Multicast-Free Clustering

By default Coherence uses a multicast protocol to discover other nodes when forming a cluster. If multicast networking is undesirable, or unavailable in your environment, the well-known-addresses feature may be used to eliminate the need for multicast traffic. If you are having difficulties in establishing a cluster by using multicast, see Chapter 16, "Performing a Multicast Connectivity Test."

Elements

Table H–37 describes the subelements you can define within the `unicast-listener` element.

Table H–37 unicast-listener Subelements

Element	Required/ Optional	Description
`<well-known-addresses>`	Optional	Contains a list of "well known" addresses (WKA) that are used by the cluster discovery protocol in place of multicast broadcast.
`<machine-id>`	Required	Specifies an identifier that should uniquely identify each server machine. If not specified, a default value is generated from the address of the default network interface. The machine id for each machine in the cluster can be used by cluster services to plan for failover by making sure that each member is backed up by a member running on a different machine.
`<address>`	Required	Specifies the IP address that a Socket will listen or publish on.**Note:** The localhost setting may not work on systems that define localhost as the loopback address; in that case, specify the machine name or the specific IP address. Default value is `localhost`. Preconfigured is `tangosol.coherence.localhost`. See Appendix L, "Command Line Overrides" for more information.
`<port>`	Required	Specifies the port that the Socket will listen or publish on. Legal values are from 1 to 65535. Default value is 8088. Preconfigured value is `tangosol.coherence.localport`. See Appendix L, "Command Line Overrides" for more information.

Table H–37 (Cont.) unicast-listener Subelements

Element	Required/ Optional	Description
<port-auto-adjust>	Required	Specifies whether the unicast port will be automatically incremented if the specified port cannot be bound to because it is already in use. Legal values are `true` or `false`. It is recommended that this value be configured to false for production environments. Default value is `true`. Preconfigured value is `tangosol.coherence.localport.adjust`. See Appendix L, "Command Line Overrides" for more information.
<packet-buffer>	Required	Specifies how many incoming packets the operating system will be requested to buffer.
<priority>	Required	Specifies a priority of the unicast listener execution thread. Legal values are from 1 to 10. Default value is `8`.
<ignore-socket-closed>	Optional	Specifies whether the unicast listener will ignore socket exceptions that indicate that a Member is unreachable. Deprecated as of Coherence 3.2.
<maximum-socket-closed-exceptions>	Optional	Specifies the maximum number of unicast listener exceptions that will be tolerated before a particular member is considered really gone and is removed from the cluster. Deprecated as of Coherence 3.2.

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "Element Attributes" on page H-53 for more information on this attribute.

volume-threshold

Used in: `packet-speaker`

Description

Specifies the minimum outgoing packet volume which must exist for the speaker daemon to be activated.

Performance Impact

When the packet load is relatively low it may be more efficient for the speaker's operations to be performed on the publisher's thread. When the packet load is high using the speaker allows the publisher to continue preparing packets while the speaker transmits them on the network.

Elements

Table H–38 describes the elements you can define within the `packet-speaker` element.

Table H–38 packet-speaker Subelements

Element	Required/ Optional	Description
`<minimum-packets>`	Required	Specifies the minimum number of packets which must be ready to be sent for the speaker daemon to be activated. A value of 0 will force the speaker to always be used, while a very high value will cause it to never be used. If unspecified, it will be set to match the `packet-buffer`, this is the default.

well-known-addresses

Used in: unicast-listener.

> **Note:** This is not a security-related feature, and does **not** limit the addresses which are allowed to join the cluster. See the authorized-hosts element for details on limiting cluster membership.
>
> Use of the Well Known Addresses (WKA) feature is not supported by Caching Edition. If you are having difficulties in establishing a cluster by using multicast, see Chapter 16, "Performing a Multicast Connectivity Test".

Description

By default, Coherence uses a multicast protocol to discover other nodes when forming a cluster. If multicast networking is undesirable, or unavailable in your environment, the Well Known Addresses feature may be used to eliminate the need for multicast traffic. When in use the cluster is configured with a relatively small list of nodes which are allowed to start the cluster, and which are likely to remain available over the cluster lifetime. There is no requirement for all WKA nodes to be simultaneously active at any point in time. This list is used by all other nodes to find their way into the cluster without the use of multicast, thus at least one well known node must be running for other nodes to be able to join.

Example

Example H–2 illustrates a configuration for two well-known-addresses with the default port.

Example H–2 Configuration for Two Well-Known-Addresses

```
<well-known-addresses>
  <socket-address id="1">
    <address>192.168.0.100</address>
    <port>8088</port>
  </socket-address>
  <socket-address id="2">
    <address>192.168.0.101</address>
    <port>8088</port>
  </socket-address>
</well-known-addresses>
```

Elements

Table H–39 describes the subelements you can define within the well-known-addresses element.

Table H–39 well-known-addresses Subelements

Element	Required/ Optional	Description
<socket-address>	Required	Specifies a list of "well known" addresses (WKA) that are used by the cluster discovery protocol in place of multicast broadcast. If one or more WKA is specified, for a member to join the cluster it will either have to be a WKA or there will have to be at least one WKA member running. Additionally, all cluster communication will be performed using unicast. If empty or unspecified multicast communications will be used. Preconfigured values are `tangosol.coherence.wka` and `tangosol.coherence.wka.port`. See Appendix L, "Command Line Overrides" for more information.

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "Element Attributes" on page H-53 for more information about this attribute.

Element Attributes

The optional `id` and `xml-override` attributes can be used to override the contents of an element. These attributes can appear, either individually or together, within the following elements:

Table H–40 lists the elements that can use `id` or `xml-override`, or both.

Table H–40 Elements that can use id or xml-override, or Both

`authorized-hosts`	`cluster-config`	`coherence`	`configurable-cache-factory-config`
`filter-name`	`filters`	`host-range`	`incoming-message-handler`
`init-param`	`logging-config`	`multicast-listener`	`packet-publisher`
services	`shutdown-listener`	`tcp-ring-listener`	`unicast-listener`

Table H–41 describes the functionality of the `id` and `xml-override` attributes.

Table H–41 id and xml-override Attribute Descriptions

Attribute	Required/ Optional	Description
`xml-override`	Optional	Allows the content of this element to be fully or partially overridden with XML documents that are external to the base document. Legal value of this attribute is the resource name of such an override document that should be accessible using the `ClassLoader.getResourceAsStream(String name)` by the classes contained in `coherence.jar` library. In general that means that resource name should be prefixed with '`/`' and located in the classpath. The override XML document referred by this attribute does not have to exist. However, if it does exist then its root element must have the same name as the element it overrides. In cases where there are multiple elements with the same name (for example, <`services`>) the `id` attribute should be used to identify the base element that will be overridden and the override element itself. The elements of the override document that do not have a match in the base document are just appended to the base.
`id`	Optional	Used in conjunction with the `xml-override` attribute in cases where there are multiple elements with the same name (for example, <`services`>) to identify the base element that will be overridden and the override element itself. The elements of the override document that do not have a match in the base document are just appended to the base.

Initialization Parameter Settings

The <init-param> element in the Coherence operational configuration deployment descriptor defines initialization parameters for a service or filter. The parameters that appear under init-param will be different, depending on the service or filter you are working with.

The following sections describe the parameters that can be configured for these services and filters:

- DistributedCache Service Parameters
- ReplicatedCache Service Parameters
- InvocationService Parameters
- ProxyService Parameters
- Compression Filter Parameters

The tables in each section describe the specific <param-name> — <param-value> pairs that can be configured for various elements. The **Parameter Name** column refers to the value of the param-name element and **Value Description** column refers to the possible values for the corresponding param-value element.

For example, the sample entry in Table I–1 means that the init-params element may look like the configuration in Example I–1 or Example I–2.

Table I–1 Sample Table Entry

Parameter Value	Value Description
local-storage	Specifies whether this member of the DistributedCache service enables the local storage. Legal values are true or false. Default value is true. Preconfigured value is tangosol.coherence.distributed.localstorage. See Appendix L, "Command Line Overrides" for more information.

Example I–1 Sample init-param Configuration

```
...
<init-params>
  <init-param>
    <param-name>local-storage</param-name>
    <param-value>false</param-value>
  </init-param>
</init-params>
...
```

or as follows:

Example I–2 Another Sample init-param Configuration

```
...
<init-params>
  <init-param>
    <param-name>local-storage</param-name>
    <param-value>true</param-value>
  </init-param>
</init-params>
...
```

DistributedCache Service Parameters

DistributedCache <services> elements support the parameters described in Table I–2. These settings may also be specified as part of the <distributed-scheme> element in the Cache Configuration Elements descriptor coherence-cache-config.xml.

Table I–2 DistributedCache Service Parameters

Parameter Name	Value, Description
backup-count	Specifies the number of members of the DistributedCache service that hold the backup data for each unit of storage in the cache. Value of 0 means that in the case of abnormal termination, some portion of the data in the cache will be lost. Value of N means that if up to N cluster nodes terminate immediately, the cache data will be preserved. To maintain the distributed cache of size M, the total memory usage in the cluster does not depend on the number of cluster nodes and will be in the order of M*(N+1). Recommended values are 0, 1 or 2. Default value is 1.
backup-storage/ class-name	Only applicable with the custom type. Specifies a class name for the custom storage implementation. If the class implements com.tangosol.run.xml.XmlConfigurable interface then upon construction the setConfig method is called passing the entire backup-storage element.
backup-storage/ directory	Only applicable with the file-mapped type. Specifies the path name for the directory that the disk persistence manager (com.tangosol.util.nio.MappedBufferManager) will use as "root" to store files in. If not specified or specifies a non-existent directory, a temporary file in the default location is used. Default value is the default temporary directory designated by the Java runtime.
backup-storage/ initial-size	Only applicable with the off-heap and file-mapped types.Specifies the initial buffer size in bytes.The value of this element must be in the following format: [\d]+[[.][\d]]?[K|k|M|m|G|g]?[B|b]? where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied: ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) If the value does not contain a factor, a factor of mega is assumed. Legal values are positive integers between 1 and Integer.MAX_VALUE - 1023 (that is, 2,147,482,624 bytes). Default value is 1MB.

Table I–2 (Cont.) DistributedCache Service Parameters

Parameter Name	Value, Description
backup-storage/ maximum-size	Only applicable with the off-heap and file-mapped types.Specifies the maximum buffer size in bytes.The value of this element must be in the following format: [\d]+[[.][\d]]?[K|k|M|m|G|g]?[B|b]? where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied: ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) If the value does not contain a factor, a factor of mega is assumed. Legal values are positive integers between 1 and Integer.MAX_VALUE - 1023 (that is, 2,147,482,624 bytes). Default value is 1024MB.
backup-storage/ scheme-name	Only applicable with the scheme type. Specifies a scheme name for the ConfigurableCacheFactory.
backup-storage/ type	Specifies the type of the storage used to hold the backup data. Legal values are: ■ on-heap—The corresponding implementations class is java.util.HashMap. ■ off-heap—The corresponding implementations class is com.tangosol.util.nio.BinaryMap using com.tangosol.util.nio.DirectBufferManager. Only available with JDK 1.4 and later. ■ file-mapped—The corresponding implementations class is com.tangosol.util.nio.BinaryMap using com.tangosol.util.nio.MappedBufferManager. Only available with JDK 1.4 and later. ■ custom—The corresponding implementations class is the class specified by the backup-storage/class element. ■ scheme—The corresponding implementations class is the map returned by the ConfigurableCacheFactory for the scheme referred to by the backup-storage/scheme-name element. Default value is on-heap. Preconfigured value is tangosol.coherence.distributed.backup. See Appendix L, "Command Line Overrides" for more information.
key-associator/ class-name	Specifies the name of a class that implements the com.tangosol.net.partition.KeyAssociator interface. This implementation must have a zero-parameter public constructor.
key-partitioning/ class-name	Specifies the name of a class that implements the com.tangosol.net.partition.KeyPartitioningStrategy interface. This implementation must have a zero-parameter public constructor.
lease-granularity	Specifies the lease ownership granularity. Available since release 2.3.Legal values are: ■ thread ■ member A value of thread means that locks are held by a thread that obtained them and can only be released by that thread. A value of member means that locks are held by a cluster node and any thread running on the cluster node that obtained the lock can release it. Default value is thread.

Table I–2 (Cont.) DistributedCache Service Parameters

Parameter Name	Value, Description
`local-storage`	Specifies whether this member of the DistributedCache service enables local storage. Normally this value should be left unspecified within the configuration file, and instead set on a per-process basis using the `tangosol.coherence.distributed.localstorage` system property. This allows cache clients and servers to use the same configuration descriptor. Legal values are `true` or `false`. Default value is `true`. Preconfigured value is `tangosol.coherence.distributed.localstorage`. See Appendix L, "Command Line Overrides" for more information.
`partition-count`	Specifies the number of partitions that a distributed cache will be "chopped up" into. Each member running the distributed cache service that has the `local-storage` option set to `true` will manage a "fair" (balanced) number of partitions. The number of partitions should be larger than the square of the number of cluster members to achieve a good balance, and it is suggested that the number be prime. Good defaults include 257 and 1021 and prime numbers in-between, depending on the expected cluster size. A list of first 1,000 primes can be found at http://www.utm.edu/research/primes/lists/small/1000.txt. Legal values are prime numbers. Default value is 257.
`partition-listener/ class-name`	Specifies the name of a class that implements the `com.tangosol.net.partition.PartitionListener` interface. This implementation must have a zero-parameter public constructor.
`request-timeout`	Specifies the maximum amount of time a client will wait for a response before abandoning the original request. The request time is measured on the client side as the time elapsed from the moment a request is sent for execution to the corresponding server node(s) and includes the following: ▪ the time it takes to deliver the request to an executing node (server) ▪ the interval between the time the task is received and placed into a service queue until the execution starts ▪ the task execution time ▪ the time it takes to deliver a result back to the client Legal values are positive integers or zero (indicating no default timeout).
`standard-lease- milliseconds`	Specifies the duration of the standard lease in milliseconds. When a lease has aged past this number of milliseconds, the lock will automatically be released. Set this value to zero to specify a lease that never expires. The purpose of this setting is to avoid deadlocks or blocks caused by stuck threads; the value should be set higher than the longest expected lock duration (for example, higher than a transaction timeout). It's also recommended to set this value higher then `packet-delivery/timeout-milliseconds` value. Legal values are from positive long numbers or zero. Default value is 0.
`task-hung-threshold`	Specifies the amount of time in milliseconds that a task can execute before it is considered "hung". **Note**: a posted task that has not yet started is never considered as hung. This attribute is applied only if the Thread pool is used (the `thread-count` value is positive). Legal values are positive integers or zero (indicating no default timeout).

Table I–2 (Cont.) DistributedCache Service Parameters

Parameter Name	Value, Description
`task-timeout`	Specifies the default timeout value in milliseconds for tasks that can be timed-out (for example, implement the `com.tangosol.net.PriorityTask` interface), but don't explicitly specify the task execution timeout value. The task execution time is measured on the server side and does not include the time spent waiting in a service backlog queue before being started. This attribute is applied only if the thread pool is used (the `thread-count` value is positive). Legal values are positive integers or zero (indicating no default timeout).
`thread-count`	Specifies the number of daemon threads used by the distributed cache service. If zero, all relevant tasks are performed on the service thread. Legal values are from positive integers or zero. Default value is 0. Preconfigured value is `tangosol.coherence.distributed.threads`. See Appendix L, "Command Line Overrides" for more information.
`transfer-threshold`	Specifies the threshold for the primary buckets distribution in kilobytes. When a new node joins the distributed cache service or when a member of the service leaves, the remaining nodes perform a task of bucket ownership re-distribution. During this process, the existing data gets rebalanced along with the ownership information. This parameter indicates a preferred message size for data transfer communications. Setting this value lower will make the distribution process take longer, but will reduce network bandwidth utilization during this activity. Legal values are integers greater then zero. Default value is 512 (0.5MB). Preconfigured value is `tangosol.coherence.distributed.transfer`. See Appendix L, "Command Line Overrides" for more information.

ReplicatedCache Service Parameters

ReplicatedCache `services` elements support the parameters described in Table I–3. These settings may also be specified as part of the `replicated-scheme` element in the Cache Configuration Elements descriptor `coherence-cache-config.xml`.

Table I–3 ReplicatedCache Service Parameters

Parameter Name	Value Description
`lease-granularity`	Specifies the lease ownership granularity. Available since release 2.3.Legal values are: ▪ `thread` ▪ `member` A value of `thread` means that locks are held by a thread that obtained them and can only be released by that thread. A value of `member` means that locks are held by a cluster node and any thread running on the cluster node that obtained the lock can release it. Default value is `thread`.
`mobile-issues`	Specifies whether lease issues should be transferred to the most recent lock holders. Legal values are `true` or `false`. Default value is `false`.
`standard-lease-milliseconds`	Specifies the duration of the standard lease in milliseconds. When a lease has aged past this number of milliseconds, the lock will automatically be released. Set this value to zero to specify a lease that never expires. The purpose of this setting is to avoid deadlocks or blocks caused by stuck threads; the value should be set higher than the longest expected lock duration (for example, higher than a transaction timeout). It's also recommended to set this value higher then packet-delivery/timeout-milliseconds value. Legal values are from positive long numbers or zero. Default value is 0.

InvocationService Parameters

`InvocationService` `services` elements support the following parameters listed in Table I–4. These settings may also be specified as part of the `invocation-scheme` element in the Cache Configuration Elements descriptor `coherence-cache-config.xml`.

Table I–4 InvocationService Parameters

Parameter Name	Value, Description
`request-timeout`	Specifies the default timeout value in milliseconds for requests that can time-out (for example, implement the `com.tangosol.net.PriorityTask` interface), but don't explicitly specify the request timeout value. The request time is measured on the client side as the time elapsed from the moment a request is sent for execution to the corresponding server node(s) and includes the following: ■ the time it takes to deliver the request to an executing node (server) ■ the interval between the time the task is received and placed into a service queue until the execution starts ■ the task execution time ■ the time it takes to deliver a result back to the client Legal values are positive integers or zero (indicating no default timeout).
`task-hung-threshold`	Specifies the amount of time in milliseconds that a task can execute before it is considered "hung". Note: a posted task that has not yet started is never considered as hung. This attribute is applied only if the Thread pool is used (the `thread-count` value is positive).
`task-timeout`	Specifies the default timeout value in milliseconds for tasks that can be timed-out (for example, implement the `com.tangosol.net.PriorityTask` interface), but don't explicitly specify the task execution timeout value. The task execution time is measured on the server side and does not include the time spent waiting in a service backlog queue before being started. This attribute is applied only if the thread pool is used (the `thread-count` value is positive). Legal values are positive integers or zero (indicating no default timeout).
`thread-count`	Specifies the number of daemon threads to be used by the invocation service. If zero, all relevant tasks are performed on the service thread. Legal values are from positive integers or zero. Preconfigured value is `tangosol.coherence.invocation.threads`. See Appendix L, "Command Line Overrides" for more information. Default value is 0.

ProxyService Parameters

ProxyService `services` elements support the parameters described in Table I–5. These settings may also be specified as part of the `proxy-scheme` element in the Cache Configuration Elements descriptor `coherence-cache-config.xml`.

Table I–5 ProxyService Parameters

Parameter Name	Value Description
`thread-count`	Specifies the number of daemon threads to be used by the proxy service. If zero, all relevant tasks are performed on the service thread. Legal values are from positive integers or zero. Default value is 0.

Compression Filter Parameters

The compression `filters` (`com.tangosol.net.CompressionFilter`), supports the parameters described in Table I–6 (see `java.util.zip.Deflater` for details).

Table I–6 Compression Filter Parameters

Parameter Name	Value Description
`buffer-length`	Specifies compression buffer length in bytes. Legal values are positive integers or zero. Default value is 0.
`level`	Specifies the compression level. Legal values are: ▪ `default` ▪ `compression` ▪ `speed` ▪ `none` Default value is default.
`strategy`	Specifies the compressions strategy. Legal values are: ▪ `gzip` ▪ `huffman-only` ▪ `filtered` ▪ `default` Default value is gzip.

J

POF User Type Configuration Elements

This appendix provides a listing of the elements that can be used to specify POF user types. POF user type configuration elements are defined in the `pof-config.dtd` file that can be found in the coherence.jar file.

You can find additional information about the POF user type configuration file in the Javadoc for the `ConfigurablePofContext` class.

POF User Type Deployment Descriptor

Use the POF user type deployment descriptor to specify the various user types which are being passed into the cluster.

Document Location

The name and location of the descriptor defaults to `pof-config.xml`. The default POF user type descriptor (packaged in `coherence.jar`) will be used unless a custom file is found within the application's classpath. The name and location of the descriptor can also be configured using system property `tangosol.pof.config`. It is recommended that all nodes within a cluster use identical POF user type descriptors.

Document Root

The root element of the POF user type descriptor is `pof-config`. This is where you may begin specifying your user types.

Document Format

The POF user type descriptor should begin with the following `DOCTYPE` declaration:

```
<!DOCTYPE pof-config SYSTEM "pof-config.dtd">
```

The format of the document and the nesting of elements is illustrated in Example J–1.

Example J–1 Format of a POF User Type Configuration File (pof-config.xml)

```
<pof-config>
   <user-type-list>
     ..
     <user-type>
       <type-id>53</type-id>
       <class-name>com.mycompany.data.Trade</class-name>
       <serializer>
         <class-name>com.tangosol.io.pof.PortableObjectSerializer</class-name>
         <init-params>
```

```
        <init-param>
          <param-type>int</param-type>
          <param-value>{type-id}</param-value>
        </init-param>
      </init-params>
    </serializer>
  </user-type>

  <user-type>
    <type-id>54</type-id>
    <class-name>com.mycompany.data.Position</class-name>
  </user-type>

  ..
  <include>file:/my-pof-config.xml</include>

  ..
 </user-type-list>

 <allow-interfaces>false</allow-interfaces>
 <allow-subclasses>false</allow-subclasses>
</pof-config>
```

Command Line Override

Oracle Coherence provides a powerful Command Line Setting Override Feature, which allows any element defined in this descriptor to be overridden from the Java command line if it has a system-property attribute defined in the descriptor.

Element Index

Table J–1 lists all elements which may be used from within a POF user type configuration.

Table J–1 POF Configuration Elements

Element	Used In:
<allow-interfaces>	<pof-config>
<allow-subclasses>	<pof-config>
<class-name>	<user-type>, <serializer>
<include>	<user-type-list>
<init-param>	<init-params>
<init-params>	<serializer>
<param-type>	<init-param>
<param-value>	<init-param>
<pof-config>	*root element*
<serializer>	<user-type>
<type-id>	<user-type>
<user-type>	<user-type-list>
<user-type-list>	<pof-config>

allow-interfaces

Used in: <pof-config>

Description

The `allow-interfaces` element indicates whether the `user-type` `class-name` can specify Java interface types in addition to Java class types.

Valid values are `true` or `false`. Default value is `false`.

Elements

Terminal element.

allow-subclasses

Used in: <pof-config>

Description

The `allow-subclasses` element indicates whether the `user-type` `class-name` can specify a Java class type that is abstract, and whether sub-classes of any specified `user-type` `class-name` will be permitted at runtime and automatically mapped to the specified super-class for purposes of obtaining a serializer.

Valid values are `true` or `false`. Default value is `false`.

Elements

Terminal element.

class-name

Used in: <user-type>, <serializer>

Description

The `class-name` element specifies the name of a Java class or interface.

Within the `user-type` element, the `class-name` element is required, and specifies the fully qualified name of the Java class or interface that all values of the user type are type-assignable to.

Within the `serializer` element, the `class-name` element is required.

Elements

Terminal element.

include

Used in: <user-type-list>

Description

The include element specifies the location of a `pof-config` file to load `user-type` elements from. The value is a locator string (either a valid path or URL) that identifies the location of the target `pof-config` file.

Elements

Terminal element.

init-param

Used in: <init-params>

Description

The init-param element provides a type for a configuration parameter and a corresponding value to pass as an argument.

Elements

Table J–2 describes the subelements you can define within the init-param element.

Table J–2 init-param Subelements

Element	Required/ Optional	Description
<param-type>	Required	The param-type element specifies the Java type of initialization parameter. Supported types are: ■ string—indicates that the value is a java.lang.String ■ boolean—indicates that the value is a java.lang.Boolean ■ int—indicates that the value is a java.lang.Integer ■ long—indicates that the value is a java.lang.Long ■ double—indicates that the value is a java.lang.Double ■ decimal—indicates that the value is a java.math.BigDecimal ■ file—indicates that the value is a java.io.File ■ date— indicates that the value is a java.sql.Date ■ time—indicates that the value is a java.sql.Timedatetime ■ datetime—indicates that the value is a java.sql.Timestamp ■ xml—indicates that the value is the entire init-param XmlElement. The value is converted to the specified type, and the target constructor or method must have a parameter of that type for the instantiation to succeed.
<param-value>	Required	The param-value element specifies a value of the initialization parameter. The value is in a format specific to the type of the parameter. There are four reserved values that can be specified. Each of these values is replaced at runtime with a specific runtime value before the constructor is invoked: ■ {type-id}—replaced with the Type ID of the User Type; ■ {class-name}—replaced with the name of the class for the User Type; ■ {class}—replaced with the Class for the User Type; ■ {class-loader}—replaced with the ConfigurablePofContext's ContextClassLoader.

init-params

Used in: <serializer>

Description

The init-params element contains zero or more arguments (each as an init-param) that correspond to the parameters of a constructor of the class that is being configured.

Elements

Table J–3 describes the elements you can define within the init-params element.

Table J–3 init-params Subelements

Element	Required/ Optional	Description
<init-param>	Required	The init-param element provides a type for a configuration parameter and a corresponding value to pass as an argument.

param-type

Used in: <init-param>

Description

The param-type element specifies the Java type of initialization parameter.

Supported types are:

- string—indicates that the value is a java.lang.String
- boolean—indicates that the value is a java.lang.Boolean
- int—indicates that the value is a java.lang.Integer
- long—indicates that the value is a java.lang.Long
- double—indicates that the value is a java.lang.Double
- decimal—indicates that the value is a java.math.BigDecimal
- file—indicates that the value is a java.io.File
- date— indicates that the value is a java.sql.Date
- time—indicates that the value is a java.sql.Timedatetime
- datetime—indicates that the value is a java.sql.Timestamp
- xml—indicates that the value is the entire init-param XmlElement.

The value is converted to the specified type, and the target constructor or method must have a parameter of that type in order for the instantiation to succeed.

Elements

Terminal element.

param-value

Used in: <init-param>

Description

The param-value element specifies a value of the initialization parameter. The value is in a format specific to the type of the parameter.

There are four reserved values that can be specified. Each of these values is replaced at runtime with a specific runtime value before the constructor is invoked:

- {type-id}—replaced with the Type ID of the User Type;
- {class-name}—replaced with the name of the class for the User Type;
- {class}—replaced with the Class for the User Type;
- {class-loader}—replaced with the ConfigurablePofContext's ContextClassLoader.

Elements

Terminal element.

pof-config

root element

Description

The `pof-config` element is the root element of the POF user type configuration descriptor.

Elements

Table J–4 describes the elements you can define within the `pof-config` element.

Table J–4 pof-config Subelements

Element	Required/ Optional	Description
<`allow-interfaces`>	Optional	The `allow-interfaces` element indicates whether the user-type `class-name` can specify Java interface types in addition to Java class types. Valid values are `true` or `false`. Default value is `false`.
<`allow-subclasses`>	Optional	The `allow-subclasses` element indicates whether the user-type `class-name` can specify a Java class type that is abstract, and whether sub-classes of any specified user-type `class-name` will be permitted at runtime and automatically mapped to the specified super-class for purposes of obtaining a serializer. Valid values are `true` or `false`. Default value is `false`.
<`user-type-list`>	Required	The `user-type-list` element contains zero or more user-type elements. Each POF user type that will be used must be listed in the `user-type-list`. The `user-type-list` element may also contain zero or more `include` elements. Each include element is used to add user-type elements defined in another `pof-config` file.

serializer

Used in: <user-type>

Description

The serializer element specifies what PofSerializer to use to serialize and deserialize a specific user type.

A PofSerializer is used to serialize and deserialize user type values to and from a POF stream. Within the serializer element, the class-name element is required, and zero or more constructor parameters can be defined within an init-params element.

If the serializer element is omitted, then the user type is assumed to implement the PortableObject interface, and the PortableObjectSerializer implementation is used as the PofSerializer.

If the init-params element is omitted from the serializer element, then the following four constructors are attempted on the specific PofSerializer implementation, in this order:

- (int nTypeId, Class clz, ClassLoader loader)
- (int nTypeId, Class clz)
- (int nTypeId)
- ()

Elements

Table J–5 describes the elements you can define within the serializer element.

Table J–5 serializer Subelements

Element	Required/ Optional	Description
<class-name>	Required	Specifies the name of the serializer.
<init-params>	Optional	The init-params element contains zero or more arguments (each as an init-param) that correspond to the parameters of a constructor of the class that is being configured.

type-id

Used in: <user-type>

Description

The type-id element specifies an integer value (n >= 0) that uniquely identifies the user type.

If none of the user-type elements contains a type-id element, then the type IDs for the user types will be based on the order in which they appear in the user-type-list, with the first user type being assigned the type ID 0, the second user type being assigned the type ID 1, and so on.

However, it is strongly recommended that user types IDs always be specified, to support schema versioning and evolution.

Note: Reserved IDs: The first 1000 IDs are reserved for Coherence internal use.

Elements

Terminal element.

user-type

Used in: <user-type-list>

Description

The user-type element contains the declaration of a POF user type. A POF user type is a uniquely identifiable, portable, versionable object class that can be communicated among systems regardless of language, operating system, hardware and location.

Within the user-type element, the type-id element is optional, but its use is strongly suggested to support schema versioning and evolution.

Within the user-type element, the class-name element is required, and specifies the fully qualified name of the Java class or interface that all values of the user type are type-assignable to.

If the serializer element is omitted, then the user type is assumed to implement the PortableObject interface, and the PortableObjectSerializer implementation is used as the PofSerializer.

Elements

Table J–6 describes the elements you can define within the user-type element.

Table J–6 user-type Subelements

Element	Required/ Optional	Description
<class-name>	Required	The class-name element specifies the name of a Java class or interface. Within the user-type element, the class-name element is required, and specifies the fully qualified name of the Java class or interface that all values of the user type are type-assignable to. Within the serializer element, the class-name element is required.
<serializer>	Optional	The serializer element specifies what PofSerializer to use to serialize and deserialize a specific user type. A PofSerializer is used to serialize and deserialize user type values to and from a POF stream. Within the serializer element, the class-name element is required, and zero or more constructor parameters can be defined within an init-params element. If the serializer element is omitted, then the user type is assumed to implement the PortableObject interface, and the PortableObjectSerializer implementation is used as the PofSerializer. If the init-params element is omitted from the serializer element, then the following four constructors are attempted on the specific PofSerializer implementation, and in this order: ▪ (int nTypeId, Class clz, ClassLoader loader) ▪ (int nTypeId, Class clz) ▪ (int nTypeId) ▪ ()
<type-id>	Optional	The type-id element specifies an integer value (n >= 0) that uniquely identifies the user type. If none of the user-type elements contains a type-id element, then the type IDs for the user types will be based on the order in which they appear in the user-type-list, with the first user type being assigned the type ID 0, the second user type being assigned the type ID 1, and so on. However, it is strongly recommended that user types IDs always be specified, to support schema versioning and evolution.

user-type-list

Used in: <pof-config>

Description

The user-type-list element contains zero or more user-type elements. Each POF user type that will be used must be listed in the user-type-list.

The user-type-list element may also contain zero or more include elements. Each include element is used to add user-type elements defined in another pof-config file.

Elements

The following table describes the elements you can define within the user-type-list element.

Table J–7 user-type-list Subelements

Element	Required/ Optional	Description
<include>	Required	The include element specifies the location of a pof-config file to load user-type elements from. The value is a locator string (either a valid path or URL) that identifies the location of the target pof-config file.
<user-type>	Required	The user-type element contains the declaration of a POF user type. A POF user type is a uniquely identifiable, portable, versionable object class that can be communicated among systems regardless of language, operating system, hardware and location. Within the user-type element, the type-id element is optional, but its use is strongly suggested to support schema versioning and evolution. Within the user-type element, the class-name element is required, and specifies the fully qualified name of the Java class or interface that all values of the user type are type-assignable to. If the serializer element is omitted, then the user type is assumed to implement the PortableObject interface, and the PortableObjectSerializer implementation is used as the PofSerializer.

K

MBean Configuration Elements

This appendix provides a description of the elements that can be used to configure MBeans.

MBeans in the Coherence Deployment Descriptor

The MBean configuration elements are defined in the `coherence.dtd` XML file, which is packaged in `coherence.jar`.

Document Root

The root element of the POF user type descriptor is mbeans. This is where you may begin configuring your MBean.

Document Format

The format and nesting of the MBean configuration elements is illustrated in Example K–1.

Example K–1 Format and Nesting of MBean Configuration Elements

```
<mbeans>
  <mbean>
    <mbean-class>
    <mbean-factory>
    <mbean-query>
    <mbean-accessor>
    <mbean-name>
    <enabled>
    <extend-lifecycle>
  </mbean>
</mbeans>
```

MBean Configuration Element Index

Table K–1 MBean Configuration Element Index

Element	Used In:
extend-lifecycle	mbean
enabled	mbean
mbean	mbeans
mbean-accessor	mbean
mbean-class	mbean
mbean-factory	mbean
mbean-name	mbean
mbean-query	mbean
mbeans	*root element*

extend-lifecycle

Used in: mbean

Description

Specifies if the MBean should extend beyond the node connection life cycle. If `false`, the MBean will be destroyed and re-created when a node is disconnected from the grid. If `true`, the MBean will maintain the statistics and values across connections

Example

```
<extend-lifecycle>true</extend-lifecycle>
```

enabled

Used in: mbean

Description

The `enabled` element specifies either `true` if the MBean should be instantiated or `false` if the MBean should be ignored.

Example

```
<enabled>true</enabled>
```

mbean

Used in: mbeans.

Description

The mbean element contains a list of elements to be instantiated and registered with the Coherence Management infrastructure.

Elements

Table K–2 describes the subelements you can define within the mbeans element.

Table K–2 Subelements of mbean

Element	Required/ Optional	Description
<mbean-class>	Optional	Specifies the class of the standard MBean to instantiate.
<mbean-factory>	Optional	Specifies the class of the factory used to instantiate the MBean
<mbean-query>	Optional	Specifies the JMX ObjectName query pattern used to retrieve the MBeans
<mbean-accessor>	Optional	Specifies the accessor method on the <mbean-factory> used to instantiate the MBean.
<mbean-name>	Required	Specifies the ObjectName prefix for the MBean.
<enabled>	Required	Specifies if the MBean should be registered on this instance.
<extend-lifecycle>	Optional	Specifies if the MBean should extend beyond the node connection life cycle.

mbean-accessor

Used in: mbean

Description

The `mbean-accessor` element contains the accessor method name on the MBean Factory that instantiates the MXBean. The `MBean` factory class must be in the class path to correctly instantiate. The `<mbean-accessor>` element requires an `mbean-factory` element.

Example

```
<mbean-factory>java.lang.management.ManagementFactory</mbean-factory>
<mbean-accessor>getMemoryMXBean</mbean-accessor>
```

mbean-class

Used in: mbean

Description

The `mbean-class` element contains the class name of a standard MBean. The MBean class must be in the class path to correctly instantiate.

Example

```
<mbean-class>com.oracle.custom.mbeans.query</mbean-class>
```

mbean-factory

Used in: mbean

Description

The mbean-factory element contains the class name of a MBean factory that instantiates MXBeans. The MBean factory class must be in the class path to correctly instantiate. The <mbean-factory> element requires an mbean-accessor element.

Example

```
<mbean-factory>java.lang.management.ManagementFactory</mbean-factory>
<mbean-accessor>getMemoryMXBean</mbean-accessor>
```

mbean-name

Used in: mbean

Description

The mbean-name element contains the ObjectName prefix for the resulting MBeans. This ObjectName prefix should be a comma-separated *Key*=*Value* pair. The Coherence MBean naming convention stipulates that the name should begin with a "type"/"value" pair (that is, type=Platform)

Example

To prefix the custom mbeans with type=Platform:

```
<mbean-name>type=Platform</mbean-name>
```

mbean-query

Used in: mbean

Description

The mbean-query element contains a JMX ObjectName query pattern. The query pattern is executed against a local MBean Server and the resulting objects are registered with the Coherence Management infrastructure. This allows the for a single point of consolidation of MBeans for the grid.

Example

```
<mbean-query>java.lang:*</mbean-query>
```

Will include all the MBeans under the java.lang domain in the Coherence management infrastructure.

Notes

- A local MBean Server must be enabled.

mbeans

Used in: *root element*

Description

The `mbeans` element is the root element of the custom mbean configuration file. It contains a list of mbean elements to be instantiated and registered with the coherence management infrastructure.

Elements

Table K–3 describes the elements you can define within the `mbeans` element.

Table K–3 Subelement of mbeans

Element	Required/ Optional	Description
<`mbean`>	Required	Specifies the MBean type, implementation, and `ObjectName` that will be instantiated and registered with the Coherence Management service.

L

Command Line Overrides

Both the Coherence Operational Configuration deployment descriptor `tangosol-coherence.xml` and the Coherence Cache Configuration deployment descriptor `coherence-cache-config.xml` can assign a Java command line option name to any element defined in the descriptor. Some elements already have these Command Line Setting Overrides defined. You can create your own or change the predefined ones.

This feature is useful when you need to change the settings for a single JVM, or to be able to start different applications with different settings without making them use different descriptors. The most common application is passing a different multicast address and/or port to allow different applications to create separate clusters.

To create a Command Line Setting Override, add a `system-property` attribute, specifying the string you would like to assign as the name for the java command line option to the element you want to create an override to. Then, specify it in the Java command line, prepended with "`-D`".

Override Example

For example, to create an override for the IP address of the multi-home server to avoid using the default `localhost`, and instead specify a specific the IP address of the interface we want Coherence to use (for instance, 192.168.0.301). We would like to call this override `tangosol.coherence.localhost`.

First, add a system-property to the `cluster-config`, `unicast-listener`, or `address` element:

```
<address>localhost</address>
```

which will look as follows with the property we added:

```
<address system-property="tangosol.coherence.localhost">localhost</address>
```

Then use it by modifying the Java command line:

```
java -jar coherence.jar
```

Specify the IP address, `192.168.0.301` (instead of the default `localhost` specified in the configuration) as follows:

```
java -Dtangosol.coherence.localhost=192.168.0.301 -jar coherence.jar
```

Preconfigured Override Values

Table L–1 lists all of the preconfigured override values:

Table L–1 Preconfigured System Property Override Values

Override Option	Setting
`tangosol.coherence.cacheconfig`	Cache configuration descriptor filename. See "configurable-cache-factory-config" on page H-11.
`tangosol.coherence.cluster`	Cluster name. See "member-identity" on page H-24.
`tangosol.coherence.clusteraddress`	Cluster (multicast) IP address. See `<multicast-listener-address>` subelement of "multicast-listener" on page H-26
`tangosol.coherence.clusterport`	Cluster (multicast) IP port. See `<multicast-listener-port>` subelement of "multicast-listener" on page H-26.
`tangosol.coherence.distributed.backup`	Data backup storage location. See `backup-storage/type` subelement in "DistributedCache Service Parameters" on page I-3.
`tangosol.coherence.distributed.backupcount`	Number of data backups. See `backup-count` subelement in "DistributedCache Service Parameters" on page I-3.
`tangosol.coherence.distributed.localstorage`	Local partition management enabled. See local-storage subelement in "DistributedCache Service Parameters" on page I-3.
`tangosol.coherence.distributed.threads`	Thread pool size. See `thread-count` subelement in "DistributedCache Service Parameters" on page I-3.
`tangosol.coherence.distributed.transfer`	Partition transfer threshold. See `transfer-threshold` subelement in"DistributedCache Service Parameters" on page I-3.
`tangosol.coherence.edition`	Product edition. See "license-config" on page H-18.
`tangosol.coherence.invocation.threads`	Invocation service thread pool size. See `thread-count` subelement in "InvocationService Parameters" on page I-8.
`tangosol.coherence.localhost`	Unicast IP address. See `<unicast-listener-address>` subelement in "unicast-listener" on page H-48.
`tangosol.coherence.localport`	Unicast IP port. See `<unicast-listener-port>` subelement in "unicast-listener" on page H-48.
`tangosol.coherence.localport.adjust`	Unicast IP port auto assignment. See `<unicast-listener-auto>` subelement in "unicast-listener" on page H-48.
`tangosol.coherence.log`	Logging destination. See `<logging-config-destination>` subelement in "logging-config" on page H-19.
`tangosol.coherence.log.level`	Logging level. See `<logging-config-level>` subelement in "logging-config" on page H-19.
`tangosol.coherence.log.limit`	Log output character limit. See `<logging-config-limit>` subelement in "logging-config" on page H-19.
`tangosol.coherence.machine`	Machine name. See "member-identity" on page H-24.

Table L–1 (Cont.) Preconfigured System Property Override Values

Override Option	Setting
`tangosol.coherence.management`	JMX management mode. See "management-config" on page H-23.
`tangosol.coherence.management.readonly`	JMX management read-only flag. "management-config" on page H-23.
`tangosol.coherence.management.remote`	Remote JMX management enabled flag. See"management-config" on page H-23.
`tangosol.coherence.member`	Member name. See "member-identity" on page H-24.
`tangosol.coherence.mode`	Operational mode. See "license-config" on page H-18.
`tangosol.coherence.override`	Deployment configuration override filename.
`tangosol.coherence.priority`	Priority. See "member-identity" on page H-24.
`tangosol.coherence.process`	Process name"member-identity" on page H-24.
`tangosol.coherence.proxy.threads`	Coherence*Extend service thread pool size. See `thread-count` subelement in "ProxyService Parameters" on page I-9.
`tangosol.coherence.rack`	Rack name. See "member-identity" on page H-24.
`tangosol.coherence.role`	Role name. See "member-identity" on page H-24.
`tangosol.coherence.security`	Cache access security enabled flag. See "security-config" on page H-41.
`tangosol.coherence.security.keystore`	Security access controller keystore file name. See "security-config" on page H-41.
`tangosol.coherence.security.password`	Keystore or cluster encryption password. "Encryption Filters" on page 8-1.
`tangosol.coherence.security.permissions`	Security access controller permissions file name. See "security-config" on page H-41.
`tangosol.coherence.shutdownhook`	Shutdown listener action. See "shutdown-listener" on page H-44.
`tangosol.coherence.site`	Site name. See "member-identity" on page H-24.
`tangosol.coherence.tcmp.enabled`	TCMP enabled flag. See `<packet-publisher-enabled>` subelement in "packet-publisher" on page H-36.
`tangosol.coherence.tcpring`	!TCP ring enabled flag. See "tcp-ring-listener" on page H-46.
`tangosol.coherence.ttl`	Multicast packet time to live (TTL). See `<mulitcast-listener-ttl>` subelement in "multicast-listener" on page H-26.
`tangosol.coherence.wka`	Well known IP address. See "well-known-addresses" on page H-51.
`tangosol.coherence.wka.port`	Well known IP port. See "well-known-addresses" on page H-51.

M

Platform-Specific Deployment Considerations

The following sections in this appendix provide information on deploying Coherence to various platforms.

Deploying to AIX

When deploying Coherence on AIX please be aware of the following:

Socket Buffers sizes and JVMs

There is an issue with IBM's 1.4.2, and 1.5 JVMs for AIX which may prevent them from allocating socket buffers larger then 64K (Oracle 2MB). This issue has been addressed in IBM's 1.4.2 SR7 SDK and 1.5 SR3 SDK. See "Operating System Tuning" on page 20-1.

Multicast and IPv6

AIX 5.2 and above default to running multicast over IPv6 rather then IPv4. If you run in a mixed IPv6/IPv4 environment you will need to configure your JVMs to explicitly use IPv4. This can be done by setting the `java.net.preferIPv4Stack` system property to true on the Java command line. See the IBM 32-bit SDK for AIX User Guide for details.

Unique Multicast Addresses and Ports

On AIX it is suggested that each Coherence cluster use a unique multicast address and port, as some versions of AIX will not take both into account when delivering packets. See "multicast-listener" on page H-26 for details on configuring the address.

Deploying to BEA JRockit JVMs

When deploying Coherence on JRockit JVMs please be aware of the following:

JRockit and the Native Posix Thread Library (NPTL)

When running JRockit on Linux, BEA recommends using 2.6 kernels, and ensuring that the NPTL is enabled. Please see BEA's documentation regarding this issue.

AtomicLong

When available Coherence will make use of the highly concurrent AtomicLong class, which allows concurrent atomic updates to long values without requiring synchronization. BEA 1.4 JVMs do not fully support AtomicLong and thus if Coherence detects that it is being run on a BEA 1.4 JVM it will default to a safe but slower synchronized implementation, and will output the following log message.

```
sun.misc.AtomicLong is not supported on this JVM; using a synchronized counter.
```

Upgrading to JRockit 1.5 will allow the use of the highly concurrent implementation.

Deploying to Cisco Switches

When deploying Coherence with Cisco switches please be aware of the following:

Buffer Space and Packet Pauses

Under heavy UDP packet load some Cisco switches may run out of buffer space and exhibit frequent multi-second communication pauses. These communication pauses can be identified by a series of Coherence log messages referencing communication delays with multiple nodes which cannot be attributed to local or remote GCs.

Experienced a **4172 ms** communication delay (probable remote GC) with `Member(Id=7, Timestamp=2008-09-15 12:15:47.511, Address=xxx.xxx.x.xx:8089, MachineId=13838); 320 packets rescheduled, PauseRate=0.31, Threshold=512`

The Cisco 6500 series support configuration the amount of buffer space available to each Ethernet port or ASIC. In high load applications it may be necessary to increase the default buffer space. This can be accomplished by executing:

```
fabric buffer-reserve high
```

See Cisco's documentation for additional details on this setting.

Multicast Connectivity on Large Networks

Cisco's default switch configuration does not support proper routing of multicast packets between switches due to the use of IGMP snooping. See the Cisco's documentation regarding the issue and solutions.

Multicast Outages

Some Cisco switches have shown difficulty in maintaining multicast group membership resulting in existing multicast group members being silently removed from the multicast group. This will cause a partial communication disconnect for the associated Coherence node(s) and they will be forced to leave and rejoin the cluster. This type of outage can most often be identified by the following Coherence log messages indicating that a partial communication problem has been detected.

A potential network configuration problem has been detected. A packet has failed to be delivered (or acknowledged) after 60 seconds, although other packets were acknowledged by the same cluster member (`Member(Id=3, Timestamp=Sat Sept 13 12:02:54 EST 2008, Address=xxx.xxx.x.xxx, Port=8088, MachineId=48991))` to this member `(Member(Id=1, Timestamp=Sat Sept 13 11:51:11 EST 2008, Address=xxx.xxx.x.xxx, Port=8088, MachineId=49002))` as recently as 5 seconds ago.

To confirm the issue you may run the using the same multicast address and port as the running cluster. If the issue affects a multicast test node its logs will show that at some point it will suddenly stop receiving multicast test messages. See Chapter 16, "Performing a Multicast Connectivity Test".

The following test logs show the issue:

Example M–1 Log for a Multicast Outage

```
Test Node 192.168.1.100:
Sun Sept 14 16:44:22 GMT 2008: Received 83 bytes from a Coherence cluster node at
182.168.1.100: ??? Sun Sept 14 16:44:23 GMT 2008: Received test packet 76 from
ip=/192.168.1.101, group=/224.3.2.0:32367, ttl=4. Sun Sept 14 16:44:23 GMT 2008:
Received 83 bytes from a Coherence cluster node at 182.168.1.100: ??? Sun Sept 14
16:44:23 GMT 2008: Sent packet 85. Sun Sept 14 16:44:23 GMT 2008: Received test
packet 85 from self. Sun Sept 14 16:44:24 GMT 2008: Received 83 bytes from a
Coherence cluster node at 182.168.1.100: ??? Sun Sept 14 16:44:25 GMT 2008:
Received test packet 77 from ip=/192.168.1.101, group=/224.3.2.0:32367, ttl=4. Sun
Sept 14 16:44:25 GMT 2008: Received 83 bytes from a Coherence cluster node at
182.168.1.100: ??? Sun Sept 14 16:44:25 GMT 2008: Sent packet 86. Sun Sept 14
16:44:25 GMT 2008: Received test packet 86 from self. Sun Sept 14 16:44:26 GMT
2008: Received 83 bytes from a Coherence cluster node at 182.168.1.100: ??? Sun
Sept 14 16:44:27 GMT 2008: Received test packet 78 from ip=/192.168.1.101,
group=/224.3.2.0:32367, ttl=4. Sun Sept 14 16:44:27 GMT 2008: Received 83 bytes
from a Coherence cluster node at 182.168.1.100: ??? Sun Sept 14 16:44:27 GMT 2008:
Sent packet 87. Sun Sept 14 16:44:27 GMT 2008: Received test packet 87 from self.
Sun Sept 14 16:44:28 GMT 2008: Received 83 bytes from a Coherence cluster node at
182.168.1.100: ??? Sun Sept 14 16:44:29 GMT 2008: Received 83 bytes from a
Coherence cluster node at 182.168.1.100: ??? Sun Sept 14 16:44:29 GMT 2008: Sent
packet 88. Sun Sept 14 16:44:29 GMT 2008: Received test packet 88 from self. Sun
Sept 14 16:44:30 GMT 2008: Received 83 bytes from a Coherence cluster node at
182.168.1.100: ??? Sun Sept 14 16:44:31 GMT 2008: Received 83 bytes from a
Coherence cluster node at 182.168.1.100: ??? Sun Sept 14 16:44:31 GMT 2008: Sent
packet 89. Sun Sept 14 16:44:31 GMT 2008: Received test packet 89 from self. Sun
Sept 14 16:44:32 GMT 2008: Received 83 bytes from a Coherence cluster node at
182.168.1.100: ??? Sun Sept 14 16:44:33 GMT 2008: Received 83 bytes from a
Coherence cluster node at 182.168.1.100: ???
Test Node 192.168.1.101:
Sun Sept 14 16:44:22 GMT 2008: Sent packet 76.Sun Sept 14 16:44:22 GMT 2008:
Received test packet 76 from self. Sun Sept 14 16:44:22 GMT 2008: Received 83
bytes from a Coherence cluster node at 192.168.1.100: ??? Sun Sept 14 16:44:22 GMT
2008: Received test packet 85 from ip=/192.168.1.100, group=/224.3.2.0:32367,
ttl=4. Sun Sept 14 16:44:23 GMT 2008: Received 83 bytes from a Coherence cluster
node at 192.168.1.100: ??? Sun Sept 14 16:44:24 GMT 2008: Sent packet 77.Sun Sept
```

```
14 16:44:24 GMT 2008: Received test packet 77 from self. Sun Sept 14 16:44:24 GMT
2008: Received 83 bytes from a Coherence cluster node at 192.168.1.100: ??? Sun
Sept 14 16:44:24 GMT 2008: Received test packet 86 from ip=/192.168.1.100,
group=/224.3.2.0:32367, ttl=4. Sun Sept 14 16:44:25 GMT 2008: Received 83 bytes
from a Coherence cluster node at 192.168.1.100: ??? Sun Sept 14 16:44:26 GMT 2008:
Sent packet 78.Sun Sept 14 16:44:26 GMT 2008: Received test packet 78 from self.
Sun Sept 14 16:44:26 GMT 2008: Received 83 bytes from a Coherence cluster node at
192.168.1.100: ??? Sun Sept 14 16:44:26 GMT 2008: Received test packet 87 from
ip=/192.168.1.100, group=/224.3.2.0:32367, ttl=4. Sun Sept 14 16:44:27 GMT 2008:
Received 83 bytes from a Coherence cluster node at 192.168.1.100: ??? Sun Sept 14
16:44:28 GMT 2008: Sent packet 79.Sun Sept 14 16:44:28 GMT 2008: Received test
packet 79 from self. Sun Sept 14 16:44:28 GMT 2008: Received 83 bytes from a
Coherence cluster node at 192.168.1.100: ??? Sun Sept 14 16:44:28 GMT 2008:
Received test packet 88 from ip=/192.168.1.100, group=/224.3.2.0:32367, ttl=4. Sun
Sept 14 16:44:29 GMT 2008: Received 83 bytes from a Coherence cluster node at
192.168.1.100: ??? Sun Sept 14 16:44:30 GMT 2008: Sent packet 80.Sun Sept 14
16:44:30 GMT 2008: Received test packet 80 from self. Sun Sept 14 16:44:30 GMT
2008: Received 83 bytes from a Coherence cluster node at 192.168.1.100: ??? Sun
Sept 14 16:44:30 GMT 2008: Received test packet 89 from ip=/192.168.1.100,
group=/224.3.2.0:32367, ttl=4. Sun Sept 14 16:44:31 GMT 2008: Received 83 bytes
from a Coherence cluster node at 192.168.1.100: ??? Sun Sept 14 16:44:32 GMT 2008:
Sent packet 81.Sun Sept 14 16:44:32 GMT 2008: Received test packet 81 from self.
Sun Sept 14 16:44:32 GMT 2008: Received 83 bytes from a Coherence cluster node at
192.168.1.100: ??? Sun Sept 14 16:44:32 GMT 2008: Received test packet 90 from
ip=/192.168.1.100, group=/224.3.2.0:32367, ttl=4. Sun Sept 14 16:44:33 GMT 2008:
Received 83 bytes from a Coherence cluster node at 192.168.1.100: ??? Sun Sept 14
16:44:34 GMT 2008: Sent packet 82.
```

Note that at 16:44:27 the first test node stops receiving multicast packets from other machines. The operating system continues to properly forward multicast traffic from other processes on the same machine, but the test packets (79 and higher) from the second test node are not received. Also note that both the test packets and the cluster's multicast traffic generated by the first node do continue to be delivered to the second node. This indicates that the first node was silently removed from the multicast group.

If you encounter this multicast issue it is suggested that you contact Cisco technical support, or you may consider changing your configuration to unicast-only by using the Coherence well-known-addresses feature. See "well-known-addresses" on page H-51.

Deploying to Foundry Switches

When deploying Coherence with Foundry switches please be aware of the following:

Multicast Connectivity

Foundry switches have shown to exhibit difficulty in handing multicast traffic. When deploying on with Foundry switches it is recommend that you use the to ensure that all machines which will be part of the Coherence cluster can communicate over multicast. See Chapter 16, "Performing a Multicast Connectivity Test".

If you encounter issues with multicast you may consider changing your configuration to unicast-only by using the well-known-addresses feature. See "well-known-addresses" on page H-51.

Deploying to IBM BladeCenters

When deploying Coherence on IBM BladeCenters please be aware of the following:

MAC Address Uniformity and Load Balancing

A typical deployment on a BladeCenter may include blades with two NICs where one is used for administration purposes and the other for cluster traffic. By default the MAC addresses assigned to the blades of a BladeCenter are uniform enough that the first NIC will generally have an even MAC address and the second will have an odd MAC address. If the BladeCenter's uplink to a central switch also has an even number of channels then layer 2 (MAC based) load balancing may prevent one set of NICs from making full use of the available uplink bandwidth as they will all be bound to either even or odd channels. This issue arises due to the assumption in the switch that MAC addresses are essentially random, which in BladeCenter's is untrue. Remedies to this situation include:

- Use layer 3 (IP based) load balancing, assuming that the IP addresses do not follow the same even/odd pattern.
  - This setting would need to be applied across all switches carrying cluster traffic.
- Randomize the MAC address assignments by swapping them between the first and second NIC on alternating machines.
  - Linux enables you to change a NIC's MAC address using the `ifconfig` command.
  - For other operating systems custom tools may be available to perform the same task.

Deploying to IBM JVMs

When deploying Coherence on IBM JVMs please be aware of the following:

UDP Socket Buffer Sizes

There is an issue with IBM's 1.4.2, and 1.5 JVMs which may prevent them from allocating socket buffers larger then 64K (Note that buffers of 2MB are recommended for Coherence). This issue has been addressed in IBM's 1.4.2 SR7 SDK and 1.5 SR3 SDK. For performance reasons it is suggested that the patch be applied.

Deploying to Linux

When deploying Coherence on Linux please be aware of the following:

Native POSIX Thread Library (NPTL)

Early versions of the NPTL are prone to deadlock, especially when combined with 2.4 Linux Kernels. The kernel version and NPTL version can be obtained by executing the following commands:

```
uname -a
getconf GNU_LIBPTHREAD_VERSION
```

If running on a 2.4 kernel, it is recommended that you avoid using any version of the NPTL, and revert to using LinuxThreads. This can be done by setting the LD_ ASSUME_KERNEL environment variable before launching Java.

```
export LD_ASSUME_KERNEL=2.4.19
getconf GNU_LIBPTHREAD_VERSION
```

If running on a 2.6 kernel, it is recommended that you use a 1.0 or higher version of NPTL. If upgrading the NPTL version is not possible then it is then recommended that you switch to LinuxThreads.

NPTL related issues are known to occur with Red Hat Linux 9 and Red Hat Enterprise Linux 3, and are also likely to effect any 2.4 based Linux distribution with a backported version of the NPTL. See http://java.sun.com/developer/technicalArticles/JavaTechandLinux/RedHat for more details on this issue.

TSC High Resolution Timesource

Linux has several high resolution timesources to choose from, the fastest TSC (Time Stamp Counter) unfortunately is not always reliable. Linux chooses TSC by default, and during boot checks for inconsistencies, if found it switches to a slower safe timesource. The slower time sources can be 10 to 30 times more expensive to query then the TSC timesource, and may have a measurable impact on Coherence performance. Note that Coherence and the underlying JVM are not aware of the timesource which the operating system is using. It is suggested that you check your system logs (/var/log/dmesg) to verify that the following is not present.

```
kernel: Losing too many ticks!
kernel: TSC cannot be used as a timesource.
kernel: Possible reasons for this are:
kernel:   You're running with Speedstep,
kernel:   You don't have DMA enabled for your hard disk (see hdparm),
kernel:   Incorrect TSC synchronization on an SMP system (see dmesg).
kernel: Falling back to a sane timesource now.
```

As the log messages suggest, this can be caused by a variable rate CPU (SpeedStep), having DMA disabled, or incorrect TSC synchronization on multi CPU machines. If present it is suggested that you work with your system administrator to identify the cause and allow the TSC timesource to be used.

Deploying to OS X

When deploying Coherence on OS X please be aware of the following:

Multicast and IPv6

OS X defaults to running multicast over IPv6 rather then IPv4. If you run in a mixed IPv6/IPv4 environment you will need to configure your JVMs to explicitly use IPv4. This can be done by setting the java.net.preferIPv4Stack system property to true on the Java command line.

Unique Multicast Addresses and Ports

On OS X it is suggested that each Coherence cluster use a unique multicast address and port, as some versions of OS X will not take both into account when delivering packets. See the multicast-listener for details on configuring the address.

Socket Buffer Sizing

Generally Coherence prefers 2MB or higher buffers, but in the case of OS X this may result in unexpectedly high kernel CPU time, which in turn reduces throughput. For OS X the suggested buffers size is 768KB, though your own tuning may find a better

sweet spot. See "packet-buffer" on page H-32 for details on specifying the amount of buffer space Coherence will request.

Deploying to Solaris

When deploying Coherence on Solaris please be aware of the following:

Solaris 10 (x86 and SPARC)

Note: If running on Solaris 10, please review Sun issues 102712 and 102741 which relate to packet corruption and multicast disconnections. These will most often manifest as either EOFExceptions, "Large gap" warnings while reading packet data, or frequent packet timeouts. It is highly recommend that the patches for both issues be applied when using Coherence on Solaris 10 systems.

Sun Alert 102712:

Possible Data Integrity Issues on Solaris 10 Systems Using the e1000g Driver for the Intel Gigabit Network Interface Card (NIC)

Sun Alert 102741:

IGMP(1) Packets do not Contain IP Router Alert Option When Sent From Solaris 10 Systems With Patch 118822-21 (SPARC) or 118844-21 (x86/x64) or Later Installed

Solaris 10 Networking

If running on Solaris 10, please review Sun issues 102712 and 102741 which relate to packet corruption and multicast disconnections. These will most often manifest as either EOFExceptions, "Large gap" warnings while reading packet data, or frequent packet timeouts. It is highly recommend that the patches for both issues be applied when using Coherence on Solaris 10 systems.

Deploying to Sun JVMs

When deploying Coherence on Sun JVMs please be aware of the following:

Heap Sizes

With 1.4 JVMs Coherence recommends keeping heap sizes below 1GB in size per JVM. Multiple cache servers can be used allow a single machine to achieve higher capacities. With Sun's 1.5 JVMs, heap sizes beyond 1GB are reasonable, though GC tuning is still advisable to minimize long GC pauses. See Sun's GC Tuning Guide for tuning details. It is also advisable to run with fixed sized heaps as this generally lowers GC times.

AtomicLong

When available Coherence will make use of the highly concurrent AtomicLong class, which allows concurrent atomic updates to long values without requiring synchronization. Sun 1.4 client JVMs include an implementation which is not stable on some multiprocessor systems. If Coherence detects that it is being run on a Sun 1.4 client JVM it will default to a safe but slower synchronized implementation, and will output the following log message.

sun.misc.AtomicLong is not supported on this JVM; using a synchronized counter.

It is suggested that you run your 1.4 JVMs in server mode to ensure that the stable and highly concurrent version can be used. To run the JVM in server mode include the -server option on the Java command line.

Deploying to Virtual Machines

Supported Deployment

Coherence is supported within virtual machine environments, and there should see no functional differences between running it there or in a non-virtualized operating system.

Multicast Connectivity

Using virtualization adds another layer to your network topology, and like all other layers it must be properly configured to support multicast networking. See "multicast-listener" on page H-26.

Performance

It is less likely that a process running in a virtualized OS will be able to fully use gigabit Ethernet. This is not specific to Coherence, and will be visible most network intensive virtualized applications.

See the following VMWare article covering their network performance as compared to non-virtualized operating systems.

Fault Tolerance

From a Coherence fault tolerance perspective there is more configuration which needs to occur to ensure that cache entry backups reside on physically separate hardware. Manual machine identity must be configured so that Coherence can ensure that backups are not inadvertently stored on the same physical machine as the primary. This can be configured by using the machine-id element within the operational configuration file. See the configuration for "unicast-listener" on page H-48 for details.

Deploying to Windows

When deploying Coherence on Windows please be aware of the following:

Performance Tuning

Out of the box Windows is not optimized for background processes and heavy network loads. This may be addressed by running the optimize.reg script included in the Coherence installation's `bin` directory. See "Operating System Tuning" on page 20-1 for details on the optimizations which will be performed.

Personal Firewalls

If running a firewall on a machine you may have difficulties in forming a cluster consisting of multiple computers. This can be resolved by either:

- Disable the firewall, though this is generally not recommended.

- Grant full network access to the Java executable which will run Coherence.
- Open up individual address and ports for Coherence.
  - By default Coherence will use TCP and UDP ports starting at 8088, subsequent nodes on the same machine will use increasing port numbers. Coherence may also communicate over multicast, the default address and port will differ with based on the release. See "unicast-listener" on page H-48 and "multicast-listener" on page H-26 for details on address and port configuration.

Deploying to z OS

When deploying Coherence on z/OS please be aware of the following:

EBCDIC

When deploying Coherence into environments where the default character set is EBCDIC rather than ASCII, please make sure that Coherence the configuration files which are loaded from JAR files or off of the classpath are in ASCII format. Configuration files loaded directly from the file system should be stored in the systems native format of EBCDIC.

Multicast

Under some circumstances, Coherence cluster nodes that run within the same logical partition (LPAR) on z/OS on IBM zSeries cannot communicate with each other. (This problem does not occur on the zSeries when running on Linux.)

The root cause is that z/OS may bind the MulticastSocket that Coherence uses to an automatically-assigned port, but Coherence requires the use of a specific port in order for cluster discovery to operate correctly. (Coherence does explicitly initialize the java.net.MulitcastSocket to use the necessary port, but that information appears to be ignored on z/OS when there already is an instance of Coherence running within that same LPAR.)

The solution is to run only one instance of Coherence within a z/OS LPAR; if multiple instances are required, each instance of Coherence should be run in a separate z/OS LPAR. Alternatively well known addresses may be used. See "well-known-addresses" on page H-51 for more information.

N

Best Practices for Coherence Extend

Run Proxy Servers with Local Storage Disabled

Each server in a partitioned cache, including the proxy server, can store a portion of the data. The proxy server has the responsibility of accepting POF formatted data from the client (either Java, C++, or .NET), deserializing POF data to get the Java objects, serializing the Java objects, then placing the resulting data in the cluster. These tasks can be expensive in terms of CPU and memory. You can preserve resources on the proxy server by disabling its local storage.

There are several ways in which you can disable storage:

Local storage for a proxy server can be enabled or disabled with the tangosol.coherence.distributed.localstorage Java property. For example:

```
-Dtangosol.coherence.distributed.localstorage=false
```

You can also disable storage in the cache configuration file. See the description of the `<local-storage>` element in "distributed-scheme" on page D-26.

Storage can also be disabled for the proxy server by modifying the `<local-storage>` setting in its `tangosol-coherence.xml` (or `tangosol-coherence-override.xml`) file. Example N–1 illustrates setting `<local-storage>` to `false` in the `tangosol-coherence-override.xml`) file.

Example N–1 Disabling Storage in tangosol-coherence-override.xml

```
<!--
Example using tangosol-coherence-override.xml
-->
<coherence>
  <cluster-config>
    <services>
      <!--
      id value must match what's in tangosol-coherence.xml for DistributedCache
service
      -->
      <service id="3">
        <init-params>
          <init-param id="4">
            <param-name>local-storage</param-name>
            <param-value
system-property="tangosol.coherence.distributed.localstorage">false</param-value>
          </init-param>
```

```
        </init-params>
      </service>
    </services>
  </cluster-config>
</coherence>
```

Do Not Run a Near Cache on a Proxy Server

By definition, a near cache provides local cache access to recently- and/or often-used data. If a proxy server is configured with a near cache, it will locally cache data accessed by its remote clients. It is unlikely that these clients will be consistently accessing the same subset of data, thus resulting in a low hit ratio on the near cache. This will result in higher heap usage and more network traffic on the proxy nodes with little to no benefit. For these reasons, it is recommended that a near cache not be used on a proxy server. To ensure that the proxy server is not running a near cache, remove all near schemes from the cache configuration being used for the proxy. See "Near Cache" for more information.

Configure Heap NIO Space to be Equal to the Max Heap Size

NIO memory is used for the TCP connection into the proxy and for POF serialization and deserialization. Older Java installations tended to run out of heap memory because it was configured too low. Newer Java JDKs will configure off heap NIO space equal to the max heap space. On Sun JVMs, this can also be set manually with this value:

```
-XX:MaxDirectMemorySize=512M
```

Set Worker Thread Pool Sizes According to the Needs of the Application

Client applications can be classified into two general categories: active and passive.

In active applications, the Coherence*Extend client sends many requests, such as put, get, and so on, to the proxy. These requests are serviced by the proxy service. The proxy will deserialize POF data put into the cache, and serialize data it returns to the client. For these tasks, configure a larger number of daemon (worker) threads for the proxy service.

In passive applications, the client waits on events (such as map listeners) based on some specified criteria. Events are serviced by the DistributedCache service. This service uses worker threads to push events to the client. For these tasks, the thread pool configuration for the DistributedCache service should include a sufficient number of worker threads.

Note that near caches on extend clients will use map listeners under the covers for the invalidation strategies of `ALL`, `PRESENT`, and `AUTO`. Applications that are write-heavy that use near caches will generate many map events.

Be Careful When Making InvocationService Calls

InvocationService allows a member of a service to invoke arbitrary code on any node in the cluster. On Coherence*Extend however, InvocationService calls are serviced by the proxy that the client is connected to by default. When sending the call through a proxy, you cannot choose the particular node on which the code will run.

Be Careful When Placing Collection Classes in the Cache

If a Coherence*Extend client puts a collection object, (such as an `ArrayList`, `HashSet`, `HashMap`, and so on) directly into the cache, it is deserialized as an immutable array. If you then extract it and cast it to its original type, then a `ClassCastExceptions` will be returned. For example, in the following pseudo-code a new `ArrayList` class object is created and put into the cache. It is then pulled out of the cache and cast to its original type. The cast will cause a `ClassCastException` to be returned.

Example N–2 Casting an ArrayList Object

```
...
ArrayList i = new ArrayList ()
   put ( .... )
   ...
   (ArrayList)get ( ... )
...
```

You can avoid receiving this exception by either using the Java interface object (such as a `List`, `Set`, `Map`, and so on) or by encapsulating the collection object in another object.

For example, if you assign the `ArrayList` collection object to the `List` Java interface, then you can safely cast the returned data to a `List` object.

Example N–3 Assigning a ArrayList Collection Object to a List Java Interface

```
...
List i = new ArrayList()
   put(...)
   ...
   (List)get( ...
...
```

In the following pseudo-code, the `ArrayList` collection object is embedded in the Person class. You can get the class object out of the cache, then extract the collection object.

Example N–4 Embedding an ArrayList Collection Object

```
class Person {
   ...
   ArrayList i = new ArrayList ()
   ...
}
...
   put ( .... )
   ...
   (Person)get ( ... )
...
```

Run Multiple Proxies Instead of Increasing Thread Pool Size

The proxy performs POF/EL conversions in the service thread. A single proxy instance can easily bottleneck on a single core due to POF/EL conversions. Running

multiple proxy instances on the same box (instead of increasing the thread pool size) helps spread the load across more cores.

Configure POF Serializers for Cache Servers

One of the tasks the proxy server performs is to deserialize POF data into Java objects. If you run C++ or .NET applications and store data to the cache, then the conversion to Java objects could be viewed as an unnecessary step. In the current release of Coherence, you have the option of configuring a POF serializer for cache servers. This will have the effect of storing POF format data directly in the cache.

This can have the following impact on your applications:

- .NET or C++ clients that only perform puts or gets will not require a Java version of the object. Java versions will still be required if deserializing on the server side (for entry processors, cache stores, and so on).
- POF serializers remove the requirement to serialize/deserialze on the proxy, thus reducing their memory and CPU requirements.

Example N–5 illustrates a fragment from `example-pof-server.xml`, which configures a POF serializer for the distributed cache. A full POF configuration file example,is attached to this topic.

Example N–5 Configuring a POFSerializer for a Distributed Cache

```
...
    <distributed-scheme>
      <scheme-name>dist-default</scheme-name>

      <serializer>

<class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>
                <init-params>
                  <init-param>
                        <param-type>string</param-type>
                        <param-value>custom-types-pof-config.xml</param-value>
                  </init-param>
                </init-params>
          </serializer>

      <backing-map-scheme>
        <local-scheme/>
      </backing-map-scheme>

      <autostart>true</autostart>
    </distributed-scheme>
...
```

Use Node Locking Instead of Thread Locking

Coherence*Extend clients can send lock, put, and unlock requests to the cluster. The proxy holds the locks for the client. The requests for locking and unlocking can be issued at the thread level or the node level. In thread level locking, a particular thread instance belonging to the proxy (Thread 1, for example) issues the lock request. If any other threads (Thread 3, for example) issue an unlock request, they will be ignored. A successful unlock request can be issued only by the thread that issued the initial lock request. This can cause application errors since unlock requests will not succeed unless

the original thread that issues the lock is also the one that receives the request to release the lock.

In node level locking, if a particular thread instance belonging to the proxy (Thread 1, for example) issues the lock request, then any other thread (Thread 3, for example) can successfully issue an unlock request.

As an alternative to using locks, Coherence recommends that you use the `EntryProcessor` API instead. `EntryProcessors` are described in "Chapter 2, "Implement Transactions, Locks, and Concurrency."

O

Scaling Out Your Data Grid Aggregations Linearly

Coherence provides a data grid by dynamically, transparently, and automatically partitioning the data set across all storage enabled nodes in a cluster. We have been doing some scale out testing on our new Dual 2.3GHz PowerPC G5 Xserve cluster and here is one of the tests that we have performed using the data grid aggregation feature.

The new InvocableMap tightly binds the concepts of a data grid (that is, partitioned cache) and the processing of the data stored in the grid. When you take the InvocableMap and combine it with the linear scalability of Coherence itself you get an **extremely powerful** solution. The following tests show that you can take an application that Coherence provides you (the developer, the engineer, the architect, and so on) the ability to build an application when that can scale out to handle any SLA requirement, any increase in throughput requirements. For example, the following test demonstrate Coherence's ability to linearly increase the number of trades aggregated per second as you increase hardware. That is, if one machine can aggregate **X** trades per second, if you add a second machine to the data grid you will be able to aggregate **2X** trades per second, if you add a third machine to the data grid you will be able to aggregate **3X** trades per second and so on.

All of the Data Grid capabilities described below are features of Coherence Enterprise Edition and higher.

The Data

First, we need some data to aggregate. Example O–1 illustrates a `Trade` object with a three properties `Id`, `Price`, and `Symbol`.

Example O–1 Trade Object Defining Three Properties

```
package com.tangosol.examples.coherence.data;


import com.tangosol.util.Base;
import com.tangosol.util.ExternalizableHelper;
import com.tangosol.io.ExternalizableLite;

import java.io.IOException;
import java.io.NotActiveException;
import java.io.DataInput;
import java.io.DataOutput;


/**
```

```
* Example Trade class
*
* @author erm 2005.12.27
*/
public class Trade
        extends Base
        implements ExternalizableLite
    {
    /**
    * Default Constructor
    */
    public Trade()
        {
        }

    public Trade(int iId, double dPrice, String sSymbol)
        {
        setId(iId);
        setPrice(dPrice);
        setSymbol(sSymbol);
        }

    public int getId()
        {
        return m_iId;
        }

    public void setId(int iId)
        {
        m_iId = iId;
        }

    public double getPrice()
        {
        return m_dPrice;
        }

    public void setPrice(double dPrice)
        {
        m_dPrice = dPrice;
        }

    public String getSymbol()
        {
        return m_sSymbol;
        }

    public void setSymbol(String sSymbol)
        {
        m_sSymbol = sSymbol;
        }

    /**
    * Restore the contents of this object by loading the object's state from the
    * passed DataInput object.
    *
    * @param in the DataInput stream to read data from to restore the
    *           state of this object
    *
    * @throws IOException        if an I/O exception occurs
```

```
    * @throws NotActiveException if the object is not in its initial state, and
    *                            therefore cannot be deserialized into
    */
    public void readExternal(DataInput in)
            throws IOException
        {
        m_iId     = ExternalizableHelper.readInt(in);
        m_dPrice  = in.readDouble();
        m_sSymbol = ExternalizableHelper.readSafeUTF(in);
        }

    /**
    * Save the contents of this object by storing the object's state into the
    * passed DataOutput object.
    *
    * @param out the DataOutput stream to write the state of this object to
    *
    * @throws IOException if an I/O exception occurs
    */
    public void writeExternal(DataOutput out)
            throws IOException
        {
        ExternalizableHelper.writeInt(out, m_iId);
        out.writeDouble(m_dPrice);
        ExternalizableHelper.writeSafeUTF(out, m_sSymbol);
        }

    private int    m_iId;
    private double m_dPrice;
    private String m_sSymbol;
    }
```

Configure a Partitioned Cache

The cache configuration is easy through the XML Cache Configuration Elements. Example O–2 defines one wildcard `cache-mapping` that maps to one `caching-scheme` which has unlimited capacity:

Example O–2 Mapping a cache-mapping to a caching-scheme with Unlimited Capacity

```
<?xml version="1.0"?>

<!DOCTYPE cache-config SYSTEM "cache-config.dtd">

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>*</cache-name>
      <scheme-name>example-distributed</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <!--
    Distributed caching scheme.
    -->
    <distributed-scheme>
      <scheme-name>example-distributed</scheme-name>
      <service-name>DistributedCache</service-name>
```

```
      <backing-map-scheme>
        <class-scheme>
          <scheme-ref>unlimited-backing-map</scheme-ref>
        </class-scheme>
      </backing-map-scheme>

      <autostart>true</autostart>
    </distributed-scheme>

    <!--
    Backing map scheme definition used by all the caches that do
    not require any eviction policies
    -->
    <class-scheme>
      <scheme-name>unlimited-backing-map</scheme-name>

      <class-name>com.tangosol.util.SafeHashMap</class-name>
      <init-params></init-params>
    </class-scheme>
  </caching-schemes>
</cache-config>
```

Add an Index to the Price Property

Example O–3 illustrates the code to add an `index` to the `Price` property. Adding an index to this property increases performance by allowing Coherence to access the values directly rather than having to deserialize each item to accomplish the calculation

Example O–3 Adding an Index to the Price Property

```
ReflectionExtractor extPrice  = new ReflectionExtractor("getPrice");
m_cache.addIndex(extPrice, true, null);
```

In our tests the aggregation speed **was increased by more than 2x** after an index was applied.

Code to perform a Parallel Aggregation

Example O–4 illustrates the code to perform a parallel aggregation across all JVMs in the data grid. The aggregation is initiated and results received by **a single client**. That is, a single "low-power" client is able to use the full processing power of the cluster/data grid in aggregate to perform this aggregation in parallel with **just one line of code**.

Example O–4 Perform a Parallel Aggregation Across all JVMs in the Grid

```
Double DResult;
DResult = (Double) m_cache.aggregate((Filter) null, new DoubleSum("getPrice"));
```

The Testing Environment and Process

Performing a "Test Run"

A test run does several things:

1. Loads 200,000 trade objects into the data grid.
2. Adds indexes to `Price` property.
3. Performs a **parallel** aggregation of **all** trade objects stored in the data grid. This aggregation step is done 20 times to obtain an "average run time" to ensure that the test takes into account garbage collection.
4. Loads 400,000 trade objects into the data grid.
5. Repeats steps 2 and 3.
6. Loads 600,000 trade objects into the data grid.
7. Repeats steps 2 and 3.
8. Loads 800,000 trade objects into the data grid.
9. Repeats steps 2 and 3.
10. Loads 1,000,000 trade objects into the data grid.
11. Repeats steps 2 and 3.

Client Considerations: The test client itself is run on an Intel Core Duo iMac which is marked as local storage disabled. The command line used to start the client was:

```
java ... -Dtangosol.coherence.distributed.localstorage=false -Xmx128m -Xms128m
com.tangosol.examples.coherence.invocable.TradeTest
```

This "Test Suite" (and Subsequent Results) Includes Data from Four "Test Runs":

1. Start 4 JVMs **on one** Xserve - Perform a "test run"
2. Start 4 JVMs **on each of two** Xserves - Perform a "test run"
3. Start 4 JVMs **on each of three** Xserves - Perform a "test run"
4. Start 4 JVMs **on each of four** Xserves - Perform a "test run"

Server Considerations: In this case a "JVM" refers to a cache server instance (that is, a data grid node) that is a standalone JVM responsible for managing/storing the data. I used the DefaultCacheServer helper class to accomplish this.

The command line used to start the server was:

```
java ... -Xmx384m -Xms384m -server com.tangosol.net.DefaultCacheServer
```

JDK Version

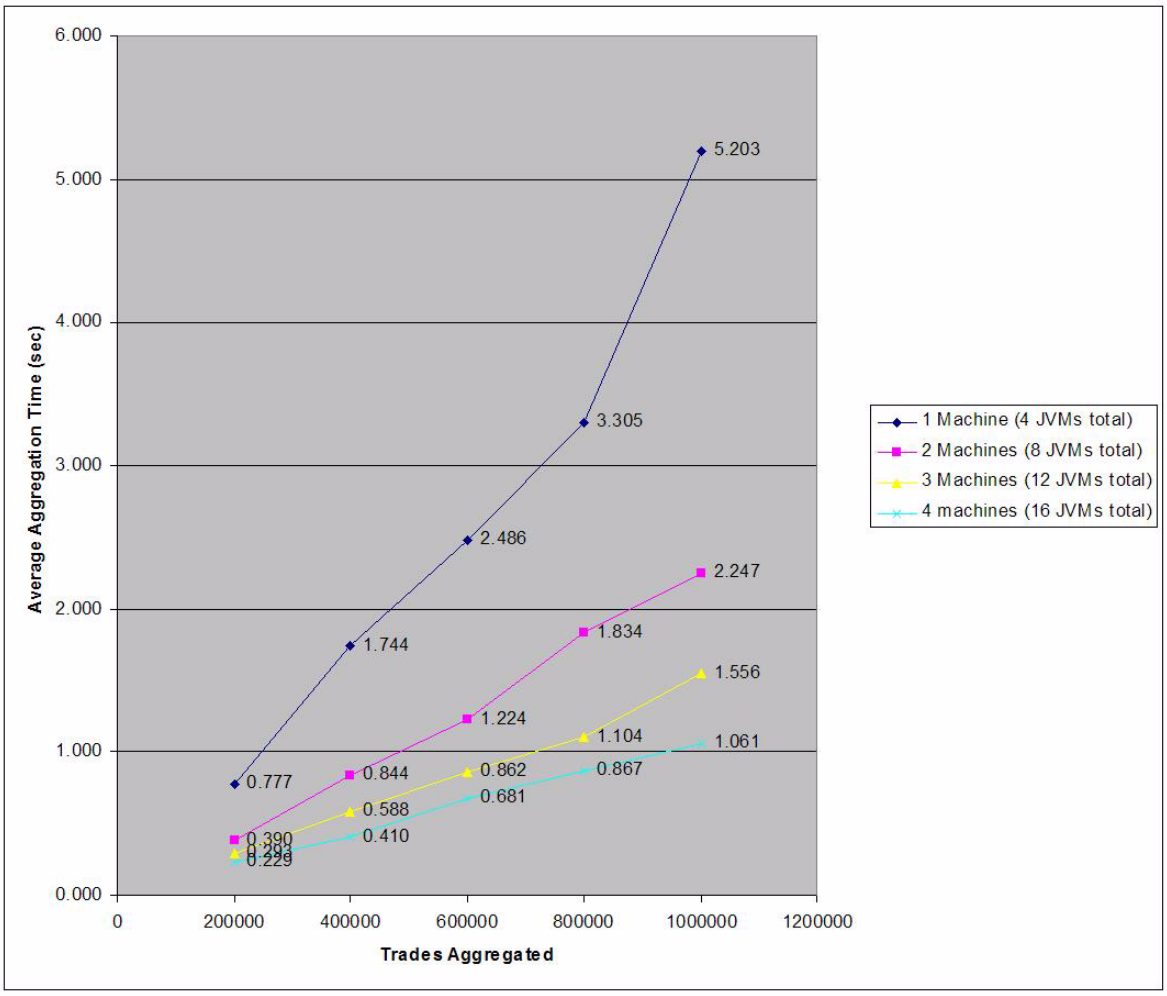
The JDK used on both the client and the servers was Java 2 Runtime Environment, Standard Edition (build 1.5.0_05-84)

The Results

As you can see in the following graph the average aggregation time for the aggregations decreases **linearly** as more cache servers/machines are added to the data grid!

> **Note:** The lowest and highest times were not used in the calculations below resulting in a data set of eighteen results used to create an average.

Figure O–1 Average Aggregation Time

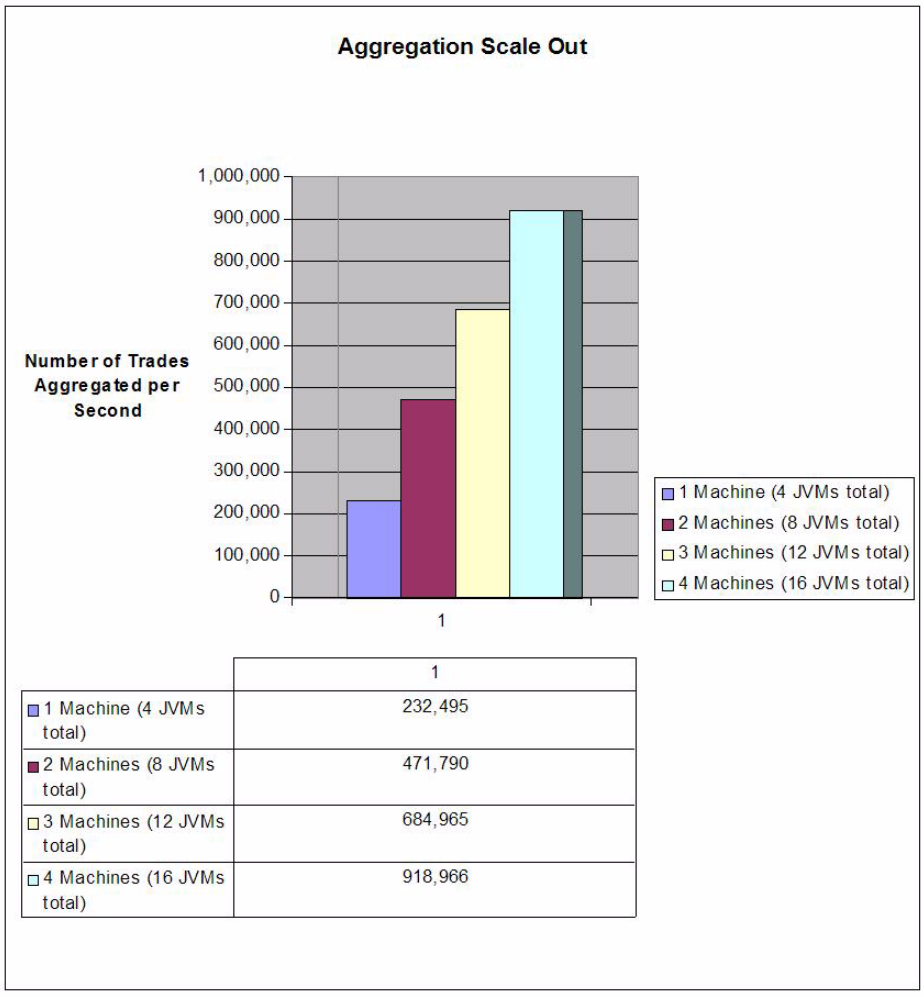


This figure is described in the text.

****

Similarly, the following graph illustrates how the aggregations per second scales **linearly** as you add more machines! When moving from 1 machine to 2 machines the trades aggregated per second **double**, when moving from 2 machines to 4 machines the trades aggregated per second **double again**.

Figure O–2 Aggregation Scale-Out



	1
1 Machine (4 JVMs total)	232,495
2 Machines (8 JVMs total)	471,790
3 Machines (12 JVMs total)	684,965
4 Machines (16 JVMs total)	918,966

This illustrations is described in the text.

****

Note: FAILOVER!

The above aggregations will complete *successfully* and *correctly* even if one of the cache servers *or* and *entire machine* fails *during* the aggregation!

Conclusion

Combining the Coherence data grid (that is, partitioned cache) with the InvocableMap features enables:

- Applications to scale out data grid calculations **linearly**;
- Groups to meet increasingly aggressive SLAs by dynamically/transparently adding more resources to the data grid. That is, if you need to achieve **1,837,932** trade aggregations per second all that is required is to start 16 more cache servers across four more machines.