Oracle® Coherence

Developer's Guide

Release 3.7.1

E22837-01

September 2011

Provides contextual information, instructions, and examples that are designed to teach Developers and Architects how to use Coherence and develop Coherence-based applications.

ORACLE®

Oracle Coherence Developer's Guide, Release 3.7.1

E22837-01

Primary Author: Joseph Ruzzi

Contents

3 Understanding Configuration

4 Building Your First Coherence Application

8 Starting and Stopping Cluster Members

9 Dynamically Managing Cluster Membership

10 Tuning TCMP Behavior

14 Caching Data Sources

15 Serialization Paged Cache

16 Using Quorum

17 Cache Configurations by Example

Part IV Using the Programming API

18 Serializing Objects

19 Using Portable Object Format

20 Pre-Loading a Cache

21 Using Cache Events

22 Querying Data In a Cache

27 Performing Transactions

B Cache Configuration Elements

List of Examples

List of Figures

List of Tables

Preface

Welcome to *Oracle Coherence Developer's Guide*. This document provides contextual information, instructions, and examples that are designed to teach developers and architects how to use Coherence and develop Coherence-based applications.

Audience

Oracle Coherence Developer's Guide is intended for the following audiences:

- **Primary Audience** – Application developers who want to understand core Oracle Coherence concepts and want to build applications that leverage an Oracle Coherence data grid.
- **Secondary Audience** – System architects who want to understand core Oracle Coherence concepts and want to build data grid-based solutions.

The audience must be familiar with Java to use this guide. In addition, the examples in this guide require the installation and use of the Oracle Coherence product. The use of an IDE is not required to use this guide, but is recommended to facilitate working through the examples. A database and basic database knowledge is required when using cache store features.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

For more information, see the following documents that are included in the Oracle Coherence documentation set:

- *Oracle Coherence Client Guide*
- *Oracle Coherence Getting Started Guide*

- *Oracle Coherence Integration Guide for Oracle Coherence*
- *Oracle Coherence Tutorial for Oracle Coherence*
- *Oracle Coherence User's Guide for Oracle Coherence*Web*
- *Oracle Coherence Java API Reference*
- *Oracle Coherence C++ API Reference*
- *Oracle Coherence .NET API Reference*
- *Oracle Coherence Release Notes for 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

Getting Started

Part I contains the following chapters:

- Chapter 1, "Introduction"
- Chapter 2, "Installing Oracle Coherence for Java"
- Chapter 3, "Understanding Configuration"
- Chapter 4, "Building Your First Coherence Application"
- Chapter 5, "Debugging in Coherence"

1 Introduction

This chapter provides a quick overview of general Coherence concepts and features. It outlines product capabilities, usage possibilities, and provides a brief overview of how one would go about implementing particular features. The items discussed in this chapter are detailed throughout this guide.

The following sections are included in this chapter:

Basic Concepts

The topics in this section describes fundamental concepts that are associated with Coherence and discusses several important features that are associated with using Coherence to cluster data.

Clustered Data Management

At the core of Coherence is the concept of clustered data management. This implies the following goals:

- A fully coherent, single system image (SSI)
- Scalability for both read and write access
- Fast, transparent failover and failback
- Linear scalability for storage and processing
- No Single-Points-of-Failure (SPOFs)
- Cluster-wide locking and transactions

Built on top of this foundation are the various services that Coherence provides, including database caching, HTTP session management, grid agent invocation and distributed queries. Before going into detail about these features, some basic aspects of Coherence should be discussed.

A single API for the logical layer, XML configuration for the physical layer

Coherence supports many topologies for clustered data management. Each of these topologies has a trade-off in terms of performance and fault-tolerance. By using a single API, the choice of topology can be deferred until deployment if desired. This allows developers to work with a consistent logical view of Coherence, while providing flexibility during tuning or as application needs change.

Caching Strategies

Coherence provides several cache implementations:

- Local Cache—Local on-heap caching for non-clustered caching.
- Replicated Cache—Perfect for small, read-heavy caches.
- Distributed Cache—True linear scalability for both read and write access. Data is automatically, dynamically and transparently partitioned across nodes. The distribution algorithm minimizes network traffic and avoids service pauses by incrementally shifting data.
- Near Cache—Provides the performance of local caching with the scalability of distributed caching. Several different near-cache strategies are available and offer a trade-off between performance and synchronization guarantees.

In-process caching provides the highest level of raw performance, since objects are managed within the local JVM. This benefit is most directly realized by the Local, Replicated, Optimistic and Near Cache implementations.

Out-of-process (client/server) caching provides the option of using dedicated cache servers. This can be helpful when you want to partition workloads (to avoid stressing the application servers). This is accomplished by using the Partitioned cache implementation and simply disabling local storage on client nodes through a single command-line option or a one-line entry in the XML configuration.

Tiered caching (using the Near Cache functionality) enables you to couple local caches on the application server with larger, partitioned caches on the cache servers, combining the raw performance of local caching with the scalability of partitioned caching. This is useful for both dedicated cache servers and co-located caching (cache partitions stored within the application server JVMs).

See Part III, "Using Caches" for detailed information on configuring and using caches.

Data Storage Options

While most customers use on-heap storage combined with dedicated cache servers, Coherence has several options for data storage:

- **On-heap**—The fastest option, though it can affect JVM garbage collection times.
- **NIO RAM**—No impact on garbage collection, though it does require serialization/deserialization.
- **NIO Disk**—Similar to NIO RAM, but using memory-mapped files.

- **File-based**—Uses a special disk-optimized storage system to optimize speed and minimize I/O.

Coherence storage is transient: the disk-based storage options are for managing cached data only. For persistent storage, Coherence offers backing maps coupled with a CacheLoader/CacheStore.

See Chapter 13, "Implementing Storage and Backing Maps," for detailed information.

Serialization Options

Because serialization is often the most expensive part of clustered data management, Coherence provides the following options for serializing/deserializing data:

- `com.tangosol.io.pof.PofSerializer` – The Portable Object Format (also referred to as POF) is a language agnostic binary format. POF was designed to be incredibly efficient in both space and time and is the recommended serialization option in Coherence. See Chapter 19, "Using Portable Object Format."
- `java.io.Serializable` – The simplest, but slowest option.
- `java.io.Externalizable` – This requires developers to implement serialization manually, but can provide significant performance benefits. Compared to `java.io.Serializable`, this can cut serialized data size by a factor of two or more (especially helpful with Distributed caches, as they generally cache data in serialized form). Most importantly, CPU usage is dramatically reduced.
- `com.tangosol.io.ExternalizableLite` – This is very similar to `java.io.Externalizable`, but offers better performance and less memory usage by using a more efficient IO stream implementation.
- `com.tangosol.run.xml.XmlBean` – A default implementation of `ExternalizableLite`.

Configurability and Extensibility

Coherence's API provides access to all Coherence functionality. The most commonly used subset of this API is exposed through simple XML options to minimize effort for typical use cases. There is no penalty for mixing direct configuration through the API with the easier XML configuration.

Coherence is designed to allow the replacement of its modules as needed. For example, the local "backing maps" (which provide the actual physical data storage on each node) can be easily replaced as needed. The vast majority of the time, this is not required, but it is there for the situations that require it. The general guideline is that 80% of tasks are easy, and the remaining 20% of tasks (the special cases) require a little more effort, but certainly can be done without significant hardship.

Namespace Hierarchy

Coherence is organized as set of services. At the root is the Cluster service. A cluster is defined as a set of Coherence instances (one instance per JVM, with one or more JVMs on each computer). A cluster is defined by the combination of multicast address and port. A TTL (network packet time-to-live; that is, the number of network hops) setting can restrict the cluster to a single computer, or the computers attached to a single switch.

Under the cluster service are the various services that comprise the Coherence API. These include the various caching services (Replicated, Distributed, and so on) and the

Invocation Service (for deploying agents to various nodes of the cluster). Each instance of a service is named, and there is typically a default service instance for each type.

The cache services contain named caches (`com.tangosol.net.NamedCache`), which are analogous to database tables—that is, they typically contain a set of related objects.

See Chapter 6, "Introduction to Coherence Clusters," for more information on the cluster service as well the other cluster-based service provided by Coherence.

Read/Write Caching

This section provides an overview of the `NamedCache` API, which is the primary interface used by applications to get and interact with cache instances. This section also includes some insight into the use of the `NamedCache` API.

NamedCache

The following source code returns a reference to a `NamedCache` instance. The underlying cache service is started if necessary. See the *Oracle Coherence Java API Reference* for details on the `NamedCache` interface.

```
import com.tangosol.net.*;
...
NamedCache cache = CacheFactory.getCache("MyCache");
```

Coherence scans the cache configuration XML file for a name mapping for `MyCache`. This is similar to Servlet name mapping in a web container's `web.xml` file. Coherence's cache configuration file contains (in the simplest case) a set of mappings (from cache name to cache scheme) and a set of cache schemes.

By default, Coherence uses the `coherence-cache-config.xml` file found at the root of `coherence.jar`. This can be overridden on the JVM command-line with `-Dtangosol.coherence.cacheconfig=file.xml`. This argument can reference either a file system path, or a Java resource path.

The `com.tangosol.net.NamedCache` interface extends several other interfaces:

- `java.util.Map`—basic `Map` methods such as `get()`, `put()`, `remove()`.
- `com.tangosol.util.ObservableMap`—methods for listening to cache events. (See Chapter 21, "Using Cache Events".
- `com.tangosol.net.cache.CacheMap`—methods for getting a collection of keys (as a `Map`) that are in the cache and for putting objects in the cache. Also supports adding an expiry value when putting an entry in a cache.
- `com.tangosol.util.QueryMap`—methods for querying the cache. (See "Query the Cache" in the *Oracle Coherence Developer's Guide*)
- `com.tangosol.util.ConcurrentMap`—methods for concurrent access such as `lock()` and `unlock()`.
- `com.tangosol.util.InvocableMap`—methods for server-side processing of cache data.

Requirements for Cached Objects

Cache keys and values must be serializable (for example, `java.io.Serializable`). Furthermore, cache keys must provide an implementation of the `hashCode()` and `equals()` methods, and those methods must return consistent results across cluster

nodes. This implies that the implementation of `hashCode()` and `equals()` must be based solely on the object's serializable state (that is, the object's non-transient fields); most built-in Java types, such as `String`, `Integer` and `Date`, meet this requirement. Some cache implementations (specifically the partitioned cache) use the serialized form of the key objects for equality testing, which means that keys for which `equals()` returns true must serialize identically; most built-in Java types meet this requirement as well.

NamedCache Usage Patterns

There are two general approaches to using a `NamedCache`:

- As a clustered implementation of `java.util.Map` with several added features (queries, concurrency), but with no persistent backing (a **"side"** cache).
- As a means of decoupling access to external data sources (an **"inline"** cache). In this case, the application uses the `NamedCache` interface, and the `NamedCache` takes care of managing the underlying database (or other resource).

Typically, an inline cache is used to cache data from:

- **a database**—The most intuitive use of a cache—simply caching database tables (in the form of Java objects).
- **a service**—Mainframe, web service, service bureau—any service that represents an expensive resource to access (either due to computational cost or actual access fees).
- **calculations**—Financial calculations, aggregations, data transformations. Using an inline cache makes it very easy to avoid duplicating calculations. If the calculation is complete, the result is simply pulled from the cache. Since any serializable object can be used as a cache key, it is a simple matter to use an object containing calculation parameters as the cache key.

See Chapter 14, "Caching Data Sources" for more information on inline caching.

Write-back options:

- **write-through**—Ensures that the external data source always contains up-to-date information. Used when data must be persisted immediately, or when sharing a data source with other applications.
- **write-behind**—Provides better performance by caching writes to the external data source. Not only can writes be buffered to even out the load on the data source, but multiple writes can be combined, further reducing I/O. The trade-off is that data is not immediately persisted to disk; however, it is immediately distributed across the cluster, so the data survives the loss of a server. Furthermore, if the entire data set is cached, this option means that the application can survive a complete failure of the data source temporarily as both cache reads and writes do not require synchronous access the data source.

To implement a read-only inline cache, you simply implement two methods on the `com.tangosol.net.cache.CacheLoader` interface, one for singleton reads, the other for bulk reads. Coherence provides an abstract class `com.tangosol.net.cache.AbstractCacheLoader` which provides a default implementation of the bulk method, which means that you need only implement a single method: `public Object load(Object oKey)`. This method accepts an arbitrary cache key and returns the appropriate value object.

If you want to implement read/write caching, you must extend `com.tangosol.net.cache.AbstractCacheStore` (or implement the interface `com.tangosol.net.cache.CacheStore`), which adds the following methods:

```
public void erase(Object oKey);
public void eraseAll(Collection colKeys);
public void store(Object oKey, Object oValue);
public void storeAll(Map mapEntries);
```

The method `erase()` should remove the specified key from the external data source. The method `store()` should update the specified item in the data source if it exists, or insert it if it does not presently exist.

After the `CacheLoader/CacheStore` is implemented, it can be connected through the `coherence-cache-config.xml` file.

Querying the Cache

Coherence provides the ability to query cached data. With partitioned caches, the queries are indexed and parallel, which means that adding servers to a partitioned cache not only increases throughput (total queries per second) but also reduces latency, with queries taking less user time. To query against a NamedCache, all objects should implement a common interface (or base class). Any field of an object can be queried; indexes are optional, and used to increase performance. With a replicated cache, queries are performed locally, and do not use indexes. See Chapter 22, "Querying Data In a Cache," for detailed information.

To add an index to a `NamedCache`, you first need a value extractor (which accepts as input a value object and returns an attribute of that object). Indexes can be added blindly (duplicate indexes are ignored). Indexes can be added at any time, before or after inserting data into the cache.

It should be noted that queries apply only to cached data. For this reason, queries should not be used unless the entire data set has been loaded into the cache, unless additional support is added to manage partially loaded sets.

Developers have the option of implementing additional custom filters for queries, thus taking advantage of query parallel behavior. For particularly performance-sensitive queries, developers may implement index-aware filters, which can access Coherence's internal indexing structures.

Coherence includes a built-in optimizer, and applies indexes in the optimal order. Because of the focused nature of the queries, the optimizer is both effective and efficient. No maintenance is required.

Transactions

Coherence provides various transaction options. The options include: basic data concurrency using the `ConcurrentMap` interface and `EntryProcessor` API, atomic transactions using the Transaction Framework API, and atomic transactions with full XA support using the Coherence resource adapter. See Chapter 27, "Performing Transactions" for detailed instructions.

HTTP Session Management

Coherence*Web is an HTTP session-management module with support for a wide range of application servers. See *Oracle Coherence User's Guide for Oracle Coherence*Web* for more information on Coherence*Web.

Using Coherence session management does not require any changes to the application. Coherence*Web uses the NearCache technology to provide fully fault-tolerant caching, with almost unlimited scalability (to several hundred cluster nodes without issue).

Invocation Service

The Coherence invocation service can deploy computational agents to various nodes within the cluster. These agents can be either execute-style (deploy and asynchronously listen) or query-style (deploy and synchronously listen). See Chapter 24, "Processing Data In a Cache," for more information on using the invocation service.

The invocation service is accessed through the `com.tangosol.net.InvocationService` interface and includes the following two methods:

Example 1–1 Methods in the InvocationService API

```
public void execute(Invocable task, Set setMembers, InvocationObserver observer);
public Map query(Invocable task, Set setMembers);
```

An instance of the service can be retrieved from the `com.tangosol.net.CacheFactory` class.

Coherence implements the `WorkManager` API for task-centric processing.

Events

All `NamedCache` instances in Coherence implement the `com.tangosol.util.ObservableMap` interface, which allows the option of attaching a cache listener implementation (of `com.tangosol.util.MapListener`). It should be noted that applications can observe events as logical concepts regardless of which computer caused the event. Customizable server-based filters and lightweight events can minimize network traffic and processing. Cache listeners follow the JavaBean paradigm, and can distinguish between system cache events (for example, eviction) and application cache events (for example, get/put operations).

Continuous Query functionality provides the ability to maintain a client-side "materialized view". Similarly, any service can be watched for members joining and leaving, including the cluster service and the cache and invocation services.

See Chapter 21, "Using Cache Events," for more detailed information on using events.

Object-Relational Mapping Integration

Most ORM products support Coherence as an "L2" caching plug-in. These solutions cache entity data inside Coherence, allowing application on multiple servers to share cached data. See *Oracle Coherence Integration Guide for Oracle Coherence* for more information.

C++/.NET Integration

Coherence provides support for cross-platform clients (over TCP/IP). All clients use the same wire protocol (the servers do not differentiate between client platforms). Also, note that there are no third-party components in any of these clients (such as embedded JVMs or language bridges). The wire protocol supports event feeds and

coherent in-process caching for all client platforms. See *Oracle Coherence Client Guide* for complete instructions on using Coherence*Extend to support remote C++ and .NET clients.

Management and Monitoring

Coherence offers management and monitoring facilities by using Java Management Extensions (JMX). See *Oracle Coherence Management Guide* for detailed information on using JMX with Coherence.

2

Installing Oracle Coherence for Java

This chapter provides instructions for installing Oracle Coherence for Java (simply referred to as Coherence). The chapter does not include instructions for installing Coherence*Extend client distributions (C++ and .NET) or Coherence*Web. Refer to the *Oracle Coherence Client Guide* and the *Oracle Coherence User's Guide for Oracle Coherence*Web*, respectively, for instructions on installing these components.

The following sections are included in this chapter:

System Requirements

The following are suggested minimum system requirements for installing Coherence in a development environment:

- 65 MB disk space for installation
- 1 GB of RAM (assuming a maximum Java heap size of 512MB) – This amount of RAM can ideally support a maximum cache size of 150MB on a single node that is configured to store a backup of all data (150MB x 2) and leaves more than a 1/3 of the heap available for scratch and JVM tasks. See *Oracle Coherence Administrator's Guide* for recommendations on calculating cache size.
- 1.6 update 23 JVM or later
- Windows or UNIX-based system that supports the required Java Version
- Network adapter

Extracting the Distribution

Coherence is distributed as a ZIP file. Use a ZIP utility or the `unzip` command-line utility to extract the ZIP file to a location on the target computer. The extracted files are organized within a single directory called `coherence`. The complete path to the `coherence` directory is referred to as *`COHERENCE_HOME`* throughout this documentation. For example, `C:\`*`INSTALL_DIR`*`\coherence`.

The following example uses the `unzip` utility to extract the distribution to the `/opt` directory which is the suggested installation directory on UNIX-based operating

systems. Use the ZIP utility provided with the target operating system if the `unzip` utility is not available.

```
unzip /path_to_zip/coherence-version_number.zip -d /opt
```

The following example extracts the distribution using the `unzip` utility to the `C:\` directory on the Windows operating system.

```
unzip C:\path_to_zip\coherence-version_number.zip -d C:\
```

The following list describes the directories that are included in *COHERENCE_HOME*:

- `bin` – This directory includes a set of common scripts for performing different tasks, such as: starting a cache server, starting development tools, and performing network tests. The scripts are provided in both Windows (`.cmd`) and UNIX-based (`.sh`) formats.
- `doc` – This directory contains a link to the Coherence documentation.
- `lib` – This directory includes all delivered libraries. The `coherence.jar` is the main development and run-time library and is discussed in detail throughout this documentation.

Setting Environment Variables

The following system environment variables can be set, but they are not required to run Coherence:

- `JAVA_HOME` – This variable is used when running the scripts that are included in the *COHERENCE_HOME*`/bin` directory. The value of this variable is the full path to the Java installation directory. If `JAVA_HOME` is not set, the scripts use the computer's default Java installation. Set this variable to ensure that the scripts use a specific Java version.
- `COHERENCE_HOME` – This variable is typically set as a convenience. The value of this variable is the full path to the *INSTALL_DIR*`/coherence` directory.

Running Coherence for the First Time

The *COHERENCE_HOME*`/bin` directory includes two scripts that are used during development and testing and are provided as a design-time convenience. The `cache-server` script starts a cache server using a default configuration. The `coherence` script starts a cache factory instance using a default configuration. The cache factory instance includes a command-line tool that is used to (among other things) create and interact with a cache.

In this scenario, a basic cluster is created and then the command-line tool is used to create and interact with a cache that is hosted in the cluster.

Create a Basic Cluster

In this step, a basic cluster is created that contains three separate Java processes: a cache server and two cache factory instances. For simplicity, the three processes are collocated on a single computer. The cache server, by default, is configured to store backup data. The two cache factory instances, by default, are configured not to store backup data. As each process is started, they automatically join and become cluster members (also referred to as cluster nodes).

For this example, the Coherence out-of-box default configuration is slightly modified to create a unique cluster which ensures that these cluster members do not attempt to join an existing Coherence cluster that may be running on the network.

> **Note:** The Coherence default behavior is to use multicast to find cluster members. Coherence can be configured to use unicast if a network does not allow the use of multicast. See "Using Well Known Addresses" on page 7-9 for details.

To create a basic cluster:

1. Using a text editor, open the *COHERENCE_HOME*`/bin/cache-server` script.
2. Modify the `java_opts` variable to include the `tangosol.coherence.cluster` and the `tangosol.coherence.clusterport` system properties as follows:

   ```
   set java_opts="-Xms%memory% -Xmx%memory% -Dtangosol.coherence.cluster=cluster_
   name -Dtangosol.coherence.clusterport=port"
   ```

   Replace *cluster_name* and *port* with values that are unique for this cluster. For example, use your name for the cluster name and the last four digits of your phone number for the port.
3. Save and close the `cache-server` script.
4. Repeat steps 1 to 3 for the *COHERENCE_HOME*`/bin/coherence` script.
5. Run the `cache-server` script. The cache server starts and output is emitted that provides information about this cluster member.
6. Run 2 instances of the `coherence` script. As each instance is started, output is emitted that provides information about the respective cluster members. Each instance returns a command prompt for the command-line tool.

Create a Cache

In this step, a cache is created and hosted on the basic cluster. A simple string is entered into the cache using the command-line tool of the first cache factory instance. The string is then retrieved from the cache using the command-line tool of the second cache factory instance. The example is simplistic and not very practical, but it does quickly demonstrate the distributed nature of Coherence caches. Moreover, these steps are typically performed directly using the Coherence API.

To create a cache:

1. At the command prompt for either cache factory instance, create a cache named `Test` using the `cache` command:

   ```
   cache Test
   ```
2. At the command prompt, use the `put` command to place a simple string in the new cache by entering a key/value pair (separated by a space):

   ```
   put key1 Hello
   ```

   The command returns and displays `null`. The `put` command always returns the previous value for a given key. The `null` value is returned because this is the first value entered for this key.
3. Switch to the other cache factory instance and from the command prompt create the `Test` cache using the `cache` command:

```
cache Test
```

4. From this command prompt, retrieve the string in the cache using the `get` command and entering the key name:

   ```
   get key1
   ```

   The command returns and displays `hello`. Either cache factory process can add or remove cache entries because the processes are part of the same cluster and because the `Test` cache is known to all cluster members. In addition, since the cache server is storing a backup of the cache data, either cache factory process (or both) can be shutdown and the cache data persists.

3

Understanding Configuration

This chapter describes each of the default configuration files that are distributed with Coherence and details how applications and solutions override these files when creating their own Coherence configurations.

The following sections are included in this chapter:

Overview of the Default Configuration Files

The Coherence distribution includes a set of default XML configuration files that are included within the *COHERENCE_HOME*\lib\coherence.jar library. The easiest way to inspect these files and their associated schemas is to extract the Coherence library to a directory.

The configuration files provide a default setup that allows Coherence to be used out-of-box with minimal changes. The files are for demonstration purposes only and can be reused or changed as required for a particular application or solution. However, the recommended approach is to provide configuration files that override the default configuration files.

The default configuration files include:

- tangosol-coherence.xml – This files provides operational and run-time settings and is used to create and configure cluster, communication, and data management services. This file is typically referred to as the operational deployment descriptor. The schema for this file is the coherence-operational-config.xsd file. See Appendix A, "Operational Configuration Elements," for a complete reference of the elements in the operational deployment descriptor.
- tangosol-coherence-override-dev.xml – This file overrides operational settings in the tangosol-coherence.xml file when Coherence is started in developer mode. By default, Coherence is started in developer mode and the

settings in this file are used. The settings in this file are suitable for development environments. The schema file for this override file and the schema for the operational deployment descriptor are the same.

- `tangosol-coherence-override-eval.xml` – This file overrides operational settings in the `tangosol-coherence.xml` file when Coherence is started in evaluation mode. The settings in this file are suitable for evaluating Coherence. The schema file for this override file and the schema for the operational deployment descriptor are the same.
- `tangosol-coherence-override-prod.xml` – This file overrides operational settings in the `tangosol-coherence.xml` file when Coherence is started in production mode. The settings in this file are suitable for production environments. The schema file for this override file and the schema for the operational deployment descriptor are the same.
- `coherence-cache-config.xml` – This file is used to specify the various types of caches which can be used within a cluster. This file is typically referred to as the cache configuration deployment descriptor. The schema for this file is the `coherence-cache-config.xsd` file. See Appendix B, "Cache Configuration Elements," for a complete reference of the elements in this file.
- `coherence-pof-config.xml` – This file is used to specify custom data types when using Portable Object Format (POF) to serialize objects. This file is typically referred to as the POF configuration deployment descriptor. The schema for this file is the `coherence-pof-config.xsd` file. See Appendix D, "POF User Type Configuration Elements," for a complete reference of the elements in this file.
- Management configuration files – A set of files that are used to configure Coherence management reports. The files are located in the `/reports` directory within `coherence.jar`. The files include a report group configuration files (`report-group.xml`, the default), which refer to any number of report definition files. Each report definition file results in the creation of a report file that displays management information based on a particular set of metrics. The schema for these files are the `coherence-report-config.xsd` file and the `coherence-report-group-config.xsd` file, respectively. See *Oracle Coherence Management Guide* for detailed information on using reports and a reference for the elements in these configuration files.

Specifying an Operational Configuration File

The `tangosol-coherence.xml` operational deployment descriptor provides operational and run-time settings and is used to create and configure cluster, communication, and data management services. At run time, Coherence uses the first instance of `tangosol-coherence.xml` that is found in the classpath.

The default operational deployment descriptor that is shipped with Coherence is located in the root of the `coherence.jar` library. This file can be changed as required; however, overriding this file is recommended when configuring the operational run time. See "Understanding the XML Override Feature" on page 3-15 for detailed information about the XML override feature.

The following topics are included in this section:

- Viewing Which Operational Override Files are Loaded

Refer to Part II, "Using Coherence Clusters" for detailed instructions on configuring the operational run time.

Using the Default Operational Override File

Elements in the default `tangosol-coherence.xml` file are overridden by placing an operational override file named `tangosol-coherence-override.xml` in the classpath at run time. The structure of the override file and the operational deployment descriptor are the same except that all elements are optional. The override file includes only the elements that are being changed. Any missing elements are loaded from the `tangosol-coherence.xml` file.

In general, using the operational override file provides the most comprehensive method of configuring the operational run time and is used in both development and production environments.

To use the default operational override file:

1. Create a file named `tangosol-coherence-override.xml`.
2. Edit the file and add any operational elements that are to be overridden.

   The following example configures a cluster name and overrides the default cluster name:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <member-identity>
         <cluster-name system-property="tangosol.coherence.cluster">
            MyCluster</cluster-name>
      </member-identity>
   </cluster-config>
</coherence>
```

3. Save and close the file.
4. Make sure the location of the operational override file is located in the classpath at run time.

   The following example demonstrates starting a cache server that uses an override file that is located in *COHERENCE_HOME*.

```
java -cp COHERENCE_HOME;COHERENCE_HOME\lib\coherence.jar
com.tangosol.net.DefaultCacheServer
```

> **Tip:** When using the `cache-server` and `coherence` scripts during development, add the location of the `tangosol-coherence-override.xml` file to the classpath using the Java `-cp` argument in each of the scripts.

Specifying an Operational Override File

The `tangosol.coherence.override` system property specifies an operational override file to be used instead of the default

tangosol-coherence-override.xml file. The structure of the specified file and the operational deployment descriptor are the same except that all elements are optional. Any missing elements are loaded from the tangosol-coherence.xml file.

The tangosol.coherence.override system property provides an easy way to switch between different operational configurations and is convenient during development and testing.

To specify an operational override file:

1. Create a text file.
2. Edit the file and add any operational elements that are to be overridden.

   The following example configures the multicast port number:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <multicast-listener>
         <port system-property="tangosol.coherence.clusterport">3059</port>
      </multicast-listener>
   </cluster-config>
</coherence>
```

3. Save the file as an XML file and close the file.
4. Specify the name of the operational override file as a value of the tangosol.coherence.override system property. If the file is not located in the classpath, enter the full (or relative) path to the file and the name. The system property also supports the use of a URL when specifying the location of an operational override file.

   The following example demonstrates starting a cache server and using an operational override file that is named cluster.xml which is located in *COHERENCE_HOME*.

```
java -Dtangosol.coherence.override=cluster.xml -cp COHERENCE_HOME;COHERENCE_
HOME\lib\coherence.jar com.tangosol.net.DefaultCacheServer
```

Defining Override Files for Specific Operational Elements

Override files can be created to override the contents of specific operational elements. The override files follow the same structure as the operational deployment descriptor except that their root element must match the element that is to be overridden. See "Defining Custom Override Files" on page 3-17 for detailed information on defining override files for specific operational elements.

In general, override files for specific operational elements provides fine-grained control over which portions of the operational deployment descriptor may be modified and allows different configurations to be created for different deployment scenarios.

To define override files for specific operational elements:

1. Create a tangosol-coherence-override.xml file as described in "Using the Default Operational Override File" on page 3-3.

2. Add an `xml-override` attribute to an element that is to be overridden. The value of the `xml-override` attribute is the name of an override file.

   The following example defines an override file named `cluster-config.xml` that is used to override the `<cluster-config>` element.

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config xml-override="/cluster-config.xml">
   ...
   </cluster-config>
</coherence>
```

3. Save and close the file.
4. Create a text file.
5. Edit the file and add an XML node that corresponds to the element that is to be overridden. The XML root element must match the element that is to be overridden.

   Using the example from step 2, the following node is created to override the `<cluster-config>` element and specifies a multicast join timeout.

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

<cluster-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <multicast-listener>
      <join-timeout-milliseconds>4000</join-timeout-milliseconds>
   </multicast-listener>
</cluster-config>
```

6. Save the file as an XML file with the same name used in the `xml-override` attribute.
7. Make sure the location of both override files are located in the classpath at run time.

   The following example demonstrates starting a cache server that uses override files that are located in *COHERENCE_HOME*.

```
java -cp COHERENCE_HOME;COHERENCE_HOME\lib\coherence.jar
com.tangosol.net.DefaultCacheServer
```

Viewing Which Operational Override Files are Loaded

The output for a Coherence node indicates the location and name of the operational configuration files that are loaded at startup. The operational configuration messages are the first messages to be emitted when starting a process. The output is especially helpful when using multiple override files and is often useful when developing and testing Coherence applications and solutions.

The following example output demonstrates typical messages that are emitted:

```
Loaded operational configuration from resource "jar:file:/D:/coherence/lib/
```

```
   coherence.jar!/tangosol-coherence.xml"
Loaded operational overrides from resource "jar:file:/D:/coherence/lib/
   coherence.jar!/tangosol-coherence-override-dev.xml"
Loaded operational overrides from resource "file:/D:/coherence/
   tangosol-coherence-override.xml"
Optional configuration override "/cluster-config.xml" is not specified
Optional configuration override "/custom-mbeans.xml" is not specified
```

The above output indicates that the operational deployment descriptor included in `coherence.jar` was loaded and that settings in this file are overridden by two loaded override files: `tangosol-coherence-override-dev.xml` and `tangosol-coherence-override.xml`. In addition, two override files were defined for specific operational elements but were not found or loaded at run time.

Specifying a Cache Configuration File

The `coherence-cache-config.xml` cache configuration deployment descriptor file is used to specify the various types of caches that can be used within a cluster. At run time, Coherence uses the first `coherence-cache-config.xml` file that is found in the classpath. A sample `coherence-cache-config.xml` file is included with Coherence and is located in the root of the `coherence.jar` library. The sample file is provided only for demonstration purposes. It can be changed or reused as required; however, it is recommended that a custom cache configuration deployment descriptor be created instead of using the sample file.

Note:

- It is recommended (although not required) that all cache server nodes within a cluster use identical cache configuration descriptors.
- Coherence requires a cache configuration deployment descriptor to start. If the cache configuration deployment descriptor is not found at run time, an error message indicates that there was a failure loading the configuration resource and also provides the name and location for the file that was not found.

The following topics are included in this section:

- Using a Default Cache Configuration File
- Overriding the Default Cache Configuration File
- Using the Cache Configuration File System Property
- Viewing Which Cache Configuration File is Loaded

Refer to Part III, "Using Caches" for detailed instructions on configuring caches.

Using a Default Cache Configuration File

Coherence is configured out-of-box to use the first `coherence-cache-config.xml` file that is found on the classpath. To use a `coherence-cache-config.xml` file, the file must be located on the classpath and must precede the `coherence.jar` library; otherwise, the sample `coherence-cache-config.xml` file that is located in the `coherence.jar` is used.

To use a default cache configuration file:

1. Make a copy of the sample `coherence-cache-config.xml` file that is located in the `coherence.jar` and save it to a different location. The cache definitions that are included in the sample file are for demonstration purposes and are used as a starting point for creating solution-specific cache configurations.
2. Ensure that the location where the `coherence-cache-config.xml` file is saved is in the classpath at run time and that the location precedes the `coherence.jar` file in the classpath.

   The following example demonstrates starting a cache server that uses a `coherence-cache-config.xml` cache configuration file that is located in *COHERENCE_HOME*.

   ```
   java -cp COHERENCE_HOME;COHERENCE_HOME\lib\coherence.jar
   com.tangosol.net.DefaultCacheServer
   ```

Overriding the Default Cache Configuration File

The default name and location of the cache configuration deployment descriptor is specified in the operational deployment descriptor within the `<configurable-cache-factory-config>` element. This element can be overridden to specify a different name an location to be used for the default cache configuration file.

To override the default cache configuration file:

1. Make a copy of the default `coherence-cache-config.xml` cache configuration file that is located in the `coherence.jar` and save it to a location with a different name.
2. Create a `tangosol-coherence-override.xml` file as described in "Using the Default Operational Override File" on page 3-3.
3. Edit the operational override file and enter a `<configurable-cache-factory-config>` node that specifies the name of the cache configuration file created in step 1. If the cache configuration file is not located in the classpath, enter the full (or relative) path to the file as well. The element also supports the use of a URL when specifying the location of a cache configuration file.

   The following example specifies a cache configuration deployment descriptor called `MyConfig.xml`.

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

   <coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
      xsi:schemaLocation="http://xmlns.oracle.com/coherence/
      coherence-operational-config coherence-operational-config.xsd">
      <configurable-cache-factory-config>
         <init-params>
            <init-param>
               <param-type>java.lang.String</param-type>
               <param-value system-property="tangosol.coherence.cacheconfig">
                  MyConfig.xml</param-value>
            </init-param>
         </init-params>
      </configurable-cache-factory-config>
   </coherence>
   ```

4. Save and close the file.
5. Ensure that the location of the operational override file is located in the classpath at run time.

   The following example demonstrates starting a cache server using an operational override file and a custom cache configuration file that are located in *COHERENCE_HOME*.

   ```
   java -cp COHERENCE_HOME;COHERENCE_HOME\lib\coherence.jar
   com.tangosol.net.DefaultCacheServer
   ```

Using the Cache Configuration File System Property

The `tangosol.coherence.cacheconfig` system property is used to specify a custom cache configuration deployment descriptor to be used instead of the configured default cache configuration deployment descriptor. The system property provides an easy way to switch between different configurations and is convenient during development and testing.

To specify a custom cache configuration file, enter the name of the file as a value of the `tangosol.coherence.cacheconfig` system property. This is typically done as a `-D` Java option when starting a Coherence node. If the file is not located in the classpath, enter the full (or relative) path to the file and the name. The system property also supports the use of a URL when specifying the location of a cache configuration file.

The following example starts a cache server and specifies a cache configuration deployment descriptor called `MyConfig.xml` that is located in *COHERENCE_HOME*.

```
java -Dtangosol.coherence.cacheconfig=MyConfig.xml -cp COHERENCE_HOME;COHERENCE_
HOME\lib\coherence.jar com.tangosol.net.DefaultCacheServer
```

Viewing Which Cache Configuration File is Loaded

The output for a Coherence node indicates the location and name of the cache configuration deployment descriptor that is loaded at startup. The configuration message is the first message to display after the Coherence copyright text is emitted. The output is especially helpful when developing and testing Coherence applications and solutions.

The following example output demonstrates a cache configuration message which indicates that a cache configuration deployment descriptor named `Myconfig.xml` was loaded:

```
Loaded cache configuration from resource "file:/D:/coherence/Myconfig.xml"
```

Specifying a POF Configuration File

The `pof-config.xml` POF configuration deployment descriptor file is used to specify custom user types when using Portable Object Format (POF) for serialization. At run time, Coherence uses the first instance of `pof-config.xml` that is found in the classpath.

Note:

- It is recommended that all nodes within a cluster use identical POF configuration deployment descriptors.
- A POF configuration deployment descriptor is only loaded if the POF serializer is either configured as part of a cache scheme or configured globally for all cache schemes. The default `coherence-cache-config.xml` provides an example cache scheme that defines the POF serializer, but it is commented out by default.

The default POF configuration deployment descriptor that is distributed with Coherence is located in the root of the `coherence.jar` library. This file should be customized, replaced, or extended for a particular application or solution. By default, the deployment descriptor references the `coherence-pof-config.xml` file. This is where the Coherence specific user types are defined and should always be included when extending or creating a POF configuration file.

The following topics are included in this section:

- Using the POF Configuration File System Property
- Combining Multiple POF Configuration Files
- Viewing Which POF Configuration Files are Loaded

Refer to Chapter 19, "Using Portable Object Format" for detailed instructions on configuring POF user types.

Using the POF Configuration File System Property

The `tangosol.pof.config` system property is used to specify a custom POF configuration deployment descriptor to be used instead of the default `pof-config.xml` file. The system property provides an easy way to switch between different configurations and is convenient during development and testing.

To specify a custom POF configuration file:

1. Create an XML file.
2. Edit the file and create a `<pof-config>` node that includes the default Coherence POF user types:

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

<pof-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-pof-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-pof-config
   coherence-pof-config.xsd">
   <user-type-list>
      <include>coherence-pof-config.xml</include>
   </user-type-list>
</pof-config>
```

3. Save and close the file.
4. Enter the name of the file as a value of the `tangosol.pof.config` system property. This is typically done as a `-D` Java option when starting a Coherence node. If the file is not located in the classpath, enter the full (or relative) path to the

file and the name. The system property also supports the use of a URL when specifying the location of a POF configuration file.

The following example starts a cache server and specifies a POF configuration deployment descriptor called `MyPOF.xml` that is located in *COHERENCE_HOME*.

```
java -Dtangosol.pof.config=MyPOF.xml -cp COHERENCE_HOME;COHERENCE_
HOME\lib\coherence.jar com.tangosol.net.DefaultCacheServer
```

Combining Multiple POF Configuration Files

The `<include>` element is used within a POF configuration deployment descriptor to include user types that are defined in different POF configuration deployment descriptors. This allows user types to be organized in meaningful ways, such as by application or development group.

> **Note:** When combining multiple POF configuration files, each user type that is defined must have a unique `<type-id>`. If no type identifier is included, then the type identifiers are based on the order in which the user types appear in the composite configuration file.

To combine multiple POF configuration files:

1. Open an existing POF configuration file that is being loaded at startup.
2. Add an `<include>` element whose value is the name of a POF configuration file. If the file is not located in the classpath, enter the full (or relative) path to the file and the name. A URL can also be used to locate the file.

   The following example combines two POF configuration files in addition to the default Coherence POF configuration file:

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

   <pof-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://xmlns.oracle.com/coherence/coherence-pof-config"
      xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-pof-config
      coherence-pof-config.xsd">
      <user-type-list>
         <include>coherence-pof-config.xml</include>
         <include>hr-pof-config.xml</include>
         <include>crm-pof-config.xml</include>
      </user-type-list>
   </pof-config>
   ```

3. Save and close the file.
4. If required, ensure that the location of the POF configuration files are located in the classpath at run time.

   The following example demonstrates starting a cache server that uses POF configuration files that are located in *COHERENCE_HOME*.

   ```
   java -cp COHERENCE_HOME;COHERENCE_HOME\lib\coherence.jar
   com.tangosol.net.DefaultCacheServer
   ```

Viewing Which POF Configuration Files are Loaded

The output for a Coherence node indicates the location and name of the POF configuration deployment descriptors that are loaded at startup. The configuration messages are among the messages that display after the Coherence copyright text is emitted and are associated with the cache service that is configured to use POF. The output is especially helpful when developing and testing Coherence applications and solutions.

The following example output demonstrates POF configuration messages which indicate that four POF configuration deployment descriptors were loaded:

```
Loading POF configuration from resource "file:/D:/coherence/my-pof-config.xml"
Loading POF configuration from resource
"file:/D:/coherence/coherence-pof-config.xml"
Loading POF configuration from resource "file:/D:/coherence/hr-pof-config.xml"
Loading POF configuration from resource "file:/D:/coherence/crm-pof-config.xml"
```

Specifying Management Configuration Files

There are several different configuration files that are used to configure management. These include:

- report group configuration file – A report group configuration file is used to list the name and location of report definition files and the output directory where reports are written. The name and location of this file is defined in the operational deployment descriptor. By default, the `report-group.xml` file is used and is located in the `/reports` directory of the `coherence.jar`. Additional report group configuration file are provided and custom report group files can be created as required.
- report configuration files – A report configuration file defines a report and results in the creation of a report file that displays management information for a particular set of metrics. Report configuration files must be referenced in a report group configuration file to be used at run time. The default report configuration files are located in the `/reports` directory of the `coherence.jar` and are referenced by the default report group configuration file. Custom report configuration files can be created as required.
- `custom-mbeans.xml` – This file is the default MBean configuration override file and is used to define custom MBeans (that is, application-level MBeans) within the Coherence JMX management and monitoring framework. This allows any application-level MBean to be managed and monitored from any node within the cluster. Custom MBeans can be defined within the operational override file. However, the MBean configuration override file is typically used instead.

The following topics are included in this section:

- Specifying a Custom Report Group Configuration File
- Specifying an MBean Configuration File
- Viewing Which Management Configuration Files are Loaded

See *Oracle Coherence Management Guide* for detailed instructions on managing Coherence.

Specifying a Custom Report Group Configuration File

The name and location of the default report group configuration file is specified in the operational configuration deployment descriptor within the `<management-config>` node. A custom report group configuration file can be specified by either using an operational override file or a system property.

> **Note:** The report group configuration file is only loaded if JMX management is enabled. The examples in this section demonstrate enabling JMX management on nodes that host an MBean server.

Overriding the Default Report Group Configuration File

The name and location of a custom report group configuration file can be specified using an operational override file. This mechanism overrides the default name and location of the report group configuration file.

To override the default report group configuration file:

1. Create an XML file.
2. Edit the file and create a `<report-group>` node as follows. This example configures a single report.

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

<report-group xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-report-group-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-report-group-config coherence-report-group-config.xsd">
   <frequency>1m</frequency>
   <output-directory>./</output-directory>
   <report-list>
      <report-config>
         <location>reports/report-node.xml</location>
      </report-config>
   </report-list>
</report-group>
```

3. Save and close the file.
4. Create a `tangosol-coherence-override.xml` file as described in "Using the Default Operational Override File" on page 3-3.
5. Edit the file and enter a `<management-config>` node that specifies the name of the report group configuration file. If the report group configuration file is not located in the classpath, enter the full (or relative) path to the file as well. The element also supports the use of a URL when specifying the location of a report group configuration file.

   The following example enables JMX management and specifies a report group configuration deployment descriptor called `my-group.xml`.

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <management-config>
      <managed-nodes system-property="tangosol.coherence.management">all
```

```
      </managed-nodes>
      <reporter>
         <configuration system-property="tangosol.coherence.management.report.
            configuration">my-group.xml</configuration>
      </reporter>
   </management-config>
</coherence>
```

6. Save and close the file.
7. Ensure that the location of the operational override file is located in the classpath at run time.

   The following example demonstrates starting a cache server using an operational override file and a report group configuration file that are located in *COHERENCE_HOME*.

   ```
   java -cp COHERENCE_HOME;COHERENCE_HOME\lib\coherence.jar
   com.tangosol.net.DefaultCacheServer
   ```

Using the Report Group Configuration File System Property

The `tangosol.coherence.management.report.configuration` system property is used to specify a custom report group configuration file to be used instead of the default `report-group.xml` file. The system property provides an easy way to switch between different configurations and is convenient during development and testing.

To specify a custom report group configuration file, enter the name of the file as a value of the `tangosol.coherence.management.report.configuration` system property. This is typically done as a `-D` Java option when starting a Coherence node. If the file is not located in the classpath, enter the full (or relative) path to the file and the name. The system property also supports the use of a URL when specifying the location of a report group configuration file.

The following example starts a cache server, enables JMX management, and specifies a report group configuration file that is named `my-group.xml` and is located in *COHERENCE_HOME*.

```
java -Dtangosol.coherence.management=all
-Dtangosol.coherence.management.report.configuration=my-group.xml -cp COHERENCE_
HOME;COHERENCE_HOME\lib\coherence.jar com.tangosol.net.DefaultCacheServer
```

Specifying an MBean Configuration File

The `tangosol-coherence.xml` operational deployment descriptor defines an operational override file that is named `custom-mbeans.xml` and is specifically used to define custom MBeans. A name and location of the override file may also be specified using the MBean configuration file system property.

Using the Default MBean Configuration Override File

Custom MBeans are defined within an override file named `custom-mbeans.xml`. At run time, Coherence uses the first instance of `custom-mbeans.xml` that is found in the classpath.

To use the default MBean configuration override file:

1. Create a file named `custom-mbeans.xml`.

2. Edit the file and create an empty <mbeans> node as follows:

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

<mbeans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
</mbeans>
```

3. Save and close the file.
4. Make sure the location of the custom MBean configuration override file is located in the classpath at run time.

   The following example demonstrates starting a cache server that uses a default MBean configuration override file that is located in *COHERENCE_HOME*.

```
java -cp COHERENCE_HOME;COHERENCE_HOME\lib\coherence.jar
com.tangosol.net.DefaultCacheServer
```

Using the MBean Configuration File System Property

The `tangosol.coherence.mbeans` system property specifies an MBean configuration override file to be used instead of the default `custom-mbeans.xml` override file. The system property provides an easy way to switch between different MBean configurations and is convenient during development and testing.

To specify an MBean configuration override file, enter the name of the file as a value of the `tangosol.coherence.mbeans` system property. This is typically done as a `-D` Java option when starting a Coherence node. If the file is not located in the classpath, enter the full (or relative) path to the file and the name. The system property also supports the use of a URL when specifying the location of an MBean configuration override file.

The following example starts a cache server and specifies an MBean configuration override file that is named `my-mbeans.xml` and is located in *COHERENCE_HOME*.

```
java -Dtangosol.coherence.mbeans=my-mbeans.xml -cp COHERENCE_HOME;COHERENCE_
HOME\lib\coherence.jar com.tangosol.net.DefaultCacheServer
```

Viewing Which Management Configuration Files are Loaded

The output for a Coherence node indicates the location and name of the report group configuration file and the MBean configuration file that are loaded at startup. The output is especially helpful when developing and testing Coherence applications and solutions.

Report Group Configuration File

The report group configuration messages are among the messages that display after the Coherence copyright text is emitted.

The following example output demonstrates a report group configuration message that indicates the `my-group.xml` file is loaded:

```
Loaded Reporter configuration from "file:/D:/coherence/my-group.xml
```

MBean Configuration Override File

The MBean configuration message is emitted with the other operational override messages and is among the first messages to be emitted when starting a process. The output is especially helpful when using override files and is often useful when developing and testing Coherence applications and solutions.

The following example output demonstrates an operational override message that indicates the default MBean configuration override file is loaded:

```
Loaded operational overrides from resource "file:/D:/coherence/custom-mbeans.xml"
```

Disabling Schema Validation

Coherence uses schema validation to ensure that configuration files adhere to their respective schema definition. Configuration files that include a schema reference are automatically validated against the schema when the configuration file is loaded. A validation error causes an immediate failure and an error message is emitted that indicates which element caused the error. Schema validation should always be used as a best practice.

Schema validation can be disabled if required. To disable schema validation, remove the `xsi:schemaLocation` attribute from a configuration file. The following example creates a `tangosol-coherence-override.xml` file that does not contain a schema reference and is not validated when loaded:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config">
   ...
</coherence>
```

Note: When schema validation is disabled, Coherence only fails if the XML is malformed. Syntactical errors are ignored and may not be immediately apparent.

Understanding the XML Override Feature

The XML override feature is a configuration mechanism that allows any operational settings to be changed without having to edit the default `tangosol-coherence.xml` operational deployment descriptor that is located in the `coherence.jar`. This mechanism is the preferred way of configuring the Coherence operational run time.

The XML override feature works by associating an XML document, commonly referred to as an override file, with a specific operational XML element. The XML element, and any of its subelements, are then modified as required in the override file. At run time, Coherence loads the override file and its elements replace (or are added to) the elements that are in the `tangosol-coherence.xml` file.

An override file does not have to exist at run time. However, if the override file does exist, then its root element must match the element it overrides. In addition, Subelements are optional. If a subelement is not defined in the override file, it is loaded from the `tangosol-coherence.xml` file. Typically, only the subelements that are being changed or added are placed in the override file.

The following topics are included in this section:

- Using the Predefined Override Files
- Defining Custom Override Files
- Defining Multiple Override Files for the Same Element

Using the Predefined Override Files

Two override files are predefined and can override elements in the operational deployment descriptor. These files must be manually created and saved to a location in the classpath.

- `tangosol-coherence-override.xml` – This override file is defined for the `<coherence>` root element and is used to override any element in the operational deployment descriptor. The root element in this file must be the `<coherence>` element.
- `custom-mbeans.xml` – This override file is defined for the `<mbeans>` element and is used to add custom MBeans to the operational deployment descriptor. The root element in this file must be the `<mbeans>` element.

The following example demonstrates a `tangosol-coherence-override.xml` file that is used to override the default cluster name. All other operational settings are loaded from the `tangosol-coherence.xml` file.

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <member-identity>
         <cluster-name system-property="tangosol.coherence.cluster">MyCluster
         </cluster-name>
      </member-identity>
   </cluster-config>
</coherence>
```

The following example demonstrates a `tangosol-coherence-override.xml` file that is used to disable local storage for the distributed cache service on this node. Notice the use of an `id` attribute to differentiate an element that can have multiple occurrences. The `id` attribute must match the `id` attribute of the element being overridden.

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <services>
         <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>
```

The following example demonstrates a `custom-mbean.xml` file that adds a standard MBean definition to the list of MBeans.

```
<mbeans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <mbean id="100">
      <mbean-class>com.oracle.customMBeans.Query</mbean-class>
      <mbean-name>type=Query</mbean-name>
      <enabled>true</enabled>
   </mbean>
</mbeans>
```

Defining Custom Override Files

Any element in the `tangosol-coherence.xml` deployment descriptor can be overridden using the predefined `tangosol-coherence-override.xml` file. However, there may be situations where more fine-grained configuration control is required. For example, a solution may want to allow changes to certain elements, but does not want to allow changes to the complete operational deployment descriptor. As another example, a solution may want to provide different configurations based on different use cases. Custom override files are used to support these types of scenarios.

Using the xml-override and id attributes

Override files are defined using the `xml-override` attribute and, if required, the `id` attribute. Both of these attributes are optional and are added to the operational element that is to be overridden. See "Attribute Reference" on page A-85 for a list of the operational elements that support the use of an override file.

The value of the `xml-override` attribute is the name of a document that is accessible to the classes contained in the `coherence.jar` library using the `ClassLoader.getResourceAsStream(String name)` method. In general, the file name contains a `/` prefix and is located in the classpath at run time. The attribute also supports the use of a URL when specifying the location of an override file.

For example, to define an override file named `cluster-config.xml` that is used to override the `<cluster-config>` element, add an `xml-override` attribute to the `<cluster-config>` element in the `tangosol-coherence-override.xml` file as shown below:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config xml-override="/cluster-config.xml">
   ...
   </cluster-config>
</coherence>
```

To use this override file, create a document named `cluster-config.xml` and ensure that it and the base document (`tangosol-coherence-override.xml` in this case) are located in a directory that is in the classpath at run time. For this example, the override file's root element must be `<cluster-config>` as shown below.

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

<cluster-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <multicast-listener>
      <join-timeout-milliseconds>4000</join-timeout-milliseconds>
   </multicast-listener>
</cluster-config>
```

An `id` attribute is used to distinguish elements that can occur more multiple times.

For example, to define a custom override file named `dist-service-config.xml` that is used to override the `<service>` element for the distributed cache service, add an `xml-override` attribute to the `<service>` element whose `id` is number `3` as shown below

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <services>
         <service id="3" xml-override="/dist-service-config.xml">
         </service>
      </services>
   </cluster-config>
</coherence>
```

To use this override file, create a document named `dist-service-config.xml` and ensure that it is located in a directory that is in the classpath at run time. For this example, the override file's root element must be `<service>` as shown below.

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

<service id="3" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <init-params>
      <init-param id="1">
         <param-name>standard-lease-milliseconds</param-name>
         <param-value>2</param-value>
      </init-param>
   </init-params>
</service>
```

Note: If the element's `id` in the override document does not have a match in the base document, the elements are just appended to the base document.

Defining Multiple Override Files for the Same Element

Multiple override files can be defined for the same element to chain operational override files. This is typically done to allow operational configurations based on different deployment scenarios, such as staging and production.

As an example, the `tangosol-coherence.xml` operational deployment descriptor located in `coherence.jar` defines an operational override file for the `<coherence>` element as follows:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd"
   xml-override="{tangosol.coherence.override/tangosol-coherence-override-{mode}
   .xml}">
   ...
</coherence>
```

The mode-specific override files are also located in `coherence.jar` and are used depending on the Coherence start mode (the value of the `<license-mode>` element). Each of the mode-specific operational override files, in turn, defines the default operational override file as follows:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd"
   xml-override="/tangosol-coherence-override.xml">
   ...
</coherence>
```

A fourth override file can be defined for the `<coherence>` element in the `tangosol-coherence-override.xml` file. For example:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd"
   xml-override="/tangosol-coherence-override-staging.xml">
   ...
</coherence>
```

The chain can continue as required. The files are all loaded at run time if they are placed in a location in the classpath. Files higher up in the chain always override files below in the chain.

Changing Configuration Using System Properties

The command-line override feature allows operational and cache settings to be overridden using system properties. System properties are typically specified on the Java command line using the Java `-D` option. This allows configuration to be customized for each node in a cluster while using the same operational configuration

file and cache configuration file across the nodes. System properties are also a convenient and quick way to change settings during development.

The following topics are included in this section:

- Using Preconfigured System Properties
- Creating Custom System Properties

Using Preconfigured System Properties

Coherence includes many preconfigured system properties that are used to override different operational and cache settings. Table C–1 lists all the preconfigured system properties. The preconfigured system properties are defined within the `tangosol-coherence.xml` and `coherence-cache-config.xml` default deployment descriptors, respectively, using `system-property` attributes.

For example, the preconfigured `tangosol.coherence.log.level` system property is defined in the `tangosol-coherence.xml` file as follows:

```
<logging-config>
   ...
   <severity-level system-property="tangosol.coherence.log.level">5
   </severity-level>
   ...
</logging-config>
```

To use a preconfigured system property, add the system property as a Java `-D` option at startup. For the above example, the log level system property is specified as follows when starting a cache server:

```
java -Dtangosol.coherence.log.level=3 -cp COHERENCE_HOME\lib\coherence.jar
com.tangosol.net.DefaultCacheServer
```

Note: When using an operational override file and when creating a custom cache configuration file; the preconfigured system properties must always be included along with the element that is to be overridden; otherwise, the property is no longer available.

Creating Custom System Properties

Custom system properties can be created for any operational or cache configuration element. The names of the preconfigured system properties can also be changed as required.

System properties are defined by adding a `system-property` attribute to the element that is to be overridden. The value of the `system-property` attribute can be any user-defined name. Custom system properties are typically defined in an operational override file (such as `tangosol-coherence-override.xml`) and a custom cache configuration file.

Defining a System Property for an Operational Element

The following example defines a system property called `multicast.join.timeout` for the `<join-timeout-milliseconds>` operational element and is added to an operational override file:

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

```
<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <multicast-listener>
         <join-timeout-milliseconds system-property="multicast.join.timeout">30000
         </join-timeout-milliseconds>
      </multicast-listener>
   </cluster-config>
</coherence>
```

Defining a System Property for a Cache Configuration element

The following example defines a system property called `cache.name` for a `<cache-name>` element and is added to a custom cache configuration file:

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <caching-scheme-mapping>
    <cache-mapping>
      <cache-name system-property="cache.name"></cache-name>
      ...
```

Changing a Preconfigured System Property

The following example changes the preconfigured system property name for the `<cluster-name>` operational element and is added to an operational override file:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <member-identity>
         <cluster-name system-property="myapp.cluster.name"></cluster-name>
      </member-identity>
   </cluster-config>
</coherence>
```

> **Note:** To remove a system property, delete the system property attribute from the element. If a system property is used at run time and it does not exist, it is disregarded.

4

Building Your First Coherence Application

This chapter provides step-by-step instructions for building and running a basic Coherence example and demonstrates many fundamental Coherence concepts. The sample application is a simple Hello World application and is implemented both as a standalone Java application and a JSP application. Lastly, a JDeveloper section has been included that provides some basic instructions for setting up JDeveloper when developing with Coherence.

> **Note:** The example in this chapter is basic and is only intended to teach general concepts. For more advanced examples, download the *Coherence Examples* included with the documentation library.

The following sections are included in this chapter:

Task 1: Define the Example Cache

Caches are defined in a cache configuration deployment descriptor and are referred to by name within an application. This allows configuration changes to be made to a cache without having to change an application's code. The following cache configuration defines a basic distributed cache which is mapped to the cache name `hello-example`.

To define the example cache:

1. Create an XML file named `example-config.xml`.
2. Copy the following distributed cache definition to the file:

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <caching-scheme-mapping>
      <cache-mapping>
```

```
            <cache-name>hello-example</cache-name>
            <scheme-name>distributed</scheme-name>
         </cache-mapping>
      </caching-scheme-mapping>

      <caching-schemes>
         <distributed-scheme>
            <scheme-name>distributed</scheme-name>
            <service-name>DistributedCache</service-name>
            <backing-map-scheme>
               <local-scheme/>
            </backing-map-scheme>
            <autostart>true</autostart>
         </distributed-scheme>
      </caching-schemes>
   </cache-config>
```

3. Save and close the file.

Task 2: Configure and Start the Example Cluster

Caches are hosted on a Coherence cluster. At run time, any JVM process that is running Coherence automatically joins the cluster and can access the caches and other services provided by the cluster. When a JVM joins the cluster, it is called a cluster node, or alternatively, a cluster member. For the sample applications in this chapter, two separate Java processes form the cluster: a cache server process and the Hello World application process. For simplicity, the two processes are collocated on a single computer. The cache server, by default, is configured to store cache data.

The example cluster uses an operational override file to modify the out-of-box default cluster configuration. In particular, the default configuration is modified to create a private cluster which ensures that the two processes do not attempt to join an existing Coherence cluster that may be running on the network. The default configuration is also modified to load the `example-config.xml` cache configuration file instead of the default cache configuration file.

To configure and start the example cluster:

1. Create a file named `tangosol-coherence-override.xml`.
2. Add the following override configuration and replace *`cluster_name`* and *`port`* with values that are unique for this cluster. For example, use your name for the cluster name and the last four digits of your phone number for the port.

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <member-identity>
         <cluster-name>cluster_name</cluster-name>
      </member-identity>

      <multicast-listener>
         <address>224.3.6.0</address>
         <port>port</port>
         <time-to-live>0</time-to-live>
      </multicast-listener>
```

```
   </cluster-config>

   <configurable-cache-factory-config>
      <init-params>
         <init-param>
            <param-type>java.lang.String</param-type>
            <param-value system-property="tangosol.coherence.cacheconfig">
               example-config.xml</param-value>
         </init-param>
      </init-params>
   </configurable-cache-factory-config>
</coherence>
```

3. Save the file to the same directory where the `example-config.xml` file was saved.
4. From a command prompt, start a cache server instance using the `DefaultCacheServer` class and include the location of the `coherence.jar` library and the configuration files as a Java `-cp` option. For example:

```
java -cp COHERENCE_HOME\config;COHERENCE_HOME\lib\coherence.jar
com.tangosol.net.DefaultCacheServer
```

Task 3: Create and Run a Basic Coherence Standalone Application

Step 3 is a multi-part step that includes a sample Hello World application and instructions for running and verifying the example. The application is run from the command line and starts a cache node that joins with a cache server. The application puts a key named `k1` with the value `Hello World!` into the `hello-example` cache and then gets and prints out the value of the key before exiting. Lastly, an additional cluster node is started to verify that the key is in the cache.

Create the Sample Standalone Application

Applications use the Coherence API to access and interact with a cache. The `CacheFactory` class is used to get an instance of a cache and the `NamedCache` interface is used to retrieve and store objects in the cache. The Hello World application is very basic, but it does demonstrate using the `CacheFactory` class and the `NamedCache` interface.

Example 4–1 The Sample HelloWorld Standalone Application

```
package com.examples;

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

public class HelloWorld {

   public static void main(String[] args) {

      String key = "k1";
      String value = "Hello World!";

      CacheFactory.ensureCluster();
      NamedCache cache = CacheFactory.getCache("hello-example");

      cache.put(key, value);
```

```
        System.out.println((String)cache.get(key));

        CacheFactory.shutdown();
    }
}
```

Run the Sample Standalone Application

To run the standalone application example:

1. From a command prompt, compile the Hello World application. For example:

   ```
   javac -cp COHERENCE_HOME\lib\coherence.jar com\examples\HelloWorld.java
   ```

2. Run the Hello World application and include the location of the `coherence.jar` library and the configuration files as a Java `-cp` option. In addition, restrict the client from locally storing partitioned data. For example:

   ```
   java -cp COHERENCE_HOME\config;COHERENCE_HOME\lib\coherence.jar
   -Dtangosol.coherence.distributed.localstorage=false com.example.HelloWorld
   ```

   The Hello World application starts. The cache factory instance is created and becomes a member of the cluster. The `k1` key with the `Hello World!` value is loaded into the `hello-example` cache. The key is then retrieved from the cache and the value is emitted as part of the output. Lastly, the cache factory is shutdown and leaves the cluster before the Hello World application exits.

Verify the Example Cache

The cache server in this example is configured, by default, to store the cache's data. The data is available to all members of the cluster and persists even after members leave the cluster. For example, the Hello World application exits after it loads and displays a key in the cache. However, the cache and key are still available for all cluster members.

This step uses the cache factory command-line tool to connect to the `hello-example` cache and list all items in the cache. It demonstrates both the persistent and distributed nature of Coherence caches.

To verify the cache:

1. From a command prompt, start a standalone cache factory instance using the `CacheFactory` class and include the location of the `coherence.jar` library and the configuration files as a Java `-cp` option. For example:

   ```
   java -cp COHERENCE_HOME\config;COHERENCE_HOME\lib\coherence.jar
   -Dtangosol.coherence.distributed.localstorage=false
   com.tangosol.net.CacheFactory
   ```

   The cache factory instance starts and becomes a member of the cluster and returns a command prompt for the command-line tool.

2. At the command-line tool command prompt, get the `hello-example` cache using the `cache` command:

   ```
   cache hello-example
   ```

3. At the command-line tool command prompt, retrieve the contents of the cache using the `list` command.

   ```
   list
   ```

The command returns and displays:

```
k1 = Hello World!
```

Task 4: Create and Run a Basic Coherence JavaEE Web Application

Step 4 is a multi-part step that includes the Hello World application re-implemented as a JSP page. Instructions are included for packaging the sample as a Web application to be deployed to a JavaEE server. The application runs on the application server and starts a cache node that joins with a cache server. The application puts a key named `k2` with the value `Hello World!` into the `hello-example` cache and then gets and prints out the value of the key before exiting. Lastly, an additional cluster node is started to verify that the key is in the cache.

Create the Sample Web Application

To create the sample Web application:

1. Create a basic Web application directory structure as follows:

```
/
/WEB-INF
/WEB-INF/classes
/WEB-INF/lib
```

2. Copy the below JSP to a text file and save the file as `hello.jsp` in the root of the Web application directory.

Example 4–2 The Sample Hello World JSP

```
<html>
   <head>
      <title>My First Coherence Cache</title>
   </head>
   <body>
      <h1>
         <%@ page language="java"
                  import="com.tangosol.net.CacheFactory,
                          com.tangosol.net.NamedCache"
         %>
         <%
            String key = "k2";
            String value = "Hello World!";

            CacheFactory.ensureCluster();
            NamedCache cache = CacheFactory.getCache("hello-example");

            cache.put(key, value);
            out.println((String)cache.get(key));

            CacheFactory.shutdown();
         %>
      </h1>
   </body>
</html>
```

3. Copy the following empty Web application deployment descriptor to a text file and save the file as `web.xml` in the `/WEB-INF` directory.

```
<?xml version = '1.0' ?>
   <web-app/>
```

4. Copy the `coherence.jar` file to the `WEB-INF/lib` directory.
5. Copy the `example-config.xml` file and the `tangosol-coherence-override.xml` file to the `WEB-INF/classes` directory.
6. Create a Web ARchive file (WAR) using the `jar` utility and save the file as `hello.war`. For example, issue the following command from a command prompt at the root of the Web application directory:

   ```
   jar -cvf hello.war *
   ```

   The archive should contain the following files

   ```
   /hello.jsp
   /WEB-INF/web.xml
   /WEB-INF/classes/example-config.xml
   /WEB-INF/classes/tangosol-coherence-override.xml
   /WEB-INF/lib/coherence.jar
   ```

Deploy and Run the Sample Web Application

To deploy and run the Web application example:

1. Deploy the `hello.war` file to a JavaEE server.
2. From a browser, run the Hello World application by accessing the `hello.jsp` file using the following URL. Substitute *`host`* and *`port`* with values specific to the deployment.

   ```
   http://host:port/hello/hello.jsp
   ```

   The Hello World application starts. The cache factory instance is created and becomes a member of the cluster. The `k2` key with the `Hello World!` value is loaded into the `hello-example` cache. The key is then retrieved from the cache and the value is displayed in the browser. Lastly, the cache factory shuts down and leaves the cluster.

Verify the Example Cache

The cache server in this example is configured, by default, to store the cache's data. The data is available to all members of the cluster and persists even after members leave the cluster. For example, the Hello World application exits after it loads and displays a key in the cache. However, the cache and key are still available for all cluster members.

This step uses the cache factory command-line tool to connect to the `hello-example` cache and list all items in the cache. It demonstrates both the persistent and distributed nature of Coherence caches.

To verify the cache:

1. From a command prompt, start a standalone cache factory instance using the `CacheFactory` class and include the location of the `coherence.jar` library and the configuration files as a Java `-cp` option. For example:

   ```
   java -cp COHERENCE_HOME\config;COHERENCE_HOME\lib\coherence.jar
   -Dtangosol.coherence.distributed.localstorage=false
   com.tangosol.net.CacheFactory
   ```

The cache factory instance starts and becomes a member of the cluster and returns a command prompt for the command-line tool.

2. At the command-line tool command prompt, get the `hello-example` cache using the `cache` command:

```
cache hello-example
```

3. At the command-line tool command prompt, retrieve the contents of the cache using the `list` command.

```
list
```

The command returns and displays:

```
k2 = Hello World!
```

Using JDeveloper for Coherence Development

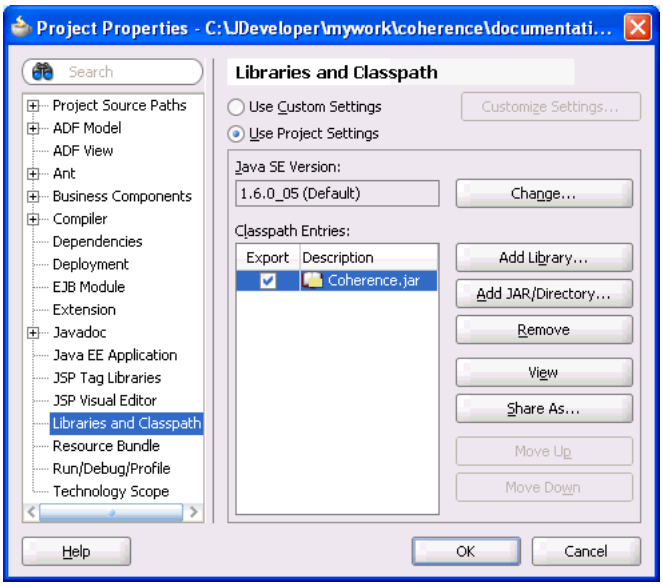
This section provides basic instructions on how to setup JDeveloper for Coherence development. The instructions are for running Coherence within the IDE which is a common approach during development. While the instructions are specific to JDeveloper, the same approach should be possible with any IDE. See your IDE's documentation for specific instructions.

Running Coherence in JDeveloper

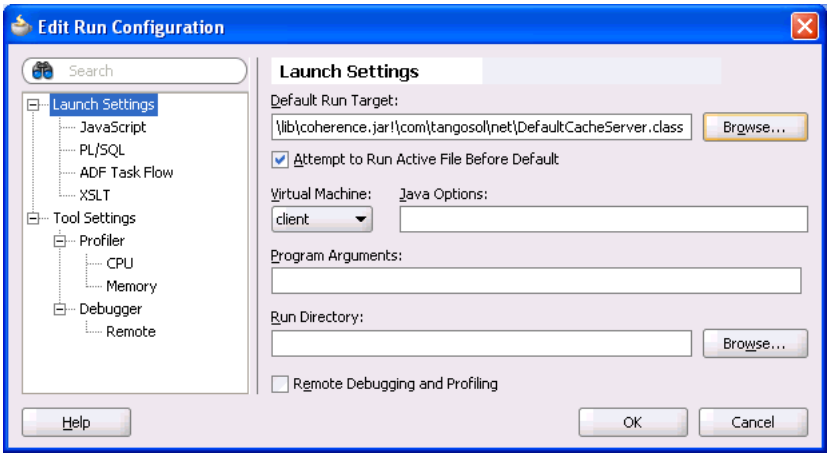
JDeveloper can run cache server (`DefaultCacheServer`) and cache (`CacheFactory`) instances. Each instance is started as a separate Java process and emits standard output to the process' log. Input (such as cache commands) can be entered directly in the process as if it were started from the command line. This configuration facilitates development and testing Coherence solutions.

To run Coherence in JDeveloper:

1. In JDeveloper, create a new Generic Application, which includes a single project. If you are new to JDeveloper, consult the Online Help for detailed instructions.
2. In the Application Navigator, double-click the new project. The Project Properties dialog box displays.
3. Select the **Libraries and Classpath** node. The Libraries and Classpath page displays
4. On the Libraries and Classpath page, click **Add JAR/Directory**. The Add Archive or Directory dialog box displays.
5. From the directory tree, select *COHERENCE_HOME*`\lib\coherence.jar` and click **Select**. The `coherence.jar` library displays in the Classpath Entries list as shown below:

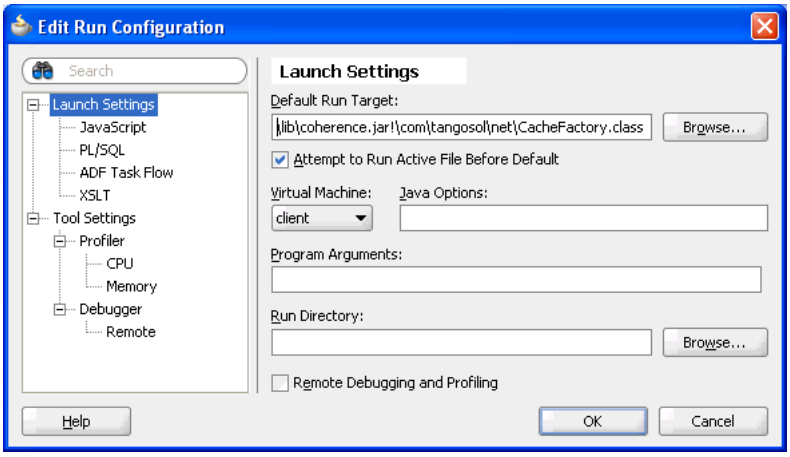


6. From the Project Properties dialog box, select the **Run/Debug/Profile** node. The Run/Debug/Profile page displays.
7. From the Run/Debug/Profile page, click **New**. The Create Run Configuration dialog box displays.
8. In the Name text box, enter a name for the new run configuration. In the Copy Settings From drop-down box, choose **default**. Click **OK**. The new run configuration displays in the Run Configuration list.
9. From the Run Configuration list, select the new Run Configuration and click **Edit**. The Edit Run Configuration dialog box displays and the Launch Settings node is selected.
10. From the Launch Settings page, click **Browse** to select a Default Run Target. The Choose Default Run Target dialog box displays.
11. From the directory tree, select *COHERENCE_HOME*`\lib\coherence.jar\com\tangosol\net\DefaultCacheServer.class` and click **Open**. The `DefaultCacheServer` class is entered as the default run target as shown below:

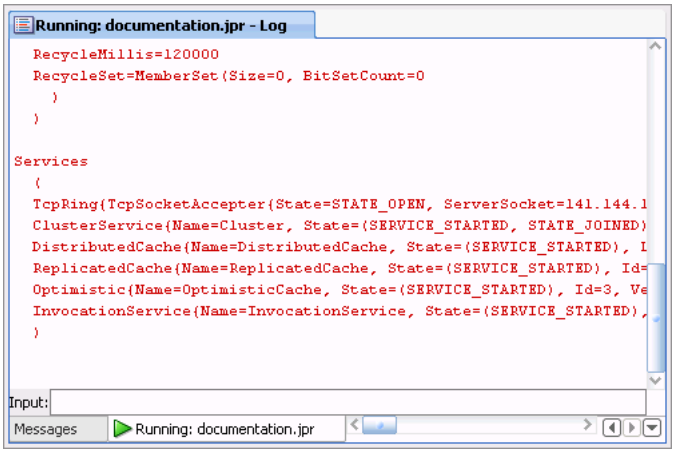


Tip: Use the Java Options text box to set Coherence system properties.

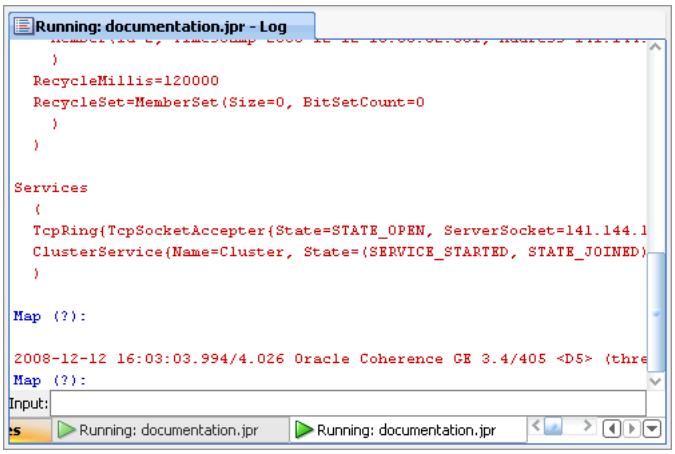
12. Select the Tool Settings Node. The Tool Settings page displays.
13. From the Additional Runner Options section, click the **Allow Program Input** check box. A check mark in the box indicates that the option is selected.
14. Click **OK**.
15. Repeat Steps 6 through 14 and select *COHERENCE_HOME*`\lib\coherence.jar\com\tangosol\net\CacheFactory.class` as the default run target as shown below:



16. Click **OK** to close the Project Properties dialog box.
17. Use the Run button drop-down list to select and start the run configuration for the cache server. A cache server instance is started and output is shown in the process's log tab as shown below:



18. Use the Run button drop-down list to select and start the run configuration for the cache. A cache instance is started and output is shown in the process's log tab as shown below.



19. From the Cache Factory's Running Log tab, use the Input text box located at the bottom of the tab to interact with the cache instance. For example, type `help` and press **Enter** to see a list of valid commands.

Viewing Thread Dumps in JDeveloper

Java can dump a list of threads and all their held locks to standard out. This is achieved in Linux environments using the `kill` command and in Windows environments using `ctrl+break`. Thread dumps are very useful for troubleshooting during development (for example, finding deadlocks).

When developing Coherence solutions in JDeveloper, you can view thread dumps directly in a process's log tab. This is achieved, by sending the above signals to the Java process running in JDeveloper.

To view thread dumps in JDeveloper:

1. From a shell or command prompt, use `JDK_HOME/bin/jps` to get the Process ID (PID) of the Java process for which you want to view a thread dump.
2. On Linux, use `kill -3` *PID* to send a `QUIT` signal to the process. On Windows, you must use a third-party tool (such as SendSignal) to send a `ctrl+break` signal to a remote Java process.

   The thread dump is viewable in the process's log in JDeveloper.

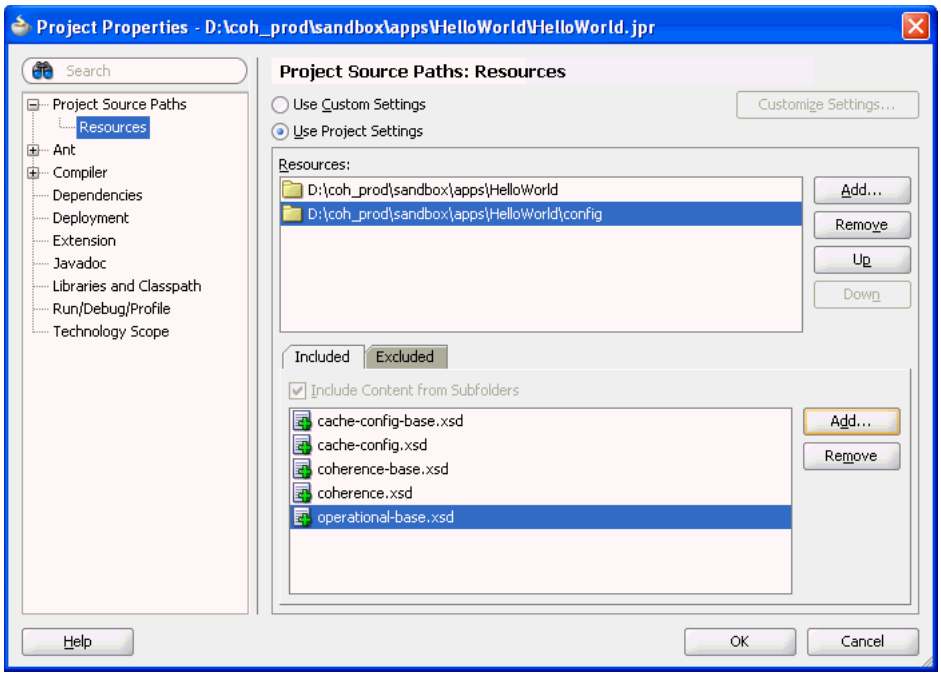
Creating Configuration Files in JDeveloper

JDeveloper can create Coherence configuration files. JDeveloper loads the appropriate XSD files and lists all the elements in the Component Palette. In addition, JDeveloper validates configuration files against the XSD and provides XML code completion. The following procedure creates both a cache configuration file and an operational override file. The same procedure can be used for any of the Coherence configuration files.

To create a cache configuration and operation override file in JDeveloper:

1. Extract `coherence-cache-config.xsd`, `coherence-cache-config-base.xsd`, `coherence-operational-config.xsd`, `coherence-operational-config-base.xsd`, and `coherence-config-base.xsd` from the *COHERENCE_HOME*`\lib\coherence.jar` library to a directory on your computer.

2. In the JDeveloper Application Navigator, double-click your coherence project. The Project Properties dialog box displays.
3. Expand the **Project Source Paths** node and click **Resources**. The Resources page displays.
4. In the Resources section, click **Add** to find and select the directory where you extracted the XSD files.
5. In the Included tab, click **Add** and select the XSD files. Alternatively, you can allow JDeveloper to include all files in this directory and not explicitly add each file.
6. Click **OK**. The XSDs are listed in the Included tab as shown below.



7. Click **OK** to close the Project Properties dialog box. The XSDs are listed in the Application Navigator under the Resources folder for your project.
8. From the File menu, click **New**. The New Gallery dialog box displays.
9. From the Categories section, expand the **General** node and click **XML**.
10. Select **XML Document** and click **OK**. The Create XML File dialog box displays.
11. Enter `coherence-cache-config.xml` as the file name and save it to the same directory where the XSD is located. At run time, this file must be found on the classpath and must be loaded before the `coherence.jar` file.
12. Click **OK**. The cache configuration file is created, opened for editing, and listed in the Application Navigator under the resources folder for your project.
13. Add the following schema reference at the beginning of the file:

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
```

The Component Palette refreshes and lists all the elements available from the `coherence-cache-config.xsd` file.

14. Save the `coherence-cache-config.xml` file.
15. Repeat steps 8 through 12 to create an operational override file called `tangosol-coherence-override.xml`. At run time, this file must be found on the classpath.
16. Add the following schema references at the beginning of the file:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
```

The Component Palette refreshes and lists all the elements available from the `coherence-operational-config.xsd` file.

17. Save the `tangosol-coherence-override.xml` file.

5

Debugging in Coherence

This chapter provides instructions for configuring logging and general advice for troubleshooting Coherence applications.

The following sections are included in this chapter:

Overview of Debugging in Coherence

Coherence applications are typically developed on a single computer. The cache server and application are started within the IDE and the application is debugged as required. This type of development environment is easy to setup, performs well, and is easy to debug. A majority of applications can be created and tested this way. See "Enabling Single-Server Mode" on page 7-12 for details on configuring Coherence to run on a single server.

Ideally, most errors can be detected during development using logging, enabling JVM debug options, and capturing thread and heap dumps as required. Moreover, IDEs and profiling tools, such as Oracle's Java VisualVM, JConsole, and JRockit Mission Control, all provide features for diagnosing problems. However, Coherence applications must eventually be tested in a more distributed environment. Debugging and troubleshooting in the testing environment is more difficult since data and processes are fully distributed across the cluster and because the network can now affect the application. Remote debugging with Java Debug Wire Protocol (JDWP) together with Coherence's JMX management and reporting capabilities facilitate debugging and troubleshooting in a distributed environment.

Using Oracle Support

Oracle support can help debug issues and is available through `https://support.oracle.com`. When sending support an issue, always include the following items in a compressed file:

- application code
- configuration files
- log files for all cluster members
- Thread and heap dumps are required under certain circumstances. Thread dumps should be sent if the application is running slow and/or appears to be hung. Heap

dumps should be sent if the application runs out of memory or is consuming more memory than expected.

Configuring Logging

Coherence has its own logging framework and also supports the use of log4j and Java logging to provide a common logging environment for an application. Logging in Coherence occurs on a dedicated and low-priority thread to reduce the impact of logging on the critical portions of the system. Logging is pre-configured and the default settings should be changed as required.

The following topics are included in this section:

- Changing the Log Level
- Changing the Log Destination
- Changing the Log Message Format
- Setting the Logging Character Limit
- Using JDK Logging for Coherence Logs
- Using Log4J Logging for Coherence Logs

Changing the Log Level

The logger's log level determines which log messages are emitted. The default log level emits error, warning, informational, and some debug messages. During development, the log level should be raised to its maximum setting to ensure all debug messages are logged. The following log levels are available:

- `0` – This level includes messages that are not associated with a logging level.
- `1` – This level includes the previous level's messages plus error messages.
- `2` – This level includes the previous levels' messages plus warning messages.
- `3` – This level includes the previous levels' messages plus informational messages.
- `4-9` – These levels include the previous levels' messages plus internal debugging messages. More log messages are emitted as the log level is increased. The default log level is `5`.
- `-1` – No log messages are emitted.

To change the log level, edit the operational override file and add a `<severity-level>` element, within the `<logging-config>` element, that includes the level number. For example:

```
...
<logging-config>
   ...
   <severity-level system-property="tangosol.coherence.log.level">9
   </severity-level>
   ...
</logging-config>
...
```

The `tangosol.coherence.log.level` system property is used to specify the log level instead of using the operational override file. For example:

```
-Dtangosol.coherence.log.level=9
```

Changing the Log Destination

The logger can be configured to emit log messages to several destinations. For standard output to the console, both `stdout` and `stderr` (the default) can be used. The logger can also emit messages to a specified file.

Coherence also supports the use of both the JDK and log4j logging frameworks to allow an application and Coherence to share a common logging framework. See "Using JDK Logging for Coherence Logs" on page 5-5 and "Using Log4J Logging for Coherence Logs" on page 5-6 for detailed instructions on using these logging frameworks with Coherence.

To change the log destination, edit the operational override file and add a `<destination>` element, within the `<logging-config>` element, that includes the destination. For example:

```
...
<logging-config>
   <destination system-property="tangosol.coherence.log">stdout</destination>
   ...
</logging-config>
...
```

The `tangosol.coherence.log` system property is used to specify the log destination instead of using the operational override file. For example:

```
-Dtangosol.coherence.log=stdout
```

Sending Log Messages to a File

The logger can be configured to emit log messages to a file by providing a path and file name in the `<destination>` element. The specified path must already exist. Make sure the specified directory can be accessed and has write permissions. Output is appended to the file and there is no size limit. Process cannot share a log file and the log file is replaced when a process is restarted. Sending log messages to a file is typically used during development and testing and is useful if the log messages need to be sent to Oracle support.

The following example demonstrates specifying a log file named `coherence.log` that is written to the `/tmp` directory:

```
...
<logging-config>
   <destination system-property="tangosol.coherence.log">/tmp/coherence.log
   </destination>
   ...
</logging-config>
...
```

Changing the Log Message Format

The default format of log messages can be changed depending on the amount of detail that is required. A log message can include static text as well as any of the following parameters that are replaced at run time.

Note: Changing the log message format must be done with caution as critical information (such as member or thread) can be lost which makes issues harder to debug.

Parameter	Description
{date}	This parameter shows the date/time (to a millisecond) when the message was logged.
{uptime}	This parameter shows the amount of time that the cluster members has been operational.
{product}	This parameter shows the product name and license type.
{version}	This parameter shows Coherence version and build details.
{level}	This parameter shows the logging severity level of the message.
{thread}	This parameters shows the thread name that logged the message.
{member}	This parameter shows the cluster member id (if the cluster is currently running).
{location}	This parameter shows the fully cluster member identification: cluster-name, site-name, rack-name, machine-name, process-name and member-name (if the cluster is currently running).
{role}	This parameter shows the specified role of the cluster member.
{text}	This parameters shows the text of the message.

To change the log message format, edit the operational override file and add a <message-format> element, within the <logging-config> element, that includes the format. For example:

```
...
<logging-config>
   ...
   <message-format>[{date}] &lt;{level}&gt; (thread={thread})  -->{text}
   </message-format>
   ...
</logging-config>
...
```

Setting the Logging Character Limit

The logging character limit specifies the maximum number of characters that the logger daemon processes from the message queue before discarding all remaining messages in the queue. The messages that are discarded are summarized by the logging system with a single log entry that details the number of messages that were discarded and their total size. For example:

```
Asynchronous logging character limit exceeded; discarding 5 log messages
(lines=14, chars=968)
```

The truncation is only temporary; when the queue is processed (emptied), the logger is reset so that subsequent messages are logged.

Note: The message that caused the total number of characters to exceed the maximum is never truncated.

The character limit is used to avoid situations where logging prevents recovery from a failing condition. For example, logging can increase already tight timings, which causes additional failures, which produces more logging. This cycle may continue until recovery is not possible. A limit on logging prevents the cycle from occurring.

To set the log character limit, edit the operational override file and add a `<character-limit>` element, within the `<logging-config>` element. The character limit is entered as `0` (`Integer.MAX_VALUE`) or a positive integer. For example:

```
...
<logging-config>
   ...
   <character-limit system-property="tangosol.coherence.log.limit">12288
   </character-limit>
</logging-config>
...
```

The `tangosol.coherence.log.limit` system property is used to specify the log character limit instead of using the operational override file. For example:

```
-Dtangosol.coherence.log.limit=12288
```

Using JDK Logging for Coherence Logs

Applications that use the JDK logging framework can configure Coherence to use JDK logging as well. Detailed information about JDK logging is beyond the scope of this documentation. For details on JDK logging, see http://download.oracle.com/javase/6/docs/technotes/guides/logging/overview.html.

To use JDK logging for Coherence logs:

1. Create a `logging.properties` file. The following example configures the JDK logger to emit messages to both the console and to a file. Output to the console is configured to use the global log level (`INFO`) and output to the file is configured to use the `FINEST` log level. For the file handler pattern, the specified path must already exist. Also, make sure the specified directory can be accessed and has write permissions.

   ```
   handlers=java.util.logging.FileHandler, java.util.logging.ConsoleHandler

   .level=INFO

   java.util.logging.FileHandler.pattern=/tmp/coherence%u.log
   java.util.logging.FileHandler.limit=50000
   java.util.logging.FileHandler.level=FINEST
   java.util.logging.FileHandler.count=1
   java.util.logging.FileHandler.formatter=java.util.logging.SimpleFormatter

   java.util.logging.ConsoleHandler.formatter=java.util.logging.SimpleFormatter
   ```

2. Configure Coherence to use JDK logging by specifying `jdk` as the value of the `<destination>` element in an operational override file. For example:

   ```
   ...
   ```

```
<logging-config>
   <destination system-property="tangosol.coherence.log">jdk</destination>
   ...
</logging-config>
...
```

3. Make sure the `logging.properties` file is found on the classpath at run time or is specified using the `java.util.logging.config.file` system property. For example:

```
-Djava.util.logging.config.file=myfile
```

Using Log4J Logging for Coherence Logs

Applications that use the log4j logging framework can configure Coherence to use log4j logging as well. Detailed information about log4j logging is beyond the scope of this documentation. For details on log4j logging, see http://logging.apache.org/log4j/1.2/manual.html.

To use log4j logging for Coherence logs:

1. Create a `log4j.properties` file. The following example configures the log4j logger to emit messages to both the console and to a file. Output to the console is configured to use the global log level (`INFO`) and output to the file is configured to use the `DEBUG` log level. For the file appender, make sure the specified directory can be accessed and has write permissions.

```
log4j.logger.Coherence=INFO, stdout, file

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%m%n

log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=/tmp/coherence.log
log4j.appender.file.threshold=DEBUG
log4j.appender.file.MaxFileSize=10MB
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%m%n
```

> **Note:** In the above example, `Coherence` is used as the logger object name and is the default name used by Coherence. A different name can be used by specifying the name within the `<logger-name>` element in the operational override file or by specifying the name as the value of the `tangosol.coherence.log.logger` system property.

2. Configure Coherence to use log4j logging by specifying `log4j` as the value of the `<destination>` element in an operational override file. For example:

```
...
<logging-config>
   <destination system-property="tangosol.coherence.log">log4j</destination>
   ...
</logging-config>
...
```

3. Make sure both the `log4j.jar` file and the `log4j.properties` file are found on the classpath at run time.

Performing Remote Debugging

Java Debug Wire Protocol (JDWP) provides the ability to debug a JVM remotely. Most IDE tools support JDWP and are used to connect to a remote JVM that has remote debugging enabled. See your IDE's documentation for instructions on how to connect to a remote JVM.

To enable remote debugging on a cache server, start the cache server with the following JVM options. Once the cache server has been started, use the IDE's debugger to connect to the JVM using the port specified (`5005` in the example).

```
-Xdebug
-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5005
```

Remote debugging a Coherence application can be difficult when the application is no longer on a single node cluster because data is distributed across the members of the cluster. For example, when performing parallel grid operations, the operations are performed on the cluster members where the data is located. Since there are no guarantees on which members the data is located, it is best to constrain a test to use a singe cache server.

In addition, the guardian and packet timeout can make cluster debugging difficult. If the debugger pauses the packet publishing, cluster, and service threads, it will cause disruptions across the cluster. In such scenarios, disable the guardian and increase the packet timeout during the debugging session. See "service-guardian" on page A-69 for details on configuring these settings.

Troubleshooting Coherence-Based Applications

The topics in this section provide general troubleshooting advice. Troubleshooting Coherence-based applications is, for the most part, no different than troubleshooting other Java application. Most IDEs provide features that facilitate the process. In addition, many tools, such as: Java VisualVM, JConsole, JRockit Mission Control, and third party tools provide easy ways to monitor and troubleshoot Java applications. See the Troubleshooting Java SE section on OTN for detailed information on troubleshooting Java:

`http://www.oracle.com/technetwork/java/javase/index-138283.html`

Troubleshooting a Coherence application on a single server cluster is typically straight forward. Most Coherence development work is done in such an environment because it facilitates debugging. Troubleshooting an application that is deployed on a distributed cluster can become more challenging.

The following topics are included in this section:

- Using Coherence Logs
- Using JMX Management and Coherence Reports
- Using JVM Options to Help Debug
- Capturing Thread Dumps
- Capturing Heap Dumps
- Monitoring the Operating System

Using Coherence Logs

Log messages provide information that is used to monitor and troubleshoot Coherence. Most log messages are explained in the Log Glossary within the *Oracle Coherence Administrator's Guide*. The glossary provides additional details as well as specific actions that can be taken when a message is encountered.

Configuring logging beyond the default out-of-box configuration is very important when developing and debugging an application. Specifically, use the highest log level (level 9 or ALL when using JDK or log4j logging) to ensure that all log messages are emitted. Also, consider using either JDK or log4j logging. Both of these frameworks support the use of rolling files and console output simultaneously. Lastly, consider placing all log files in a common directory. A common directory makes it easier to review the log files and package them for the Coherence support team. See "Configuring Logging" on page 5-2 for detailed information on configuring all aspects of logging.

Using JMX Management and Coherence Reports

Coherence management is implemented using Java Management Extensions (JMX). Many MBeans are provided that detail the health and stability of Coherence. The MBeans provide valuable insight and should always be used when moving an application from a development environment to a fully distributed environment. MBeans are accessible using JConsole and VisualVM or any management tool that supports JMX. In addition, Coherence includes reports that gather information from the MBeans over time and provide a historical context that is not possible simply by monitoring the MBeans. The reports are most often used to identify trends that are valuable for troubleshooting. Management and reporting are not enabled by default and must be enabled. See *Oracle Coherence Management Guide* for detailed instructions on using the management features included with Coherence.

Using JVM Options to Help Debug

Most JVMs include options that facilitate debugging and troubleshooting. These options should be used to get as much information as possible. Consult your JVM vendor's documentation for their available options. The JVM options discussed in this section are Java HotSpot specific. See the Java HotSpot VM Options Web page for detailed information and usage instructions for all JVM options:

`http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html`

The following JVM options (standard and non standard) can help when debugging and troubleshooting applications:

- `-verbose:gc` or `-Xloggc:`*`file`* – These options are used to enable additional logs for each garbage collection event. In a distributed system, a GC pause on a single JVM can affect the performance of many JVMs, so it is essential to monitor garbage collection very closely. The `-Xloggc` option is similar to verbose GC but includes timestamps.
- `-Xprof` and `-Xrunhprof` – These options are used to view JVM profile data and are not intended for production systems.
- `-XX:-PrintGC`, `-XX:-PrintGCDetails`, and `-XX:-PrintGCTimeStamps` – These options are also used print messages at garbage collection.

- `-XX:-HeapDumpOnOutOfMemoryError` and `-XX:HeapDumpPath=./java_pid<pid>.hprof` – These options are used to initiate a heap dump when a `java.lang.OutOfMemoryError` is thrown.
- `-XX:ErrorFile=./hs_err_pid<pid>.log` – This option saves error data to a file.

Capturing Thread Dumps

Thread dumps are used to see detailed thread information, such as thread state, for each thread in the JVM. A thread dump also includes information on each deadlocked thread (if applicable). Thread dumps are useful because of Coherence's multi-threaded and distributed architecture. Thread dumps are often used to troubleshoot an application that is operating slowly or is deadlocked. Make sure to always collect several dumps over a period of time since a thread dump is only snapshot in time. Always include a set of thread dumps when submitting a support issue.

To perform a thread dump on Unix or Linux operating systems, press `Ctrl+\` at the application console. To perform a thread dump on Windows, press `Ctrl+Break` (or `Pause`). Both methods include a heap summary with the thread dump.

Most IDEs provide a thread dump feature that can be used to capture thread dumps while working in the IDE. In addition, Unix and Linux operating systems can use the kill -3 *pid* to cause a remote thread dump in the IDE. On Windows, use a third party tool (such as SendSignal) to send a `ctrl+break` signal to a remote Java process and cause a dump in the IDE.

Profiling tools, such as Oracle's VirtualVM (`jvisualvm`), JConsole (`jconsole`), and JRockit Mission Control (`jrmc`) are able to perform thread dumps. These tools are very useful because they provide a single tool for troubleshooting and debugging and display many different types of information in addition to just thread details.

Lastly, the `jstack` tool can be used to capture a thread dump for any process. For example, use `jps` to find a Java process ID and then execute the following from the command line:

```
jstack <pid>
```

The `jstack` tool is unsupported and may or may not be available in future versions of the JDK.

Capturing Heap Dumps

Heap dumps are used to see detailed information for all the objects in a JVM heap. The information includes how many instance of an object are loaded and how much memory is allocated to the objects. Heap information is typically used to find parts of an application that may potentially be wasting resources and causing poor performance. In a fully distributed Coherence environment, heap dumps can be tricky because application processing is occurring across the cluster and problematic objects may not necessarily be local to a JVM. Make sure to always collect several dumps over a period of time since a heap dump is only a snapshot in time. Always include heap dumps when submitting a support issue.

The easiest way to capture a heap dump is to use a profiling tool. Oracle's VirtualVM (`jvisualvm`), JConsole (`jconsole`), and JRockit Mission Control (`jrmc`) all provide heap dump features. In addition, most IDEs provide a heap dump feature that can be used to capture heap dumps while working in the IDE.

As an alternative, the `jmap` tool can be used to capture heap dumps, and the `jhat` tool can be used to view heap dumps. For example, use `jps` to find a Java process ID and then execute the following from the command line:

```
jmap -dump:format=b,file=/coherence.bin pid
```

To view the heap dump in a browser, execute the following from the command line and then browse to the returned address. The file can also be loaded into VisualVM for viewing.

```
jhat /coherence.bin
```

The `jmap` and `jhat` tools are unsupported and may or may not be available in future versions of the JDK.

Monitoring the Operating System

Always monitor a cluster member's operating system when troubleshooting and debugging Coherence-based applications. Poorly tuned operating systems can affect the overall performance of the cluster and may have adverse effects on an application. See *Oracle Coherence Administrator's Guide* for details on performance tuning.

In particular, the following areas are important to monitor:

- CPU – Is the processor running at 100% for extended periods of time?
- Memory/Swapping – Is the available RAM memory being exhausted and causing swap space to be used?
- Network – Is buffer size, the datagram size, and the Maximum Transmission Unit (MTU) size affecting performance and success rates?

To monitor the overall health of the operating system, use tools such as `vmstat` and `top` for Unix/Linux; for Windows, use `perfmon` and tools available from Windows Sysinternals (for example `procexp` and `procmon`). See *Oracle Coherence Administrator's Guide* for detailed instructions on how to test network performance.

Part II

Using Coherence Clusters

Part II contains the following chapters:

6

Introduction to Coherence Clusters

The following sections are included in this chapter:

Cluster Overview

A Coherence cluster is a collection of JVM processes. At run time, JVM processes that run Coherence automatically join and cluster. JVMs that join a cluster are called cluster members or cluster nodes. Cluster members communicate using Tangosol Cluster Management Protocol (TCMP). Cluster members use TCMP for both multicast communication (broadcast) and unicast communication (point-to-point communication).

A cluster contains services that are shared by all cluster members. The services include connectivity services (such as the Cluster service), cache services (such as the Distributed Cache service), and processing services (such as the invocation service). Each cluster member can provide and consume such services. The first cluster member is referred to as the senior member and typically starts the core services that are required to create the cluster. If the senior member of the cluster is shutdown, another cluster member assumes the senior member role.

Understanding TCMP

TCMP is an IP-based protocol that is used to discover cluster members, manage the cluster, provision services, and transmit data. TCMP can be configured to use:

- A combination of UDP/IP multicast and UDP/IP unicast. This is the default configuration.
- UDP/IP unicast only (that is, no multicast). See "Disabling Multicast Communication" on page 7-5. This configuration is used for network environments that do not support multicast or where multicast is not optimally configured.
- TCP/IP only (no UDP/IP multicast or UDP/IP unicast). See "Using the TCP Socket Provider" on page 10-15. This configuration is used for network environments that favor TCP.
- SSL over TCP/IP. See "Using the SSL Socket Provider" on page 10-15. This configuration is used for network environments that require highly secure communication between cluster members.

Use of Multicast

Multicast is used as follows:

- Cluster discovery: Multicast is used to discover if there is a cluster running that a new member can join.
- Cluster heartbeat: The most senior member in the cluster issues a periodic heartbeat through multicast; the rate can be configured and defaults to one per second.
- Message delivery: Messages that must be delivered to multiple cluster members are often sent through multicast, instead of unicasting the message one time to each member.

Use of Unicast

Unicast is used as follows:

- Direct member-to-member (point-to-point) communication, including messages, asynchronous acknowledgments (ACKs), asynchronous negative acknowledgments (NACKs) and peer-to-peer heartbeats. A majority of the communication on the cluster is point-to-point.
- Under some circumstances, a message may be sent through unicast even if the message is directed to multiple members. This is done to shape traffic flow and to reduce CPU load in very large clusters.
- All communication is sent using unicast if multicast communication is disabled.

Use of TCP

TCP is used as follows:

- A TCP/IP ring is used as an additional death detection mechanism to differentiate between actual node failure and an unresponsive node (for example, when a JVM conducts a full GC).
- TCMP can be configured to exclusively use TCP for data transfers. Like UDP, the transfers can be configured to use only unicast or both unicast and multicast.

Protocol Reliability

The TCMP protocol provides fully reliable, in-order delivery of all messages. Since the underlying UDP/IP protocol does not provide for either reliable or in-order delivery, TCMP uses a queued, fully asynchronous ACK- and NACK-based mechanism for reliable delivery of messages, with unique integral identity for guaranteed ordering of messages.

Protocol Resource Utilization

The TCMP protocol (as configured by default) requires only three UDP/IP sockets (one multicast, two unicast) and six threads per JVM, regardless of the cluster size. This is a key element in the scalability of Coherence; regardless of the number of servers, each node in the cluster still communicates either point-to-point or with collections of cluster members without requiring additional network connections.

The optional TCP/IP ring uses a few additional TCP/IP sockets, and an additional thread.

Protocol Tunability

The TCMP protocol is very tunable to take advantage of specific network topologies, or to add tolerance for low-bandwidth and high-latency segments in a geographically

distributed cluster. Coherence comes with a pre-set configuration. Some TCMP attributes are dynamically self-configuring at run time, but can also be overridden and locked down for deployment purposes.

Understanding Cluster Services

Coherence functionality is based on the concept of cluster services. Each cluster node can provide as well as consume any number of named services. These named services may be running on one or more other cluster nodes or a cluster node can register new named services. Each named service has a service name that uniquely identifies the service within the cluster and a service type that defines what the service can do. There may be multiple named instances of each service type (other than the root Cluster service). The following service types are supported by Coherence.

Connectivity Services

- **Cluster Service:** This service is automatically started when a cluster node must join the cluster; each cluster node always has exactly one service of this type running. This service is responsible for the detection of other cluster nodes, for detecting the failure of a cluster node, and for registering the availability of other services in the cluster.
- **Proxy Service:** This service allows connections (using TCP) from clients that run outside the cluster. While many applications are configured so that all clients are also cluster members, there are many use cases where it is desirable to have clients running outside the cluster. Remote clients are especially useful in cases where there are hundreds or thousands of client processes, where the clients are not running on the Java platform, or where a greater degree of coupling is desired.

Processing Services

- **Invocation Service:** This service provides clustered invocation and supports grid computing architectures. This services allows applications to invoke agents on any node in the cluster, or any group of nodes, or across the entire cluster. The agent invocations can be request/response, fire and forget, or an asynchronous user-definable model.

Data Services

- **Distributed Cache Service:** This service allows cluster nodes to distribute (partition) data across the cluster so that each piece of data in the cache is managed (held) by only one cluster node. The Distributed Cache Service supports pessimistic locking. Additionally, to support failover without any data loss, the service can be configured so that each piece of data is backed up by one or more other cluster nodes. Lastly, some cluster nodes can be configured to hold no data at all; this is useful, for example, to limit the Java heap size of an application server process, by setting the application server processes to not hold any distributed data, and by running additional cache server JVMs to provide the distributed cache storage. For more information on distributed caches, see "Distributed Cache" on page 11-1.
- **Replicated Cache Service:** This is a synchronized replicated cache service that fully replicates all of its data to all cluster nodes that run the service. Replicated caches support pessimistic locking to ensure that all cluster members receive the update when data is modified. Replicated caches are often used to manage internal application metadata. For more information on replicated caches, see "Replicated Cache" on page 11-5.

- **Optimistic Cache Service:** This is an optimistic-concurrency version of the Replicated Cache Service that fully replicates all of its data to all cluster nodes and employs an optimization similar to optimistic database locking to maintain coherency. All servers end up with the same current value even if multiple updates occur at the same exact time from different servers. The Optimistic Cache Service does not support pessimistic locking; so, in general, it should only be used for caching most recently known values for read-only uses. This service is rarely used. For more information on optimistic caches, see "Optimistic Cache" on page 11-7.

A clustered service typically uses one daemon thread and optionally has a thread pool that can be configured to provide the service with additional processing bandwidth. For example, the invocation service and the distributed cache service both fully support thread pooling to accelerate database load operations, parallel distributed queries, and agent invocations.

The above services are only the basic cluster services and not the full set of types of caches provided by Coherence. By combining clustered services with cache features, such as backing maps and overflow maps, Coherence provides an extremely flexible and configurable set of options for clustered applications.

Within a cache service, there exists any number of named caches. A named cache provides the standard JCache API, which is based on the Java collections API for key-value pairs, known as `java.util.Map`.

7

Setting Up a Cluster

This chapter provides instructions for completing common tasks that are associated with setting up a cluster.

The following sections are included in this chapter:

Overview of Setting Up Clusters

Coherence provides a default out-of-box cluster configuration that is used for demonstration purposes. It allows clusters to be quickly created and often requires little or no configuration changes. However, beyond demonstration, the default setup should not be used. Instead, unique clusters should be set up based on the network environment in which they run and based on the requirements of the applications that use them. A cluster that runs in single-server mode can be configured for unit testing and trivial development.

At a minimum, setting up a cluster includes defining the cluster's name and the cluster's multicast address. If multicast is undesirable or unavailable in an environment, then setting up the Well Known Addresses (WKA) feature is required. The rest of the tasks presented in this chapter are typically used when setting up a cluster and are completed when the default settings must be changed.

Clusters are set up within an operational override file (`tangosol-coherence-override.xml`). Each cluster member uses an override file to specify unique values that override the default configuration that is defined in the operational deployment descriptor. See "Specifying an Operational Configuration File" on page 3-2 for detailed information on using an operational override file. In addition, refer to Appendix A, "Operational Configuration Elements," for descriptions and usage information for all the operational elements that are discussed in this chapter.

Specifying a Cluster's Name

A cluster name is a user-defined name that uniquely identifies a cluster from other clusters that run on the network. Cluster members must specify the same cluster name to join and cluster. A cluster member does not start if the wrong name is specified when attempting to join an existing cluster. A unique cluster name is often used with a unique multicast port to create distinct clusters on the same network.

> **Note:** A cluster member uses a system generated cluster name if a name is not explicitly specified. Using the system generated name (and the out-of-box multicast defaults) increases the chance of having overlapping cluster configurations on the network. This can lead to cluster members accidentally joining an unexpected cluster.

To specify a cluster name, edit the operational override file and add a `<cluster-name>` element, within the `<member-identity>` element, that includes the cluster name. For example:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <member-identity>
         <cluster-name system-property="tangosol.coherence.cluster">MyCluster
         </cluster-name>
      </member-identity>
   </cluster-config>
</coherence>
```

The `tangosol.coherence.cluster` system property is used to specify the cluster name instead of using the operational override file. For example:

```
-Dtangosol.coherence.cluster=name
```

Specifying a Cluster Member's Identity

A set of identifiers are used to give a cluster member an identity within the cluster. The identity information is used to differentiate cluster members and conveys the members' role within the cluster. Some identifiers are also used by the cluster service when performing cluster tasks. Lastly, the identity information is valuable when displaying management information (for example, JMX) and facilitates interpreting log entries. The following list describes each of the identifiers:

- Site Name – the name of the geographic site that hosts the cluster member. The server's domain name is used if no name is specified. For WAN clustering, this value identifies the datacenter where 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).
- Rack Name – the name of the location within the geographic site that hosts the cluster member. 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).

- Machine Name – the name of the server that hosts the cluster member. The server's host name is used if no name is specified. The name is used as the basis for creating an ID. The cluster service uses the ID to ensure that data are backed up on different computers to prevent single points of failure.
- Process Name – the name of the JVM process that hosts the cluster member. The JVM process number is used if no name is specified. The process name makes it possible to easily differentiate among multiple JVMs running on the same computer.
- Member Name – the cluster member's unique name. The name makes it easy to differentiate cluster members especially when multiple members run on the same computer or within the same JVM. Always specify a member name (as a best practice) even though it is not required to do so.
- Role Name – the cluster member's role in the cluster. The role name allows an application to organize cluster members into specialized roles, such as cache servers and cache clients. Default role names (`CoherenceServer` for cache servers and *`application_class_name`* for cache clients) are used if no role name is specified.

To specify member identity information, edit the operational override file and add the member identity elements within the `<member-identity>` element as demonstrated below:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <member-identity>
         <site-name system-property="tangosol.coherence.site">pa-1</site-name>
         <rack-name system-property="tangosol.coherence.rack">100A</rack-name>
         <machine-name system-property="tangosol.coherence.machine">prod001
         </machine-name>
         <process-name system-property="tangosol.coherence.process">JVM1
         </process-name>
         <member-name system-property="tangosol.coherence.member">C1</member-name>
         <role-name system-property="tangosol.coherence.role">Server</role-name>
      </member-identity>
   </cluster-config>
</coherence>
```

The following system properties are used to specify a cluster member's identity information instead of using the operational override file.

```
-Dtangosol.coherence.site=pa-1 -Dtangosol.coherence.rack=100A
-Dtangosol.coherence.machine=prod001 -Dtangosol.coherence.process=JVM1
-Dtangosol.coherence.member=C1 -Dtangosol.coherence.role=Server
```

Configuring Multicast Communication

Cluster members use multicast communication to discover other cluster members and when a message must be communicated to multiple members of the cluster. The cluster protocol makes very judicious use of multicast and avoids things such as multicast storms. By default, data is only transmitted over multicast if it is intended for more than 25% of the cluster members. The vast majority of traffic is transmitted

using unicast even when multicast is enabled. For typical partitioned cache based clusters, most transmissions is point-to-point and only cluster membership and partition ownership is broadcast to the entire cluster.

Multicast communication is configured in an operational override file within the `<multicast-listener>` node. Many system properties are also available to configure multicast communication when starting a cluster member.

The following topics are included in this section:

- Specifying a Cluster's Multicast Address
- Disabling Multicast Communication
- Specifying the Multicast Time-to-Live
- Specifying the Multicast Join Timeout
- Changing the Multicast Threshold

Specifying a Cluster's Multicast Address

A multicast address (IP address and port) can be specified for a cluster member. Cluster members must use the same multicast address and port to join and cluster. Distinct clusters on the same network must use different multicast addresses.

A cluster member uses a default multicast address if an address is not explicitly specified. The default value depends on the release version and follows the convention of `{build}.{major version}.{minor version}.{patch}` for the address and `{major version}.{minor version}.{patch}` for the port.

> **Note:** Using the default multicast address and port (and the system generated cluster name) increases the chance of having overlapping cluster configurations on the network. This can lead to cluster members accidentally joining an unexpected cluster. Always use a unique port value to create a distinct cluster.

To specify a cluster multicast address, edit the operational override file and add both an `<address>` and `<port>` element and specify the address and port to be used by the cluster member. For example:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <multicast-listener>
         <address system-property="tangosol.coherence.clusteraddress">224.3.6.0
         </address>
         <port system-property="tangosol.coherence.clusterport">3059</port>
      </multicast-listener>
   </cluster-config>
</coherence>
```

The `tangosol.coherence.clusteraddress` and `tangosol.coherence.clusterport` system properties are used to specify the cluster multicast address instead of using the operational override file. For example:

```
-Dtangosol.coherence.clusteraddress=224.3.6.0
-Dtangosol.coherence.clusterport=3059
```

Changing the Multicast Socket Interface

The multicast socket is bound to the same network interface (NIC) as the unicast listener IP address. A different NIC for multicast can be configured but, with rare exception, it is strongly discouraged as it can lead to partial failure of the cluster.

With two NICs, the interface (and thus network) used for multicast traffic is different from the interface (and thus network) used for unicast (UDP/IP) and TCP-ring (TCP/IP) traffic. Communication on one interface (or network) continues to succeed even if the other interface has failed; this scenario prolongs failure detection and failover. Since the clustering protocol handles member (and thus interface) failure, it is preferable to have all communication fail so that a failed member is quickly detected and removed from the cluster.

To change the default multicast network interface, edit the operational override file and add an `<interface>` element that specifies the IP address to which the multicast socket binds. For example:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <multicast-listener>
         <interface>192.168.0.1</interface>
      </multicast-listener>
   </cluster-config>
</coherence>
```

Disabling Multicast Communication

Multicast traffic may be undesirable or may be disallowed in some network environments. In this case, use the Well Known Addresses feature to prevent Coherence from using multicast. This disables multicast discovery and also disable multicast for all data transfers; unicast (point-to-point) is used instead. Coherence is designed to use point-to-point communication as much as possible, so most application profiles do not see a substantial performance impact. See "Using Well Known Addresses" on page 7-9.

> **Note:** Disabling multicast does put a higher strain on the network. However, this only becomes an issue for large clusters with greater than 100 members.

Specifying the Multicast Time-to-Live

The time-to-live value (TTL) setting designates how far multicast UDP/IP packets can travel on a network. The TTL is expressed in terms of how many hops a packet survives; each network interface, router, and managed switch is considered one hop.

The TTL value should be set to the lowest integer value that works. Setting the value too high can use unnecessary bandwidth on other LAN segments and can even cause

the operating system or network devices to disable multicast traffic. Typically, setting the TTL value to `1` works on a simple switched backbone. A value of `2` or more may be required on an advanced backbone with intelligent switching. A value of `0` is used for single server clusters that are used for development and testing. See "Enabling Single-Server Mode" on page 7-12 for more information on single server clusters.

To specify the TTL, edit the operational override file and add a `<time-to-live>` element that includes the TTL value. For example:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <multicast-listener>
         <time-to-live system-property="tangosol.coherence.ttl">3</time-to-live>
      </multicast-listener>
   </cluster-config>
</coherence>
```

The `tangosol.coherence.ttl` system property is used to specify the TTL value instead of using the operational override file. For example:

```
-Dtangosol.coherence.ttl=3
```

Specifying the Multicast Join Timeout

The multicast join timeout defines how much time a cluster member waits to join a cluster. If the timeout is reached and an existing cluster is not detected, then the cluster member starts its own cluster and elects itself as the senior cluster member. A short timeout can be specified during development and testing. A timeout of 30 seconds is generally adequate for production environments.

> **Note:** The first member of the cluster waits the full duration of the join timeout before it assumes the role of the senior member. If the cluster startup timeout is less than the join timeout, then the first member of the cluster fails during cluster startup. The cluster member timeout is specified using the packet publisher timeout (`<timeout-milliseconds>`). See "packet-delivery" on page A-46.

To specify the join timeout, edit the operational override file and add a `<join-timeout-milliseconds>` element that includes the timeout value. For example:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <multicast-listener>
         <join-timeout-milliseconds>6000</join-timeout-milliseconds>
      </multicast-listener>
   </cluster-config>
```

```
</coherence>
```

Changing the Multicast Threshold

Cluster members use both multicast and unicast communication when sending cluster packets. The multicast threshold value is used to determine whether to use multicast for packet delivery or unicast. Setting the threshold higher or lower can force a cluster to favor one style of communication over the other. The threshold setting is not used if multicast communication is disabled.

The multicast threshold is a percentage value and is in the range of 1% to 100%. In a cluster of *n* members, a cluster member that is sending a packet to a set of destination nodes (not counting itself) of size *d* (in the range of `0` to *n*`-1`) sends a packet using multicast only if the following hold true:

- The packet is being sent over the network to multiple nodes (*d* `> 1`).
- The number of nodes is greater than the specified threshold (*d* `> (`*n*`-1) * (`*threshold*`/100)`).

  For example, in a 25 member cluster with a multicast threshold of 25%, a cluster member only uses multicast if the packet is destined for 6 or more members (`24 * .25 = 6`).

Setting this value to `1` allows the cluster to use multicast for basically all multi-point traffic. Setting this value to `100` forces the cluster to use unicast for all multi-point traffic except for explicit broadcast traffic (for example, cluster heartbeat and discovery) because the 100% threshold is never exceeded. With the setting of `25` (the default) a cluster member sends the packet using unicast if it is destined for less than one-fourth of all nodes, and sends the packet using multicast if it is destined for one-fourth or more of all nodes.

To specify the multicast threshold, edit the operational override file and add a `<multicast-threshold-percent>` element that includes the threshold value. For example:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <multicast-listener>
         <multicast-threshold-percent>40</multicast-threshold-percent>
      </multicast-listener>
   </cluster-config>
</coherence>
```

Specifying a Cluster Member's Unicast Address

Cluster members use unicast for direct member-to-member (point-to-point) communication, which makes up the majority of communication on the cluster. A default unicast address (IP address and ports) is used but can be specified as required within the `<unicast-listener>` element.

The unicast listener, as configured out-of-box, selects the unicast address as follows:

- address – A cluster member attempts to obtain the IP to bind to using the `java.net.InetAddress.getLocalHost()` call. Explicitly specify the address on computers with multiple IPs or NICs, if required. Moreover, the localhost setting may not work on systems that define localhost as the loopback address; in that case, the computer name or the specific IP address must be specified. The multicast socket binds to the same interface as defined by this address. See "Changing the Multicast Socket Interface" on page 7-5.
- ports – A cluster member uses two unicast UDP ports. The default behavior is to attempt to use port 8088 for the first port (*`port1`*). If port 8088 is not available, automatic port adjustment is used to select the next available port. The second port (*`port2`*) is automatically opened and defaults to the next available port after *`port1`* (*`port1`* `+ 1` if available). Automatic port adjustment can be disabled. In this case, *`port1`* must be available and the second port is always *`port1`* `+ 1`.

  Two UDP ports are used because:

  - It reduces contention between inbound and outbound traffic and avoids doing both heavy transmits and receives on the same port
  - It allows for coherence members to communicate at the optimal packet size based on the Maximum Transmission Unit (MTU) of the operating system. One UDP port is used for large packets, and the other port is for packets with sizes at or under the network MTU. The separation allows for separate packet buffers based on size.
  - It allows for large clusters (> 500 members) to be run without artificially increasing the size of the socket buffers

To specify a cluster member's unicast address, edit the operational override file and add both an `<address>` and `<port>` element (and optionally a `<port-auto-adjust>` element) and specify the address and port to be used by the cluster member. For example:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <unicast-listener>
         <address system-property="tangosol.coherence.localhost">192.168.0.1
         </address>
         <port system-property="tangosol.coherence.localport">8090</port>
         <port-auto-adjust system-property="tangosol.coherence.localport.adjust">
            true
         </port-auto-adjust>
      </unicast-listener>
   </cluster-config>
</coherence>
```

The `tangosol.coherence.localhost`, `tangosol.coherence.localport`, and `tangosol.coherence.localport.adjust` system properties are used to specify the unicast address instead of using the operational override file. For example:

```
-Dtangosol.coherence.localhost=192.168.0.1 -Dtangosol.coherence.localport=8090
-Dtangosol.coherence.localport.adjust=true
```

Using Well Known Addresses

The Well Known Addresses (WKA) feature is a mechanism that allows cluster members to discover and join a cluster using unicast instead of multicast. WKA is most often used when multicast networking is undesirable or unavailable in an environment or when an environment is not properly configured to support multicast. All cluster multicast communication is disabled if WKA is enabled.

WKA is enabled by specifying a small subset of cluster members (referred to as WKA members) that are able to start a cluster. The optimal number of WKA members varies based on the cluster size. Generally, WKA members should be about 10% of the cluster. One or two WKA members for each switch is recommended.

WKA members are expected to remain available over the lifetime of the cluster but are not required to be simultaneously active at any point in time. Only one WKA member must be operational for cluster members to discover and join the cluster. In addition, after a cluster member has joined the cluster, it receives the addresses of all cluster members and then broadcasts are performed by individually sending messages to each cluster member. This allows a cluster to operate even if all WKA members are stopped. However, new cluster members are not able to join the cluster unless they themselves are a WKA member or until a WKA member is started. In this case, the senior-most member of the cluster polls the WKA member list and allows the WKA member to rejoin the existing cluster.

There are two ways to specify WKA members. The first method explicitly defines a list of addresses. The second method uses an address provider implementation to get a list of WKA addresses. Both methods are configured in an operational override file within the `<well-known-addresses>` subelement of the `<unicast-listener>` element.

The following topics are included in this section:

Specifying WKA Member Addresses

WKA members are explicitly specified within the `<socket-address>` element. Any number of `<socket-address>` elements can be specified and each must define both the address and port of a WKA member by using the `<address>` and `<port>` elements. If a cluster member specifies its own address, then the cluster member is a WKA member when it is started. The list of WKA members must be the same on every cluster member to ensure that different cluster members do not operate independently from the rest of the cluster. The following example specifies two WKA members:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <unicast-listener>
         <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>
      </unicast-listener>
   </cluster-config>
</coherence>
```

> **Note:** When setting up a WKA member, the port value must match the port value that is specified for the member's unicast listener port. See "Specifying a Cluster Member's Unicast Address" on page 7-7 for more information on setting the unicast port.

Using WKA System Properties

A single WKA member can be specified using the `tangosol.coherence.wka` and `tangosol.coherence.wka.port` system properties instead of specifying the address in an operational override file. The system properties are intended for demonstration and testing scenarios to quickly specify a single WKA member. For example:

```
-Dtangosol.coherence.wka=192.168.0.100 -Dtangosol.coherence.wka.port=8088
```

To create additional system properties to specify multiple WKA member addresses, an operational override file must be used to define multiple WKA member addresses and a `system-property` attribute must be defined for each WKA member address element. The attributes must include the system property names to be used to override the elements. The below example defines two addresses including system properties:

> **Note:** Defining additional system properties to specify a list of WKA members can be used during testing or in controlled production environments. However, the best practice is to exclusively use an operational override file to specify WKA members in production environments. This ensure the same list of WKA members exists on each cluster member.

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <unicast-listener>
         <well-known-addresses>
            <socket-address id="1">
               <address system-property="tangosol.coherence.wka"></address>
               <port system-property="tangosol.coherence.wka.port"></port>
            </socket-address>
            <socket-address id="2">
               <address system-property="tangosol.coherence.wka2"></address>
               <port system-property="tangosol.coherence.wka2.port"></port>
            </socket-address>
         </well-known-addresses>
      </unicast-listener>
   </cluster-config>
</coherence>
```

For the above example, the WKA member addresses are specified using the system properties as follows:

```
-Dtangosol.coherence.wka=192.168.0.102 -Dtangosol.coherence.wka.port=8090
-Dtangosol.coherence.wka2=192.168.0.103 -Dtangosol.coherence.wka2.port=8094
```

See "Creating Custom System Properties" on page 3-20 for more information on defining system properties.

Specifying a WKA Address Provider

A WKA address provider offers a programmatic way to define WKA members. A WKA address provider must implement the `com.tangosol.net.AddressProvider` interface. Implementations may be as simple as a static list or as complex as using dynamic discovery protocols. The address provider must return a terminating `null` address to indicate that all available addresses have been returned. The address provider implementation is called when the cluster member starts.

> **Note:** implementations must exercise extreme caution since any delay with returned or unhandled exceptions causes a discovery delay and may cause a complete shutdown of the cluster service on the member. Implementations that involve more expensive operations (for example, network fetch) may choose to do so asynchronously by extending the `com.tangosol.net.RefreshableAddressProvider` class.

To use a WKA address provider implementation, add an `<address-provider>` element and specify the fully qualified name of the implementation class within the `<class-name>` element. For example:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <unicast-listener>
         <well-known-addresses>
            <address-provider>
               <class-name>package.MyAddressProvider</class-name>
            </address-provider>
         </well-known-addresses>
      </unicast-listener>
   </cluster-config>
</coherence>
```

As an alternative, the `<address-provider>` element supports the use of a `<class-factory-name>` element that is used to specify a factory class for creating `AddressProvider` instances, and a `<method-name>` element to specify the static factory method on the factory class that performs object instantiation. The following example gets an address provider instance using the `getAddressProvider` method on the `MyAddressProviderFactory` class.

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

```
<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <unicast-listener>
         <well-known-addresses>
            <address-provider>
               <class-factory-name>package.MyAddressProviderFactory
               </class-factory-name>
               <method-name>getAddressProvider</method-name>
            </address-provider>
         </well-known-addresses>
      </unicast-listener>
   </cluster-config>
</coherence>
```

Any initialization parameters that are required for a class or class factory implementation can be specified using the `<init-params>` element. Initialization parameters are accessible by implementations that support the `com.tangosol.run.xml.XmlConfigurable` interface or implementations that include a public constructor with a matching signature. The following example sets the `iMaxTime` parameter to `2000`.

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <unicast-listener>
         <well-known-addresses>
            <address-provider>
               <class-name>package.MyAddressProvider</class-name>
               <init-params>
                  <init-param>
                     <param-name>iMaxTime</param-name>
                     <param-value>2000</param-value>
                  </init-param>
               </init-params>
            </address-provider>
         </well-known-addresses>
      </unicast-listener>
   </cluster-config>
</coherence>
```

Enabling Single-Server Mode

Single-Server mode is a cluster that is constrained to run on a single computer and does not access the network. Single-Server mode offers a quick way to start and stop a cluster and is used for development and unit testing.

To enable single-server mode, edit the operational override file and add a `<time-to-live>` element that is set to `0` and a unicast `<address>` element that is set to an address that is routed to loopback. On most computers, setting the address to `127.0.0.1` works. For example:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <unicast-listener>
         <address system-property="tangosol.coherence.localhost">127.0.0.1
         </address>
      </unicast-listener>
      <multicast-listener>
         <time-to-live system-property="tangosol.coherence.ttl">0</time-to-live>
      </multicast-listener>
   </cluster-config>
</coherence>
```

The `tangosol.coherence.ttl` and `tangosol.coherence.localhost` system properties are used to enable single-server mode instead of using the operational override file. For example:

```
-Dtangosol.coherence.ttl=0 -Dtangosol.coherence.localhost=127.0.0.1
```

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 computer. In such cases, a unique cluster name, such as an email address, must also be configured. A cluster member cannot join an existing cluster if it uses a different cluster name. See "Specifying a Cluster's Name" on page 7-2 for details on configuring a cluster name.

Configuring Death Detection

Death detection is a cluster mechanism that quickly detects when a cluster member has failed. Failed cluster members are removed from the cluster and all other cluster members are notified about the departed member. Death detection allows the cluster to differentiate between actual member failure and an unresponsive member, such as the case when a JVM conducts a full garbage collection.

Death detection works by creating a ring of TCP connections between all cluster members. TCP communication is sent on the same port that is used for cluster UDP communication. Each cluster member issues a unicast heartbeat, and the most senior cluster member issues the cluster heartbeat, which is a broadcast message. Each cluster member uses the TCP connection to detect both process death (TcpRing component) and hardware death (IpMonitor component) of another node within the heartbeat interval. Death detection is enabled by default and is configured within the `<tcp-ring-listener>` element.

The following topics are included in this section:

- Changing TCP-Ring Settings
- Changing the Heartbeat Interval
- Disabling Death Detection

Changing TCP-Ring Settings

Several settings are used to change the default behavior of the TCP-ring listener. This includes changing the amount of attempts and time before determining that a computer that is hosting cluster members has become unreachable. These default to 3

and 15 seconds, respectively. The TCP/IP server socket backlog queue can also be set and defaults to the value used by the operating system.

To change the TCP-ring settings, edit the operational override file and add the following TCP-Ring elements:

> **Note:** The values of the `<ip-timeout>` and `<ip-attempts>` elements should be high enough to insulate against allowable temporary network outages.

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <tcp-ring-listener>
         <ip-timeout system-property="tangosol.coherence.ipmonitor.pingtimeout">
         25s
         </ip-timeout>
         <ip-attempts>5</ip-attempts>
         <listen-backlog>10</listen-backlog>
      </tcp-ring-listener>
   </cluster-config>
</coherence>
```

The `tangosol.coherence.ipmonitor.pingtimeout` system property is used to specify a timeout instead of using the operational override file. For example:

```
-Dtangosol.coherence.ipmonitor.pingtimeout=20s
```

Changing the Heartbeat Interval

The death detection heartbeat interval can be changed. A higher interval may alleviate minimal network traffic but may also prolongs detection of failed members. The default heartbeat value is 1 second.

> **Note:** The heartbeat setting technically controls how often to evaluate whether or not a heartbeat needs to be emitted. The actual heartbeat interval may or may not be emited within the specified interval depending on the evaluation process.

To change the death detection heartbeat interval, edit the operational override file and add a `<heartbeat-milliseconds>` element that includes the heartbeat value. For example:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <packet-publisher>
         <packet-delivery>
```

```
            <heartbeat-milliseconds>5000</heartbeat-milliseconds>
         </packet-delivery>
      </packet-publisher>
   </cluster-config>
</coherence>
```

Disabling Death Detection

Death detection is enabled by default and must be explicitly disabled. Disabling death detection alleviates only minimal network traffic and prolongs the detection of failed members. If disabled, a cluster member uses the packet publisher's resend timeout interval to determine that another member has stopped responding to UDP packets. By default, the timeout interval is set to 5 minutes. See "Changing the Packet Resend Timeout" on page 10-6 for more details.

> **Note:** Using the packet publisher's resend timeout to detect a failed cluster member is generally not recommended as it is error proned and can produce false positives due to high garbage collection intervals.

To disable death detection, edit the operational override file and add an `<enabled>` element that is set to `false`. For example:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <tcp-ring-listener>
         <enabled>false</enabled>
      </tcp-ring-listener>
   </cluster-config>
</coherence>
```

Specifying Cluster Priorities

The cluster priority mechanism allows a priority value to be assigned to a cluster member and to different threads running within a member.

The following topics are included in this section:

- Specifying a Cluster Member's Priority
- Specifying Thread Priority

Specifying a Cluster Member's Priority

A cluster member's priority is used as the basis for determining tie-breakers between members. If a condition occurs in which one of two members is 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 is ejected.

To specify a cluster member's priority, edit the operational override file and add a `<priority>` element, within the `<member-identity>` node, that includes a priority value between 1 and 10 where 1 is the highest priority. For example:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <member-identity>
         <priority system-property="tangosol.coherence.priority">1</priority>
      </member-identity>
   </cluster-config>
</coherence>
```

The `tangosol.coherence.priority` system property can also be used to specify a cluster member's priority instead of using the operational override file. For example:

```
-Dtangosol.coherence.priority=1
```

Specifying Thread Priority

Multiple cluster components support thread priority. The priority is used as the basis for determining Java thread execution importance. The components include: the multicast listener, the unicast listener, the TCP ring listener, the packet speaker, the packet publisher, and the incoming message handler. The default priority setup gives the packet publisher the highest priority followed by the incoming message handler followed by the remaining components.

Thread priority is specified within each component's configuration element (`<unicast-listener>`, `<multicast-listener>`, `<packet-speaker>`, `<packet-publisher>`, `<tcp-ring-listener>`, and `<incoming-message-handler>` elements, respectively). For example, to specify a thread priority for the unicast listener, edit the operational override file and add a `<priority>` element, within the `<unicast-listener>` node, that includes a priority value between 1 and 10 where 1 is the highest priority:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <unicast-listener>
         <priority>5</priority>
      </unicast-listener>
   </cluster-config>
</coherence>
```

8

Starting and Stopping Cluster Members

This chapter provides basic instructions for starting and stopping cache servers and cache clients. If you are having difficulties establishing a cluster when using multicast, see *Oracle Coherence Administrator's Guide* for instructions on performing a multicast connectivity test.

The following sections are included in this chapter:

Starting Cache Servers

Cache servers are cluster members that are responsible for storing cached data. A cluster may be comprised of many cache servers. Each cache server runs in its own JVM.

The following topics are included in this section:

Starting Cache Servers From the Command Line

The `com.tangosol.net.DefaultCacheServer` class is used to start a cache server instance from the command line. Use the Java `-cp` option to indicate the location of the `coherence.jar` file and the location where the `tangosol-coherence-override.xml` and `coherence-cache-config.xml` files are located. The location of the configuration files must precede the `cohernce.jar` file on the classpath; otherwise, the default configuration files that are located in the `coherence.jar` file are used to start the cache server instance. See Chapter 3, "Understanding Configuration," for detailed information on configuration files.

The following example starts a cache server member and uses any configuration files that are placed in *COHERENCE_HOME*`\config`:

```
java -server -Xms512m -Xmx512m -cp COHERENCE_HOME\config;COHERENCE_
HOME\lib\coherence.jar com.tangosol.net.DefaultCacheServer
```

The *COHERENCE_HOME*`\bin\cache-server` script is provided as a convenience and can startup a cache server instance. The script sets up a basic environment and then runs the `DefaultCacheServer` class. There is a script for both the Windows and

UNIX-based platforms. The scripts are typically modified as required for a particular cluster.

> **Tip:** During testing, it is sometimes useful to create multiple scripts with different names that uniquely identify each cache server. For example: `cahe-server-a`, `cache-server-b`, and so on.

Starting Cache Servers Programmatically

An application can use or extend the `DefaultCacheServer` class as required when starting a cache server. For example, an application may want to do some application-specific setup or processing before starting a cache server and its services.

For basic use cases, the `main` method can be called and requires two arguments: the name of a cache configuration file that is found on the classpath, and the number of seconds between checks for stopped services. Stopped services are started if they are set to be automatically started (as configured by an `<autostart>` element in the cache configuration file). The following example starts a cache server using the `main` method:

```
String[] args = new String[]{"my-cache-config.xml", "5"};
DefaultCacheServer.main(args);
```

The `DefaultCacheServer(DefaultConfigurableCacheFactory)` constructor uses a factory class to create a cache server instance that uses a specified cache configuration file. The following example creates a `DefaultCacheServer` instance and uses the `startAndMonitor(long)` method to start a cache server as in the previous example:

```
DefaultConfigurableCacheFactory factory;
factory = new DefaultConfigurableCacheFactory("my-cache-config.xml");

DefaultCacheServer dcs = new DefaultCacheServer(factory);
dcs.startAndMonitor(5000);
```

Two static start methods (`start()` and `start(ConfigurableCacheFactory)`) are also available to start a cache server and return control. However, the `CacheFactory` class is typically used instead of these methods which remain for backward compatibility.

Applications that require even more fine-grained control can subclass the `DefaultCacheServer` class and override its methods to perform any custom processing as required. See *Oracle Coherence Java API Reference* for detailed information on the `DefaultCacheServer` class.

Starting Cache Clients

Cache clients are cluster members that join the cluster to interact with the cluster's services. Cache clients can be as simple as an application that gets and puts data in a cache or can be as complex as a data grid compute application that processes data that is in a cache. The main difference between a cache client and a cache server is that cache clients are generally not responsible for cluster storage.

The following topics are included in this section:

- Disabling Local Storage
- Using the CacheFactory Class to Start a Cache Client

Disabling Local Storage

Cache clients that use the partition cache service (distributed caches) should not maintain any partitioned data. Cache clients that have storage disabled perform better and use less resources. Partitioned data should only be distributed among cache server instances.

Local storage is disabled on a per-process basis using the `tangosol.coherence.distributed.localstorage` system property. This allows cache clients and servers to use the same configuration descriptors. For example:

```
java -cp COHERENCE_HOME\config;COHERENCE_HOME\lib\coherence.jar
-Dtangosol.coherence.distributed.localstorage=false com.MyApp
```

Using the CacheFactory Class to Start a Cache Client

Any application that uses the `com.tangosol.net.CacheFactory` class to get an instance of a cache becomes a cluster member and is considered a cache client. The following example demonstrates the most common way of starting a cache client:

```
CacheFactory.ensureCluster();
NamedCache cache = CacheFactory.getCache("cache_name");
```

When starting an application that is a cache client, use the Java `-cp` option to indicate the location of the `coherence.jar` file and the location where the `tangosol-coherence-override.xml` and `coherence-cache-config.xml` files are located. The location of the configuration files must precede the `cohernce.jar` file on the classpath; otherwise, the default configuration files that are located in the `coherence.jar` file are used to start the cache server instance. See Chapter 3, "Understanding Configuration," for detailed information on configuration files. For example:

```
java -cp COHERENCE_HOME\config;COHERENCE_HOME\lib\coherence.jar
-Dtangosol.coherence.distributed.localstorage=false com.MyApp
```

Stopping Cluster Members

The following topics are included in this section:

- Stopping Cluster Members From the Command Line
- Stopping Cache Servers Programmatically

Stopping Cluster Members From the Command Line

Cluster members are most often shutdown using the `kill` command when on the UNIX platform and `Ctrl+c` when on the Windows platform. These commands initiate the standard JVM shutdown hook which is invoked upon normal JVM termination.

Note: Issuing the `kill -9` command triggers an abnormal JVM termination and the shutdown hook does not run. However, a graceful shutdown is generally not required if a service is known to be node-safe (as seen using JMX management) before termination.

The action a cluster member takes when receiving a shutdown command is configured in the operational override file within the `<shutdown-listener>` element. The following options are available:

- `none` — perform no explicit shutdown actions. This is the suggested value for production unless testing has verified that the behavior on external shutdown is exactly what is desired.
- `force` — (default) perform a hard-stop on the node by calling `Cluster.stop()`. This is the default out-of-box action.
- `graceful` — perform a normal shutdown by calling `Cluster.shutdown()`
- `true` — same as force
- `false` — same as none

The following example sets the shutdown hook to `none`.

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <shutdown-listener>
         <enabled system-property="tangosol.coherence.shutdownhook">none</enabled>
      </shutdown-listener>
   </cluster-config>
</coherence>
```

The `tangosol.coherence.shutdownhook` system property is used to specify the shutdown hook behavior instead of using the operational override file. For example:

```
-Dtangosol.coherence.shutdownhook=none
```

Stopping Cache Servers Programmatically

The `DefaultCacheServer` class provides two methods that are used to shutdown a cache server:

> **Note:** Shutdown is supposed to be called in a standalone application where it shuts down the instance which the `DefaultCacheServer` class itself maintains as a static member.

- `shutdown()` – This is a static method that is used to shut down a cache server that was started on a different thread using the `DefaultCacheServer.main()` or `DefaultCacheServer.start()` methods.
- `shutdownServer()` – This method is called on a `DefaultCacheServer` instance which an application keeps hold of.

9

Dynamically Managing Cluster Membership

Coherence manages cluster membership by automatically adding new servers to the cluster when they start and automatically detecting their departure when they are shut down or fail. Applications have full access to this information and can sign up to receive event notifications when members join and leave the cluster. Coherence also tracks all the services that each member is providing and consuming. This information is used to, among other things, plan for service resiliency in case of server failure and to load-balance data management across all members of the cluster.

The following sections are included in this chapter:

Using the Cluster and Service Objects

From any cache, the application can obtain a reference to the local representation of a cache's service. From any service, the application can obtain a reference to the local representation of the cluster.

```
CacheService service = cache.getCacheService();
Cluster      cluster = service.getCluster();
```

From the `Cluster` object, the application can determine the set of services that run in the cluster. This is illustrated in Example 9–1.

Example 9–1 Determining Services Running in the Cluster

```
...
for (Enumeration enum = cluster.getServiceNames(); enum.hasMoreElements(); )
    {
    String sName = (String) enum.nextElement();
    ServiceInfo info = cluster.getServiceInfo(sName);
    // ...
    }
...
```

The `ServiceInfo` object provides information about the service, including its name, type, version and membership.

For more information on this feature, see the API documentation for `NamedCache`, `CacheService`, `Service`, `ServiceInfo` and `Cluster`.

Using the Member Object

The primary information that an application can determine about each member in the cluster is:

- The Member's IP address
- What date/time the Member joined the cluster

As an example, if there are four servers in the cluster with each server running one copy ("instance") of the application and all four instances of the application are clustered, then the cluster is composed of four Members. From the `Cluster` object, the application can determine what the local `Member` is:

```
Member memberThis = cluster.getLocalMember();
```

From the `Cluster` object, the application can also determine the entire set of cluster members:

```
Set setMembers = cluster.getMemberSet();
```

From the `ServiceInfo` object, the application can determine the set of cluster members that are participating in that service:

```
ServiceInfo info = cluster.getServiceInfo(sName);
Set setMembers = info.getMemberSet();
```

For more information on this feature, see the API documentation for `Member`.

Listening to Member Events

Applications must create a class that implements the `MemberListener` interface (see Example 9–2) to listen for cluster and service membership changes. The listener class is then added on a service by either using the service's `addMemberListener` method or by adding a `<member-listener>` element to a cache scheme definition.

There are two advantages to using the configuration approach versus the programmatic approach. First, programmatically, listeners can only be added to a service that is running. As such, the first `MEMBER_JOINED` event is missed. Secondly, the `addMemberListener` call must be issued on each and every cluster node that runs the corresponding service. The configuration approach solves both of these issues.

The following example adds a listener implementation named `MyMemberListener` to a service using the `addMemberListener` method:

```
Service service = cache.getCacheService();
service.addMemberListener(package.MyMemberListener);
```

The service can also be looked up by its name:

```
Service service = cluster.getService(sName);
service.addMemberListener(package.MyMemberListener);
```

The following example adds a listener implementation named `MyMemberListener` to a service named `DistributedCache` by adding the `<member-listener>` element to a distributed cache scheme definition:

```
<distributed-scheme>
   <scheme-name>example-distributed</scheme-name>
   <service-name>DistributedCache</service-name>
   <member-listener>
```

```
      <class-name>package.MyMemberListener</class-name>
   </member-listener>
   <backing-map-scheme>
      <local-scheme>
         <scheme-ref>example-binary-backing-map</scheme-ref>
      </local-scheme>
   </backing-map-scheme>
   <autostart>true</autostart>
</distributed-scheme>
```

The `<member-listener>` element can be used within the `<distributed-scheme>`, `<replicated-scheme>`, `<optimistic-scheme>`, `<invocation-scheme>`, and `<proxy-scheme>` elements. See Appendix B, "Cache Configuration Elements" for a reference of valid cache configuration elements.

> **Note:** A `MemberListener` implementation must have a public default constructor when using the `<member-listener>` element to add a listener to a service.

Example 9–2 demonstrates a `MemberListener` implementation that prints out all the membership events that it receives:

Example 9–2 A Sample MemberListener Implementation

```
public class MemberEventPrinter
        extends Base
        implements MemberListener
    {
    public void memberJoined(MemberEvent evt)
        {
        out(evt);
        }

    public void memberLeaving(MemberEvent evt)
        {
        out(evt);
        }

    public void memberLeft(MemberEvent evt)
        {
        out(evt);
        }
    }
```

The `MemberEvent` object carries information about the event type (either `MEMBER_JOINED`, `MEMBER_LEAVING`, or `MEMBER_LEFT`), the member that generated the event, and the service that acts as the source of the event. Additionally, the event provides a method, `isLocal()`, that indicates to the application that it is *this* member that is joining or leaving the cluster. This is useful for recognizing soft restarts in which an application automatically rejoins a cluster after a failure occurs.

> **Note:** Calling the `CacheFactory.shutdown()` method unregisters all listeners. In this case, both the `MEMBER_LEAVING` and `MEMBER_LEFT` events are sent. If a member terminates for any other reason, only the `MEMBER_LEFT` event is sent.

Example 9–3 illustrates how information encapsulated in a `MemberEvent` object can be used.

Example 9–3 Using Event Type Information in a MemberEvent Object

```
public class RejoinEventPrinter
        extends Base
        implements MemberListener
    {
    public void memberJoined(MemberEvent evt)
        {
        if (evt.isLocal())
            {
            out("this member just rejoined the cluster: " + evt);
            }
        }

    public void memberLeaving(MemberEvent evt)
        {
        }

    public void memberLeft(MemberEvent evt)
        {
        }
    }
```

For more information on these feature, see the API documentation for `Service`, `MemberListener` and `MemberEvent`.

10

Tuning TCMP Behavior

This chapter provides instructions for changing default TCMP settings. A brief overview of TCMP is also provided. See "Understanding TCMP" on page 6-1 for additional details on TCMP. Also, see *Oracle Coherence Administrator's Guide* the which includes many tuning recommendations and instructions.

The following sections are included in this chapter:

Overview of TCMP Data Transmission

Cluster members communicate using Tangosol Cluster Management Protocol (TCMP). TCMP is an IP-based protocol that is used to discover cluster members, manage the cluster, provision services, and transmit data. TCMP is an asynchronous protocol; communication is never blocking even when many threads on a server are communicating at the same time. Asynchronous communication also means that the latency of the network (for example, on a routed network between two different sites) does not affect cluster throughput, although it affects the speed of certain operations.

The TCMP protocol is very tunable to take advantage of specific network topologies, or to add tolerance for low-bandwidth and high-latency segments in a geographically distributed cluster. Coherence comes with a pre-set configuration. Some TCMP attributes are dynamically self-configuring at run time, but can also be overridden and locked down for deployment purposes. TCMP behavior should always be changed based on performance testing. Coherence includes a datagram test that is used to evaluate TCMP data transmission performance over the network. See *Oracle Coherence Administrator's Guide* for instructions on using the datagram test utility to test network performance.

TCMP data transmission behavior is configured within the `tangosol-coherence-override.xml` file using the `<packet-publisher>`, `<packet-speaker>`, `<incoming-message-handler>`, and `<outgoing-message-handler>` elements. See Appendix A, "Operational Configuration Elements," for a reference of all TCMP-related elements that are discussed in this chapter.

Throttling Data Transmission

The speed at which data is transmitted is controlled using the `<flow-control>` and `<traffic-jam>` elements. These elements can help achieve the greatest throughput with the least amount of packet failure. The throttling settings discussed in this section are typically changed when dealing with slow networks, or small packet buffers.

The following topics are included in this section:

- Adjusting Packet Flow Control Behavior
- Disabling Packet Flow Control
- Adjusting Packet Traffic Jam Behavior

Adjusting Packet Flow Control Behavior

Flow control is used to dynamically throttle the rate of packet transmission to a given cluster member based on point-to-point transmission statistics which measure the cluster member's responsiveness. Flow control stops a cluster member from being flooded with packets while it is incapable of responding.

Flow control is configured within the `<flow-control>` element. There are two settings that are used to adjust flow control behavior:

- `<pause-detection>` – This setting controls the maximum number of packets that are resent to an unresponsive cluster member before determining that the member is paused. When a cluster member is marked as paused, packets addressed to it are sent at a lower rate until the member resumes responding. Pauses are typically due to long garbage collection intervals. The value is specified using the `<maximum-packets>` element and defaults to `16` packets. A value of `0` disables pause detection.
- `<outstanding-packets>` – This setting is used to define the number of unconfirmed packets that are sent to a cluster member before packets addressed to that member are deferred. The value may be specified as either an explicit number by using the `<maximum-packets>` element, or as a range by using both a `<maximum-packets>` and `<minimum-packets>` elements. When a range is specified, this setting is dynamically adjusted based on network statistics. The maximum value should always be greater than 256 packets and defaults to 4096 packets. The minimum range should always be greater than 16 packets an defaults to 64 packets.

To adjust flow control behavior, edit the operational override file and add the `<pause-detection>` and `<outstanding-packets>` elements as follows:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
```

```
      <packet-publisher>
         <packet-delivery>
            <flow-control>
               <pause-detection>
                  <maximum-packets>32</maximum-packets>
               </pause-detection>
               <outstanding-packets>
                  <maximum-packets>2048</maximum-packets>
                  <minimum-packets>128</minimum-packets>
               </outstanding-packets>
            </flow-control>
         </packet-delivery>
      </packet-publisher>
   </cluster-config>
</coherence>
```

Disabling Packet Flow Control

To disable flow control, edit the operational override file and add an `<enabled>` element, within the `<flow-control>` element, that is set to `false`. For example

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <packet-publisher>
         <packet-delivery>
            <flow-control>
               <enabled>false</enabled>
            </flow-control>
         </packet-delivery>
      </packet-publisher>
   </cluster-config>
</coherence>
```

Adjusting Packet Traffic Jam Behavior

A packet traffic jam is when the number of pending packets that are enqueued by client threads for the packet publisher to transmit on the network grows to a level that the packet publisher considers intolerable. Traffic jam behavior is configured within the `<traffic-jam>` element. There are two settings that are used to adjust traffic jam behavior:

- `<maximum-packets>` – This setting controls the maximum number of pending packets that the packet publisher tolerates before determining that it is clogged and must slow down client requests (requests from local non-system threads). When the configured maximum packets limit is exceeded, client threads are forced to pause until the number of outstanding packets drops below the specified limit. This setting prevents most unexpected out-of-memory conditions by limiting the size of the resend queue. A value of `0` means no limit. The default value is `8192`.
- `<pause-milliseconds>` – This setting controls the number of milliseconds that the publisher pauses a client thread that is trying to send a message when the publisher is clogged. The publisher does not allow the message to go through until

the clog is gone, and repeatedly sleeps the thread for the duration specified by this property. The default value is `10`.

Specifying a packet 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. A warning is periodically logged if this condition is detected. Ideal values ensure that the publisher is never left without work to do, but at the same time prevent the queue from growing uncontrollably. The pause should be set short (10ms or under) and the limit on the number of packets be set high (that is, greater than 5000).

When the `<traffic-jam>` element is used with the `<flow-control>` element, the setting 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>` element's `<maximum-packets>` subelement value be greater than the `<maximum-packets>` value for the `<outstanding-packets>` element. When `<flow-control>` is disabled, the `<traffic-jam>` setting takes all outstanding packets into account.

To adjust the enqueue rate behavior, edit the operational override file and add the `<maximum-packets>` and <pause-milliseconds> elements as follows:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <packet-publisher>
         <traffic-jam>
            <maximum-packets>8192</maximum-packets>
            <pause-milliseconds>10</pause-milliseconds>
         </traffic-jam>
      </packet-publisher>
   </cluster-config>
</coherence>
```

Bundling Packets to Reduce Load

Multiple small packets can be bundled into a single larger packet to reduce the load on the network switching infrastructure. Packet bundling is configured within the `<packet-bundling>` element and includes the following settings:

- `<maximum-defferal-time>` – This setting specifies the maximum amount of time to defer a packet while waiting for additional packets to bundle. A value of zero results in the algorithm not waiting, and only bundling the readily accessible packets. A value greater than zero causes 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. The default value is `1us` (microsecond).
- `<agression-factor>` – This setting 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 allows for high packet utilization while keeping latency to a minimum. The default value is `0`.

The default packet-bundling settings are minimally aggressive allowing for bundling to occur without adding a measurable delay. The benefits of more aggressive bundling is based on the network infrastructure and the application object's typical data sizes and access patterns.

To adjust packet bundling behavior, edit the operational override file and add the <maximum-defferal-time> and `<agression-factor>` elements as follows:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <packet-publisher>
         <packet-delivery>
            <packet-bundling>
               <maximum-deferral-time>1us</maximum-deferral-time>
               <aggression-factor>0</aggression-factor>
            </packet-bundling>
         </packet-delivery>
      </packet-publisher>
   </cluster-config>
</coherence>
```

Changing Packet Retransmission Behavior

TCMP utilizes notification packets to acknowledge the receipt of packets which require confirmation. A positive acknowledgment (ACK) packet indicates that a packet was received correctly and that the packet must not be resent. Multiple ACKs for a given sender are batched into a single ACK packet to avoid wasting network bandwidth with many small ACK packets. Packets that have not been acknowledged are retransmitted based on the packet publisher's configured resend interval.

A negative acknowledgment (NACK) packet indicates that the packet was received incorrectly and causes the packet to be retransmitted. Negative acknowledgment is determined by inspecting packet ordering for packet loss. Negative acknowledgment causes a packet to be resent much quicker than relying on the publisher's resend interval. See "Disabling Negative Acknowledgments" on page 10-12 to disable negative acknowledgments.

The following topics are included in this section:

- Changing the Packet Resend Interval
- Changing the Packet Resend Timeout
- Configuring Packet Acknowledgment Delays

Changing the Packet Resend Interval

The packet resend interval specifies the minimum amount of time, in milliseconds, that the packet publisher waits for a corresponding ACK packet, before resending a packet. The default resend interval is `200` milliseconds.

To change the packet resend interval, edit the operational override file and add a `<resend-milliseconds>` element as follows:

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

```
<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <packet-publisher>
         <packet-delivery>
            <resend-milliseconds>400</resend-milliseconds>
         </packet-delivery>
      </packet-publisher>
   </cluster-config>
</coherence>
```

Changing the Packet Resend Timeout

The packet resend timeout interval specifies the maximum amount of time, in milliseconds, that a packet continues to be resent if no ACK packet is received. After this timeout expires, a determination is made if the recipient is to be considered terminated. This determination takes additional data into account, such as if other nodes are still able to communicate with the recipient. The default value is `300000` milliseconds. For production environments, the recommended value is the greater of 300000 and two times the maximum expected full GC duration.

> **Note:** The default death detection mechanism is the TCP-ring listener, which detects failed cluster members before the resend timeout interval is ever reached. See "Configuring Death Detection" on page 7-13 for more information on death detection.

To change the packet resend timeout interval, edit the operational override file and add a `<timeout-milliseconds>` element as follows:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <packet-publisher>
         <packet-delivery>
            <timeout-milliseconds>420000</timeout-milliseconds>
         </packet-delivery>
      </packet-publisher>
   </cluster-config>
</coherence>
```

Configuring Packet Acknowledgment Delays

The amount of time the packet publisher waits before sending ACK and NACK packets can be changed as required. The ACK and NACK packet delay intervals are configured within the `<notification-queueing>` eminent using the following settings:

- `<ack-delay-milliseconds>` – This element specifies the maximum number of milliseconds that the packet publisher delays before sending an ACK packet. The ACK packet may be transmitted earlier if multiple batched acknowledgments fills the ACK packet. This value should be set substantially lower then the remote member's packet delivery resend timeout to allow ample time for the ACK to be received and processed before the resend timeout expires. The default value is `16`.
- `<nack-delay-milliseconds>` – This element specifies the number of milliseconds that the packet publisher delays before sending a NACK packet. The default value is `1`.

To change the ACK and NACK delay intervals, edit the operational override file and add the `<ack-delay-milliseconds>` and `<nack-delay-milliseconds>` elements as follows:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <packet-publisher>
         <notification-queueing>
            <ack-delay-milliseconds>32</ack-delay-milliseconds>
            <nack-delay-milliseconds>1</nack-delay-milliseconds>
         </notification-queueing>
      </packet-publisher>
   </cluster-config>
</coherence>
```

Configuring the Transmission Packet Pool Size

The transmission packet pool is a buffer for use in transmitting UDP packets. Unlike the packet buffers, this buffer is internally managed by Coherence rather then the operating system and is allocated on the JVM's heap.

The packet pool is used as a reusable buffer between Coherence network services and allows for faster socket-layer processing at the expense of increased memory usage. The pool is initially empty and grows on demand up to the specified size limit; therefore, memory is reserved only when it is needed which allows the buffer to conserve memory.

The transmission packet pool size controls the maximum number of packets which can be queued on the packet speaker before the packet publisher must block. The pool is configured within the `<packet-publisher>` node using the `<packet-pool>` element. The `<size>` element is used to specify the maximum size of the pool. The value is entered in bytes. By default, the size is unspecified and the default value is `0`. A zero value indicates that the buffer is calculated by factoring the preferred MTU size with 2048. If a size is explicitly defined, then the number of packets is calculated as *`pool size/MTU size`*. See "Configuring the Incoming Handler's Packet Pool" on page 10-13 for instructions on configuring the incoming handler's packet pool size.

To configure the transmission packet pool size, edit the operational override file and add the `<packet-pool>` element as follows:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <packet-publisher>
         <packet-pool>
            <size>3072</size>
         </packet-pool>
      </packet-publisher>
   </cluster-config>
</coherence>
```

Configuring the Size of the Packet Buffers

Packet buffers are operating system buffers used by datagram sockets (also referred to as socket buffers). Packet buffers can be configured to control how many packets the operating system is requested to buffer. Packet buffers are used by unicast and multicast listeners (inbound buffers) and by the packet publisher (outbound buffer).

The following topics are included in this section:

- Understanding Packet Buffer Sizing
- Configuring the Outbound Packet Buffer Size
- Configuring the Inbound Packet Buffer Size

Understanding Packet Buffer Sizing

Packet buffer size can be configured based on either the number of packets or based on bytes using the following settings:

- `<maximum-packets>` – This setting specifies the number of packets (based on the configured packet size) that the datagram socket is asked to size itself to buffer. See `java.net.SocketOptions#SO_SNDBUF` and `java.net.SocketOptions#SO_RCVBUF` properties for additional details. Actual buffer sizes may be smaller if the underlying socket implementation cannot support more than a certain size.
- `<size>` – Specifies the requested size of the underlying socket buffer in bytes.

The operating system only treats the specified packet buffer size as a hint and is not required to allocate the specified amount. In the event that less space is allocated then requested, Coherence issues a warning and continues to operate with the constrained buffer, which may degrade performance. See *Oracle Coherence Administrator's Guide* for details on configuring your operating system to allow larger buffers.

Large inbound buffers can help insulate the Coherence network layer from JVM pauses that are caused by the Java Garbage Collector. While the JVM is paused, Coherence cannot dequeue packets from any inbound socket. If the pause is long enough to cause the packet buffer to overflow, the packet reception is delayed as the originating node must detect the packet loss and retransmit the packet(s).

Configuring the Outbound Packet Buffer Size

The outbound packet buffer is used by the packet publisher when transmitting packets. When making changes to the buffer size, performance should be evaluated both in terms of throughput and latency. A large buffer size may allow for increased throughput, while a smaller buffer size may allow for decreased latency.

To configure the outbound packet buffer size, edit the operational override file and add a <packet-buffer> element within the <packet-publisher> node and specify the packet buffer size using either the <size> element (for bytes) or the <maximum-packets> element (for packets). The default value is 32 packets. The following example demonstrates specifying the packet buffer size based on the number of packets:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <packet-publisher>
         <packet-buffer>
            <maximum-packets>64</maximum-packets>
         </packet-buffer>
      </packet-publisher>
   </cluster-config>
</coherence>
```

Configuring the Inbound Packet Buffer Size

The multicast listener and unicast listener each have their own inbound packet buffer. To configure an inbound packet buffer size, edit the operational override file and add a <packet-buffer> element (within either a <multicast-listener> or <unicast-listener> node, respectively) and specify the packet buffer size using either the <size> element (for bytes) or the <maximum-packets> element (for packets). The default value is 64 packets for the multicast listener and 1428 packets for the unicast listener.

The following example specifies the packet buffer size for the unicast listener and is entered using bytes:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <unicast-listener>
         <packet-buffer>
            <size>1500000</size>
         </packet-buffer>
      </unicast-listener>
   </cluster-config>
</coherence>
```

The following example specifies the packet buffer size for the multicast listener and is entered using packets:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
```

```
   <cluster-config>
      <packet-publisher>
         <packet-buffer>
            <maximum-packets>128</maximum-packets>
         </packet-buffer>
      </packet-publisher>
   </cluster-config>
</coherence>
```

Adjusting the Maximum Size of a Packet

The maximum and preferred UDP packet sizes can be adjusted to optimize the efficiency and throughput of cluster communication. All cluster nodes must use identical maximum packet sizes. For optimal network utilization, this value should be 32 bytes less then the network maximum transmission unit (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. The *COHRENCE_HOME*/bin/optimize.reg registration file contains the registry settings. See *Oracle Coherence Administrator's Guide* for details on setting the Datagram size on Windows.

Packet size is configured within the <packet-size> element and includes the following settings:

- <maximum-length> – Specifies the packet size, in bytes, which all cluster members can safely support. This value must be the same for all members in the cluster. A low value can artificially limit the maximum size of the cluster. This value should be at least 512. The default value is 64KB.
- <preferred-length> – Specifies the preferred size, in bytes, of the DatagramPacket objects that is sent and received on the unicast and multicast sockets.

  This value can be larger or smaller then the <maximum-length> value, and need not be the same for all cluster members. The ideal value is one which fits within the network MTU, leaving enough space for either the UDP or TCP packet headers, which are 32 and 52 bytes respectively.

  This value should be at least 512 and defaults to a value based on the local nodes MTU. An MTU of 1500 is assumed if the MTU cannot be obtained.

To adjust the packet size, edit the operational override file and add the <maximum-length> and <preferred-length> elements as follows:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <packet-publisher>
         <packet-size>
            <maximum-length>49152</maximum-length>
            <preferred-length>1500</preferred-length>
```

```
         </packet-size>
      </packet-publisher>
   </cluster-config>
</coherence>
```

Changing the Packet Speaker Volume Threshold

The packet speaker is responsible for sending packets on the network when the `packet-publisher` detects that a network send operation is likely to block. This allows the packet publisher to avoid blocking on I/O and continue to prepare outgoing packets. The packet publisher dynamically chooses whether to use the speaker as the packet load changes.

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.

The packet speaker is configured using the `<volume-threshold>` element to specify the minimum number of packets which must be ready to be sent for the speaker daemon to be activated. A value of `0` forces the speaker to always be used, while a very high value causes it to never be used. If the value is unspecified (the default), it is set to match the packet buffer.

To specify the packet speaker volume threshold, edit the operational override file and add the `<volume-threshold>` element as follows:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <packet-speaker>
         <volume-threshold>
            <minimum-packets>32</minimum-packets>
         </volume-threshold>
      </packet-speaker>
   </cluster-config>
</coherence>
```

Changing Message Handler Behavior

Cluster services transmit and receive data using message handlers. There is handler for processing incoming data and a handler for processing outgoing data. Both handlers have settings that can be configured as required.

The following topics are included in this section:

- Configuring the Incoming Message Handler
- Configuring the Outgoing Message Handler

Configuring the Incoming Message Handler

The incoming message handler assembles UDP packets into logical messages and dispatches them to the appropriate Coherence service for processing. The incoming message handler is configured within the `<incoming-message-handler>` element.

The following topics are included in this section:

- Changing the Time Variance
- Disabling Negative Acknowledgments
- Configuring the Incoming Handler's Packet Pool

Changing the Time Variance

The `<maximum-time-variance>` element 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 is closer between multiple systems running in the cluster; however, the process of joining the cluster is 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, a larger value may be necessary. The default value is `16`.

To change the maximum time variance, edit the operational override file and add the `<maximum-time-variance>` element as follows:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <incoming-message-handler>
         <maximum-time-variance>16</maximum-time-variance>
      </incoming-message-handler>
   </cluster-config>
</coherence>
```

Disabling Negative Acknowledgments

Negative acknowledgments can be disabled for the incoming message handler. When disabled, the handler does not notify the packet sender if packets were received incorrectly. In this case, the packet sender waits the specified resend timeout interval before resending the packet. See "Changing Packet Retransmission Behavior" on page 10-5 for more information on packet acknowledgments.

To disable negative acknowledgment, edit the operational override file and add a `<use-nack-packets>` element that is set to `false`. For example:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <incoming-message-handler>
```

```
         <use-nack-packets>false</use-nack-packets>
      </incoming-message-handler>
   </cluster-config>
</coherence>
```

Configuring the Incoming Handler's Packet Pool

The incoming packet pool is a buffer for use in receiving UDP packets. Unlike the packet buffers, this buffer is internally managed by Coherence rather then the operating system and is allocated on the JVM's heap.

The packet pool is used as a reusable buffer between Coherence network services and allows for faster socket-layer processing at the expense of increased memory usage. The pool is initially empty and grows on demand up to the specified size limit; therefore, memory is reserved only when it is needed which allows the buffer to conserve memory.

The incoming handler's packet pool size controls the number of packets which can be queued before the unicast listener and multicast listener must block. The pool is configured within the `<incoming-message-handler>` node using the `<packet-pool>` element. The `<size>` element is used to specify the maximum size of the pool. The value is entered in bytes. By default, the size is unspecified and the default value is `0`. A zero value indicates that the buffer is calculated by factoring the preferred MTU size with 2048. If a size is explicitly defined, then the number of packets is calculated as *`pool size/MTU size`*. See "Configuring the Transmission Packet Pool Size" on page 10-7 for instructions on configuring the packet pool size used to transmit packets.

To configure the incoming handler's packet pool size, edit the operational override file and add the `<size>` element as follows:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <incoming-message-handler>
         <packet-pool>
            <size>3072</size>
         </packet-pool>
      </incoming-message-handler>
   </cluster-config>
</coherence>
```

Configuring the Outgoing Message Handler

The outgoing message handler is used by cluster services to process messages that are to be transmitted. The outgoing message handler uses a specialized message pool whose size can be configured as required. The outgoing message handler is configured within the `<outgoing-message-handler>` element.

Configuring the Outgoing Handler's Message Pool

The outgoing message handler uses a message pool to control how many message buffers are pooled for message transmission. Pooling message buffers relieves the

pressure on the JVM garbage collector by pooling the memory resources needed for messaging.

The message pool contains any number of segments of a specified size. For example, a pool with 4 segments and a segment size of 10MB can hold, at most, 40 MB of space for serialization. The number of segments and the segment size are defined using the `<segment>` and `<segment-size>` elements, respectively.

Each pool segment stores message buffers of a specific size. The smallest size buffer is defined by the `<min-buffer-size>` element. The next buffer size for the next segment is then calculated using bitwise left shift using the `<growth-factor>` value (`'min-buffer-size' << growth-factor`). A left shift by *n* is equivalent to multiplying by 2^n; where `n` is the growth factor value. For a growth factor of 2, multiply the minimum buffer size by 4. For a growth factory of 3, multiply the minimum buffer size by 8, and so on.

The following example shows the default pool values and results in a message pool that is 64MB in total size where: the first pool segment contains message buffers of 1KB; the second pool segment contains message buffers of 4KB; the third pool segment contains message buffers of 16KB; and the fourth pool segment contains message buffers of 64KB. Using the same default values but increasing the growth factor to 3, results in buffer sizes of 1KB, 8KB, 64KB, and 512KB.

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <outgoing-message-handler>
         <message-pool>
            <segments>4</segments>
            <segment-size>16MB</segment-size>
            <min-buffer-size>1KB</min-buffer-size>
            <growth-factor>2</growth-factor>
         </message-pool>
      </outgoing-message-handler>
   </cluster-config>
</coherence>
```

Space that is claimed for network buffers (in and out) and serialization buffers is periodically reclaimed when the capacity is higher than the actual usage.

Changing the TCMP Socket Provider Implementation

Coherence provides three underlying socket provider implementations for use by TCMP:

- system socket provider (default) – A socket provider that produces instances of the JVM's default socket and channel implementations.
- TCP socket provider – A socket provider that produces TCP-based sockets and channel implementations.
- SSL socket provider – A socket provider that produces socket and channel implementations which use SSL.

custom socket providers can also be enabled as required. Socket providers for use by TCMP are configured for the unicast listener within the `<unicast-listener>` element.

The following topics are included in this section:

- Using the TCP Socket Provider
- Using the SSL Socket Provider
- Enabling a Custom Socket Provider

Using the TCP Socket Provider

The TCP socket provider is a socket provider which, whenever possible, produces TCP-based sockets. This socket provider creates `DatagramSocket` instances which are backed by TCP. When used with the WKA feature (mulitcast disabled), TCMP functions entirely over TCP without the need for UDP.

> **Note:** if this socket provider is used without the WKA feature (multicast enabled), TCP is used for all unicast communications; while, multicast is utilized for group based communications.

The TCP socket provider uses up to two TCP connections between each pair of cluster members. No additional threads are added to manage the TCP traffic as it is all done using nonblocking NIO based sockets. Therefore, the existing TCMP threads handle all the connections. The connections are brought up on demand and are automatically reopened as needed if they get disconnected for any reason. Two connections are utilized because it reduces send/receive contention and noticeably improves performance. TCMP is largely unaware that it is using a reliable protocol and as such still manages guaranteed delivery and flow control.

To specify the TCP socket provider, edit the operational override file and add a `<socket-provider>` element that includes the `tcp` value. For example:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <unicast-listener>
         <socket-provider system-property="tangosol.coherence.socketprovider">tcp
         </socket-provider>
      </unicast-listener>
   </cluster-config>
</coherence>
```

The `tangosol.coherence.socketprovider` system property is used to specify the socket provider instead of using the operational override file. For example:

```
-Dtangosol.coherence.socketprovider=tcp
```

Using the SSL Socket Provider

The SSL socket provider is a socket provider which only produces SSL protected sockets. This socket provider creates `DatagramSocket` instances which are backed by

SSL/TCP. SSL is not supported for multicast sockets; therefore, the WKA feature (multicast disabled) must be used for TCMP to function with this provider.

The default SSL configuration allows for easy configuration of two-way SSL connections, based on peer trust where every trusted peer resides within a single JKS keystore. More elaborate configuration can be defined with alternate identity and trust managers to allow for Certificate Authority trust validation. See *Oracle Coherence Security Guide* for detailed instructions on configuring and using SSL with TCMP.

Enabling a Custom Socket Provider

Custom socket providers can be created and enabled for use by TCMP as required. Custom socket providers must implement the `com.tangosol.net.SocketProvider` interface. See *Oracle Coherence Java API Reference* for details on this API.

Custom socket providers are enabled within the `<socket-provider>` element using the `<instance>` element. The preferred approach is to use the `<socket-provider>` element to reference a custom socket provider configuration that is defined within the `<socket-providers>` node.

The following example demonstrates enabling a custom socket provider by referencing a provider named `mySocketProvider` which is implemented in the `MySocketProvider` class. The provider is referenced using the name defined in the `id` attribute.

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <unicast-listener>
         <socket-provider>mySocketProvider</socket-provider>
      </unicast-listener>

      <socket-providers>
         <socket-provider id="mySocketProvider">
            <instance>
               <class-name>package.MySocketProvider</class-name>
            </instance>
         </socket-provider>
      </socket-providers>
   </cluster-config>
</coherence>
```

As an alternative, the `<instance>` element supports the use of a `<class-factory-name>` element to use a factory class that is responsible for creating `SocketProvider` instances, and a `<method-name>` element to specify the static factory method on the factory class that performs object instantiation. The following example gets a custom socket provider instance using the `createProvider` method on the `MySocketProviderFactory` class.

```
<socket-providers>
   <socket-provider id="mySocketProvider">
      <instance>
         <class-factory-name>package.MySocketProviderFactory</class-factory-name>
         <method-name>createProvider</method-name>
      </instance>
```

```
   </socket-provider>
</socket-providers>
```

Any initialization parameters that are required for an implementation can be specified using the `<init-params>` element. The following example sets the `iMaxTimeout` parameter to `2000`.

```
<socket-providers>
   <socket-provider id="mySocketProvider">
      <instance>
         <class-name>package.MySocketProvider</class-name>
         <init-params>
            <init-param>
               <param-name>iMaxTimeout</param-name>
               <param-value>2000</param-value>
            </init-param>
         </init-params>
      </instance>
   </socket-provider>
</socket-providers>
```

Part III

Using Caches

Part III contains the following chapters:

11

Introduction to Caches

This chapter provides an overview and comparison of basic cache types offered by Coherence. The chapter includes the following sections:

Distributed Cache

A distributed, or partitioned, cache is a clustered, fault-tolerant cache that has linear scalability. Data is partitioned among all the computers of the cluster. For fault-tolerance, partitioned caches can be configured to keep each piece of data on one or more unique computers within a cluster. Distributed caches are the most commonly used caches in Coherence.

Coherence defines a distributed cache as a collection of data that is distributed (or, *partitioned*) across any number of cluster nodes such that exactly one node in the cluster is responsible for each piece of data in the cache, and the responsibility is distributed (or, load-balanced) among the cluster nodes.

There are several key points to consider about a distributed cache:

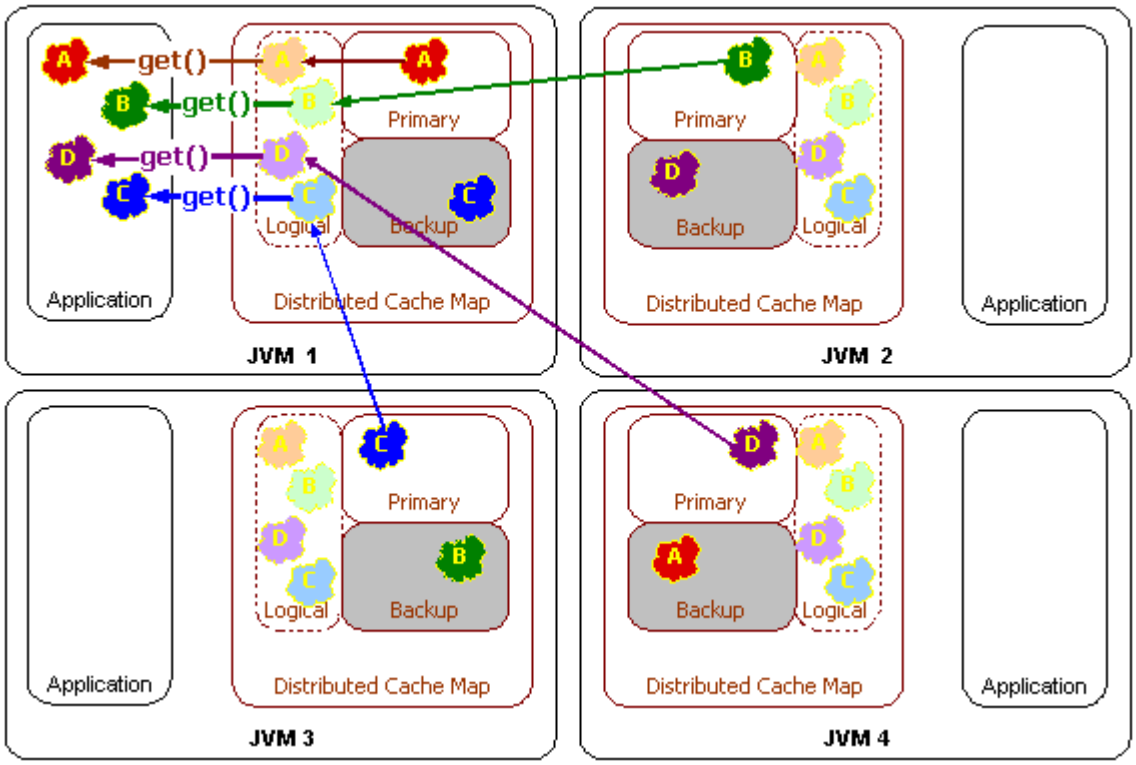
- **Partitioned:** The data in a distributed cache is spread out over all the servers in such a way that no two servers are responsible for the same piece of cached data. The size of the cache and the processing power associated with the management of the cache can grow linearly with the size of the cluster. Also, it means that operations against data in the cache can be accomplished with a "single hop," in other words, involving at most one other server.
- **Load-Balanced:** Since the data is spread out evenly over the servers, the responsibility for managing the data is automatically load-balanced across the cluster.
- **Location Transparency:** Although the data is spread out across cluster nodes, the exact same API is used to access the data, and the same behavior is provided by each of the API methods. This is called location transparency, which means that the developer does not have to code based on the topology of the cache, since the

API and its behavior is the same with a local JCache, a replicated cache, or a distributed cache.

- **Failover:** All Coherence services provide failover and failback without any data loss, and that includes the distributed cache service. The distributed cache service allows the number of backups to be configured; if the number of backups is one or higher, any cluster node can fail without the loss of data.

Access to the distributed cache often must go over the network to another cluster node. All other things equals, if there are n cluster nodes, (n - 1) / n operations go over the network:

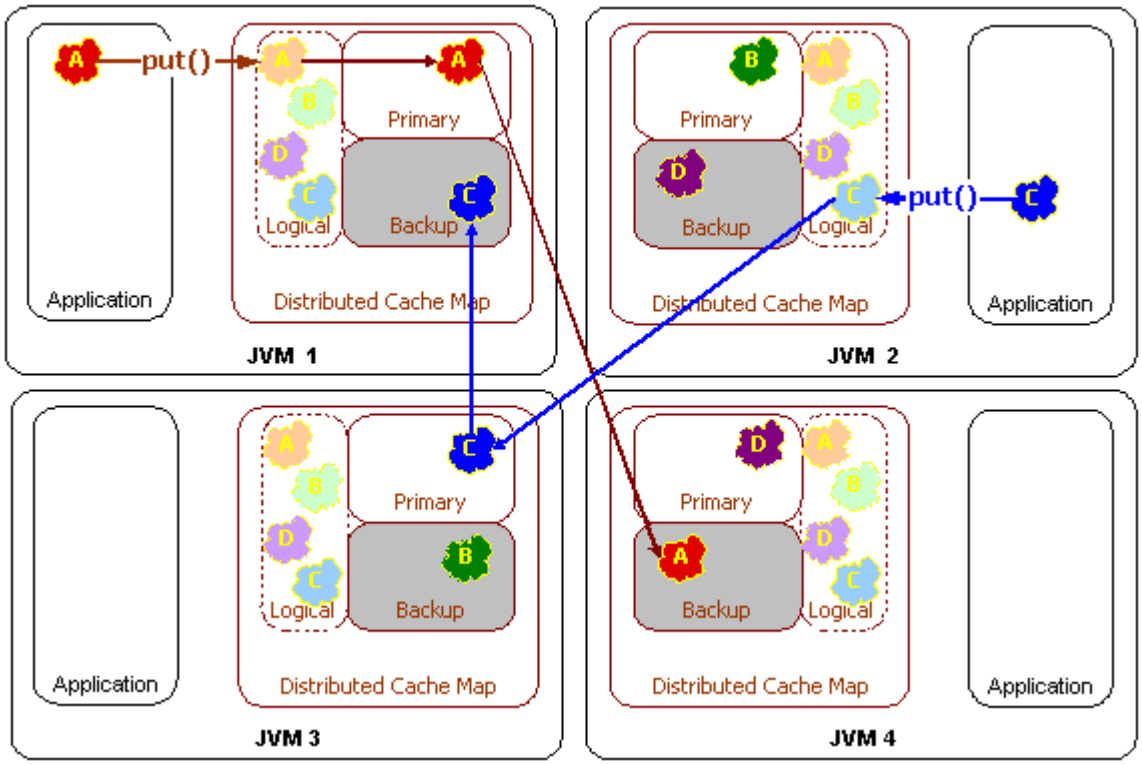
Figure 11–1 Get Operations in a Partitioned Cache Environment



Since each piece of data is managed by only one cluster node, an access over the network is only a "single hop" operation. This type of access is extremely scalable, since it can use point-to-point communication and thus take optimal advantage of a switched network.

Similarly, a cache update operation can use the same single-hop point-to-point approach, which addresses a known limitation of a replicated cache, the requirement to push cache updates to all cluster nodes.

Figure 11–2 Put Operations in a Partitioned Cache Environment

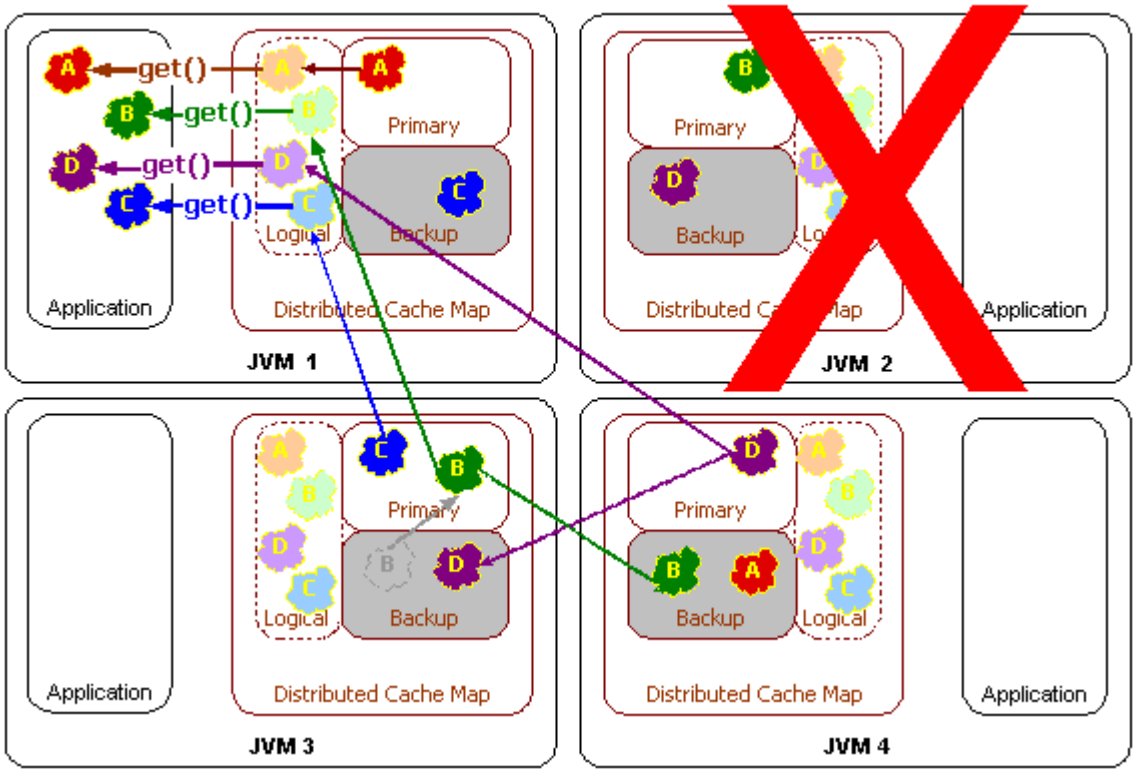


In the figure above, the data is being sent to a primary cluster node and a backup cluster node. This is for failover purposes, and corresponds to a backup count of one. (The default backup count setting is one.) If the cache data were not critical, which is to say that it could be re-loaded from disk, the backup count could be set to zero, which would allow some portion of the distributed cache data to be lost if a cluster node fails. If the cache were extremely critical, a higher backup count, such as two, could be used. The backup count only affects the performance of cache modifications, such as those made by adding, changing or removing cache entries.

Modifications to the cache are not considered complete until all backups have acknowledged receipt of the modification. There is a slight performance penalty for cache modifications when using the distributed cache backups; however it guarantees that if a cluster node were to unexpectedly fail, that data consistency is maintained and no data is lost.

Failover of a distributed cache involves promoting backup data to be primary storage. When a cluster node fails, all remaining cluster nodes determine what data each holds in backup that the failed cluster node had primary responsible for when it died. Those data becomes the responsibility of whatever cluster node was the backup for the data:

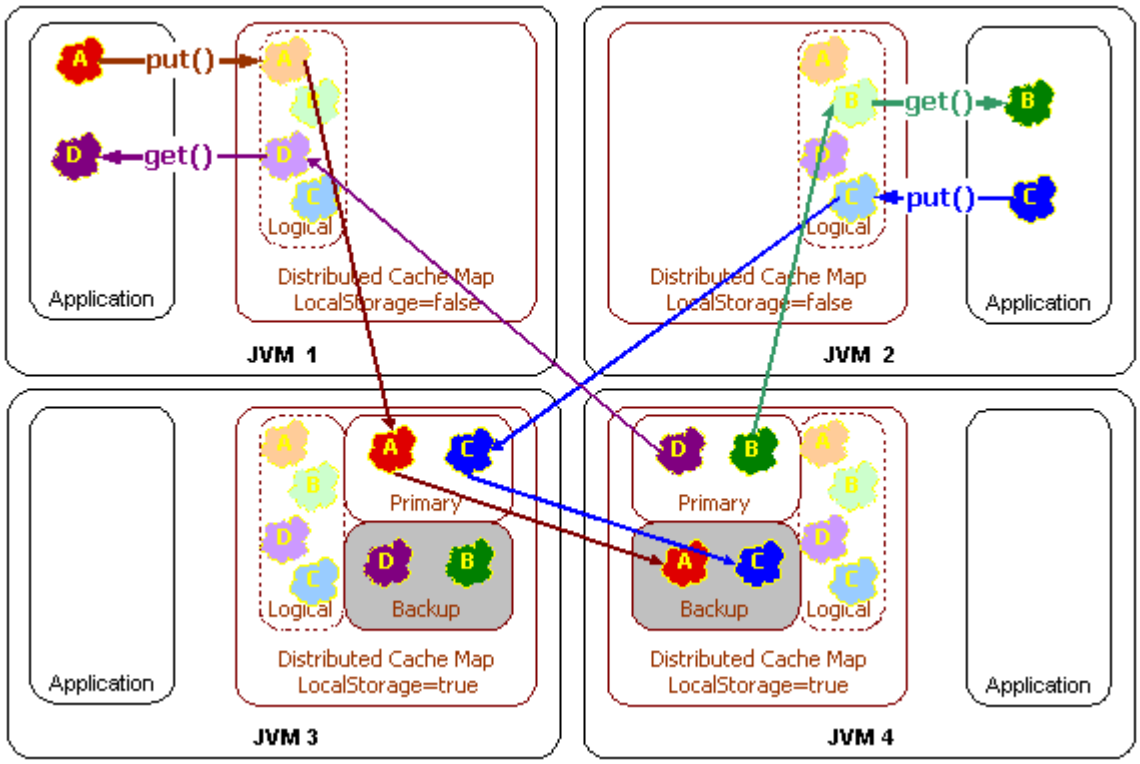
Figure 11–3 Failover in a Partitioned Cache Environment



If there are multiple levels of backup, the first backup becomes responsible for the data; the second backup becomes the new first backup, and so on. Just as with the replicated cache service, lock information is also retained with server failure; the sole exception is when the locks for the failed cluster node are automatically released.

The distributed cache service also allows certain cluster nodes to be configured to store data, and others to be configured to not store data. The name of this setting is *local storage enabled*. Cluster nodes that are configured with the local storage enabled option provides the cache storage and the backup storage for the distributed cache. Regardless of this setting, all cluster nodes have the same exact view of the data, due to location transparency.

Figure 11–4 Local Storage in a Partitioned Cache Environment



There are several benefits to the local storage enabled option:

- The Java heap size of the cluster nodes that have turned off local storage enabled are not affected at all by the amount of data in the cache, because that data is cached on other cluster nodes. This is particularly useful for application server processes running on older JVM versions with large Java heaps, because those processes often suffer from garbage collection pauses that grow exponentially with the size of the heap.
- Coherence allows each cluster node to run any supported version of the JVM. That means that cluster nodes with local storage enabled turned on could be running a newer JVM version that supports larger heap sizes, or Coherence's off-heap storage using the Java NIO features.
- The local storage enabled option allows some cluster nodes to be used just for storing the cache data; such cluster nodes are called Coherence cache servers. Cache servers are commonly used to scale up Coherence's distributed query functionality.

Replicated Cache

A replicated cache is a clustered, fault tolerant cache where data is fully replicated to every member in the cluster. This cache offers the fastest read performance 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 servers, adding servers does not increase aggregate cache capacity.

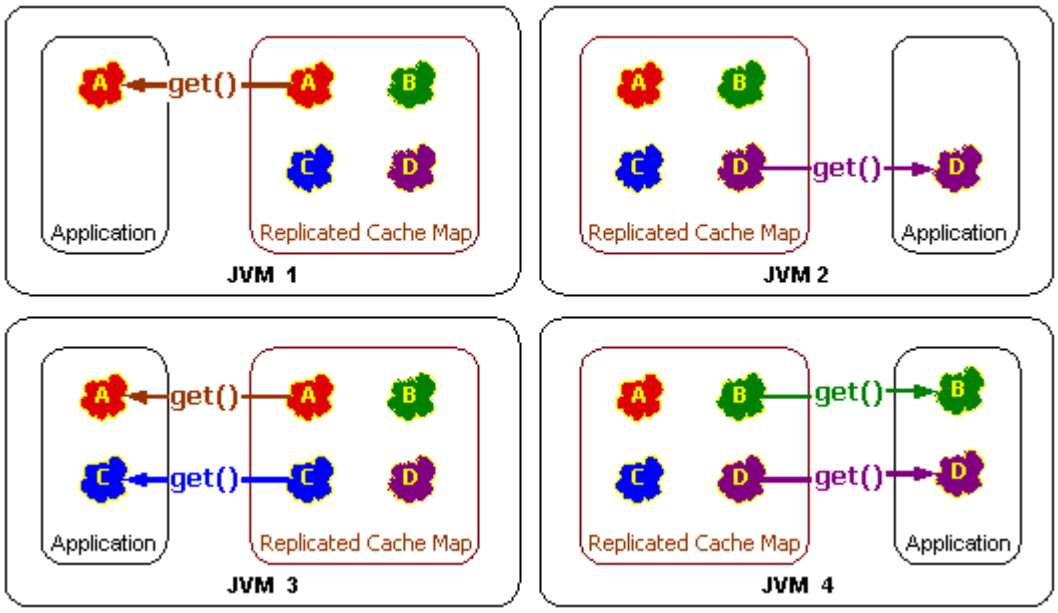
The replicated cache excels in its ability to handle data replication, concurrency control and failover in a cluster, all while delivering in-memory data access speeds. A

clustered replicated cache is exactly what it says it is: a cache that replicates its data to all cluster nodes.

There are several challenges to building a reliable replicated cache. The first is how to get it to scale and perform well. Updates to the cache have to be sent to all cluster nodes, and all cluster nodes have to end up with the same data, even if multiple updates to the same piece of data occur at the same time. Also, if a cluster node requests a lock, it should not have to get all cluster nodes to agree on the lock, otherwise it scales extremely poorly; yet with cluster node failure, all of the data and lock information must be kept safely. Coherence handles all of these scenarios transparently, and provides the most scalable and highly available replicated cache implementation available for Java applications.

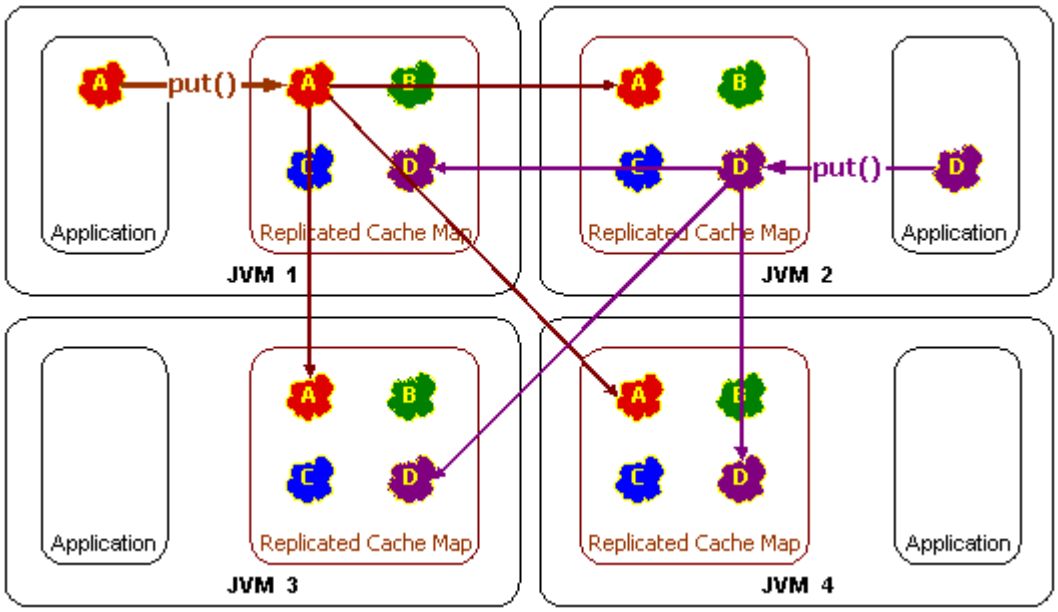
The best part of a replicated cache is its access speed. Since the data is replicated to each cluster node, it is available for use without any waiting. This is referred to as "zero latency access," and is perfect for situations in which an application requires the highest possible speed in its data access. Each cluster node (JVM) accesses the data from its own memory:

Figure 11–5 Get Operation in a Replicated Cache Environment



In contrast, updating a replicated cache requires pushing the new version of the data to all other cluster nodes:

Figure 11–6 Put Operation in a Replicated Cache Environment



Coherence implements its replicated cache service in such a way that all read-only operations occur locally, all concurrency control operations involve at most one other cluster node, and only update operations require communicating with all other cluster nodes. The result is excellent scalable performance, and as with all of the Coherence services, the replicated cache service provides transparent and complete failover and failback.

The limitations of the replicated cache service should also be carefully considered. First, however much data is managed by the replicated cache service is on each and every cluster node that has joined the service. That means that memory utilization (the Java heap size) is increased for each cluster node, which can impact performance. Secondly, replicated caches with a high incidence of updates do not scale linearly as the cluster grows; in other words, the cluster suffers diminishing returns as cluster nodes are added.

Optimistic Cache

An optimistic cache is a clustered cache implementation similar to the replicated cache implementation but without any concurrency control. This implementation offers higher write throughput than a replicated cache. It also allows 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, the store content held by each member may be different.

Near Cache

A near cache is a hybrid cache; it typically fronts a distributed cache or a remote cache with a local cache. Near cache invalidates front cache entries, using a configured 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.

The objective of a Near Cache is to provide the best of both worlds between the extreme performance of the Replicated Cache and the extreme scalability of the Distributed Cache by providing fast read access to Most Recently Used (MRU) and Most Frequently Used (MFU) data. Therefore, the Near Cache is an implementation that wraps two caches: a "front cache" and a "back cache" that automatically and transparently communicate with each other by using a read-through/write-through approach.

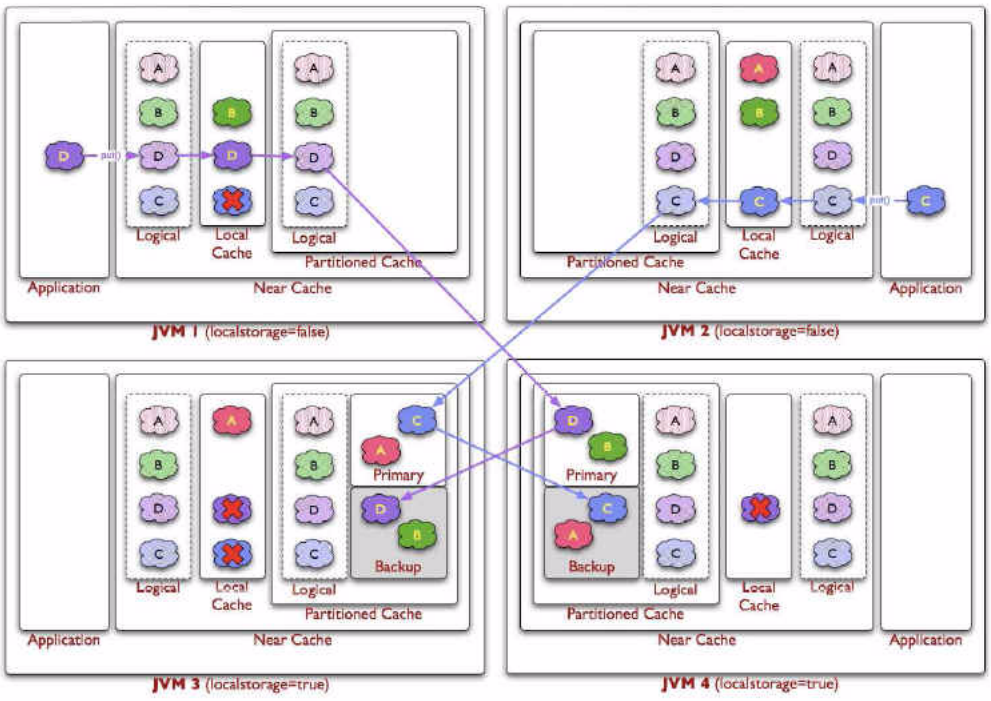
The "front cache" provides local cache access. It is assumed to be inexpensive, in that it is fast, and is limited in terms of size. The "back cache" can be a centralized or multitiered cache that can load-on-demand in case of local cache misses. The "back cache" is assumed to be complete and correct in that it has much higher capacity, but more expensive in terms of access speed. The use of a Near Cache is not confined to Coherence*Extend; it also works with TCMP.

This design allows Near Caches to configure cache coherency, from the most basic expiry-based caches and invalidation-based caches, up to advanced caches that version data and provide guaranteed coherency. The result is a tunable balance between the preservation of local memory resources and the performance benefits of truly local caches.

The typical deployment uses a Local Cache for the "front cache". A Local Cache is a reasonable choice because it is thread safe, highly concurrent, size-limited, auto-expiring, and stores the data in object form. For the "back cache", a remote, partitioned cache is used.

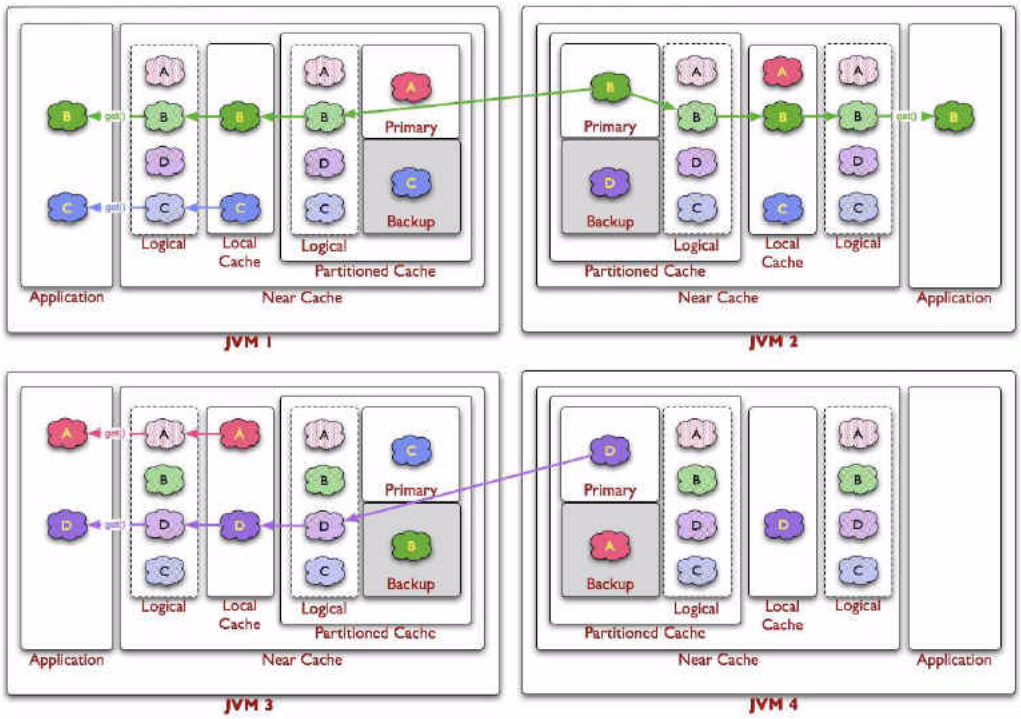
The following figure illustrates the data flow in a Near Cache. If the client writes an object D into the grid, the object is placed in the local cache inside the local JVM and in the partitioned cache which is backing it (including a backup copy). If the client requests the object, it can be obtained from the local, or "front cache", in object form with no latency.

Figure 11–7 Put Operations in a Near Cache Environment



If the client requests an object that has been expired or invalidated from the "front cache", then Coherence automatically retrieves the object from the partitioned cache. The "front cache" stores the object before the object is delivered to the client.

Figure 11–8 Get Operations in a Near Cache Environment



Local Cache

While it is not a clustered service, the Coherence local cache implementation is often used in combination with various Coherence clustered cache services. The Coherence local cache is just that: a cache that is local to (completely contained within) a particular cluster node. There are several attributes of the local cache that are particularly interesting:

- The local cache implements the same standard collections interface that the clustered caches implement, meaning that there is no programming difference between using a local or a clustered cache. Just like the clustered caches, the local cache is tracking to the JCache API, which itself is based on the same standard collections API that the local cache is based on.
- The local cache can be size-limited. The local cache can restrict the number of entries that it caches, and automatically evict entries when the cache becomes full. Furthermore, both the sizing of entries and the eviction policies can be customized. For example, the cache can be size-limited based on the memory used by the cached entries. The default eviction policy uses a combination of Most Frequently Used (MFU) and Most Recently Used (MRU) information, scaled on a logarithmic curve, to determine what cache items to evict. This algorithm is the best general-purpose eviction algorithm because it works well for short duration and long duration caches, and it balances frequency versus recentness to avoid cache thrashing. The pure LRU and pure LFU algorithms are also supported, and the ability to plug in custom eviction policies.

- The local cache supports automatic expiration of cached entries, meaning that each cache entry can be assigned a time to live in the cache.
- The local cache is thread safe and highly concurrent, allowing many threads to simultaneously access and update entries in the local cache.
- The local cache supports cache notifications. These notifications are provided for additions (entries that are put by the client, or automatically loaded into the cache), modifications (entries that are put by the client, or automatically reloaded), and deletions (entries that are removed by the client, or automatically expired, flushed, or evicted.) These are the same cache events supported by the clustered caches.
- The local cache maintains hit and miss statistics. These run-time statistics can accurately project the effectiveness of the cache, and adjust its size-limiting and auto-expiring settings accordingly while the cache is running.

The local cache is important to the clustered cache services for several reasons, including as part of Coherence's near cache technology, and with the modular backing map architecture.

Remote Cache

A remote cache describes any out of process cache accessed by a Coherence*Extend client. All cache requests are sent to a Coherence proxy where they are delegated to a cache (Replicated, Optimistic, Partitioned). See *Oracle Coherence Client Guide* for more information on using remote 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 11–1 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**

Table 11–1 (Cont.) Summary of Cache Types and Characteristics

	Replicated Cache	Optimistic Cache	Partitioned Cache	Near Cache backed by partitioned cache	LocalCache not clustered
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 100mb 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.

12

Configuring Caches

This chapter provides detailed instructions on how to configure caches within a cache configuration deployment descriptor. Refer to Appendix B, "Cache Configuration Elements," for a complete reference of all the elements available in the descriptor. In addition, see Chapter 17, "Cache Configurations by Example," for various sample cache configurations.

The following sections are included in this chapter:

Overview

Caches are configured in a cache configuration deployment descriptor. By default, Coherence attempts to load the first `coherence-cache-config.xml` deployment descriptor that is found in the classpath. Coherence includes a sample `coherence-cache-config.xml` file in the `coherence.jar`. To use a different `coherence-cache-config.xml` file, the file must be located on the classpath and must be loaded before the `coherence.jar` library; otherwise, the sample cache configuration deployment descriptor is used. See "Specifying a Cache Configuration File" on page 3-6 for alternate methods that are available for specifying a cache configuration deployment descriptor.

The cache configuration descriptor allows caches to be defined independently from the application code. At run time, applications get an instance of a cache by referring to a cache using the name that is defined in the descriptor. This allows application code to be written independent of the cache definition. Based on this approach, cache definitions can be modified without making any changes to the application code. This approach also maximizes cache definition reuse.

The schema definition of the cache configuration descriptor is the `coherence-cache-config.xsd` file, which imports the `coherence-cache-config-base.xsd` file, which, in turn, implicitly imports the `coherence-config-base.xsd` file. This file is located in the root of the `coherence.jar` file. A cache configuration deployment descriptor consists of two primary elements that are detailed in this chapter: the `<caching-scheme-mapping>`

element and the <caching-schemes> element. These elements are used to define caches schemes and to define cache names that map to the cache schemes.

Defining Cache Mappings

Cache mappings map a cache name to a cache scheme definition. The mappings provide a level of separation between applications and the underlying cache definitions. The separation allows cache implementations to be changed as required without having to change application code. Cache mappings can also be used to set initialization parameters that are applied to the underlying cache scheme definition.

Cache mappings are defined using a <cache-mapping> element within the <cache-scheme-mapping> node. Any number of cache mappings can be created. The cache mapping must include the cache name and the scheme name to which the cache name is mapped. See "cache-mapping" on page B-21 for a detailed reference of the <cache-mappings> element.

Using One-to-One Cache Mappings

One-to-one cache mappings map a specific cache name to a cache scheme definition. An applications must provide the exact name as specified in the mapping to use a cache. Example 12–1 creates a single cache mapping that maps the cache name example to a distributed cache scheme definition with the scheme name distributed.

Example 12–1 Sample One-to-One Cache Mapping

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <caching-scheme-mapping>
      <cache-mapping>
         <cache-name>example</cache-name>
         <scheme-name>distributed</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <distributed-scheme>
         <scheme-name>distributed</scheme-name>
      </distributed-scheme>
   </caching-schemes>
</cache-config>
```

Using Cache Name Pattern Mappings

Cache name pattern mappings allow applications to use patterns when specifying a cache name. Patterns use the asterisk (*) wildcard. Cache name patterns alleviate an application from having to know the exact name of a cache. Example 12–2 creates two cache mappings. The first mapping uses the wildcard (*) to map any cache name to a distributed cache scheme definition with the scheme name distributed. The second

mapping maps the cache name pattern `account-*` to the cache scheme definition with the scheme name `account-distributed`.

Example 12–2 Sample Cache Name Pattern Mapping

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <caching-scheme-mapping>
      <cache-mapping>
         <cache-name>*</cache-name>
         <scheme-name>distributed</scheme-name>
      </cache-mapping>
      <cache-mapping>
         <cache-name>account-*</cache-name>
         <scheme-name>account-distributed</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <distributed-scheme>
         <scheme-name>distributed</scheme-name>
      </distributed-scheme>
      <distributed-scheme>
         <scheme-name>account-distributed</scheme-name>
      </distributed-scheme>
   </caching-schemes>
</cache-config>
```

For the first mapping, an application can use any name when creating a cache and the name is mapped to the cache scheme definition with the scheme name `distributed`. The second mapping requires an application to use a pattern when specifying a cache name. In this case, an application must use the prefix `account-` before the name. For example, an application that specifies `account-overdue` as the cache name uses the cache scheme definition with the scheme name `account-distributed`.

Specifying Initialization Parameters in a Mapping

Cache mappings support the use of initialization parameters to override the properties of the underlying cache scheme definition. Initialization parameters are typically used to facilitate cache scheme definition reuse. In such cases, multiple cache names map to the same cache scheme definition, but each mapping overrides cache properties as required.

Initialization parameters are defined using an `<init-param>` element within the `<init-params>` node. The `<init-param>` element must include the `<param-name>` element and the `<param-value>` element. Any number of parameters can be specified. See "init-param" on page B-46 for a detailed reference of the `<init-param>` element.

Example 12–3 creates two cache mappings that map to the same cache scheme definition. However, the first mapping overrides the `back-size-limit` property on the underlying cache scheme definition; while, the second mapping uses the `back-size-limit` as configured in the underlying cache scheme definition.

Example 12–3 Initialization Parameters in a Cache Mapping

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <caching-scheme-mapping>
      <cache-mapping>
         <cache-name>*</cache-name>
         <scheme-name>distributed</scheme-name>
         <init-params>
            <init-param>
               <param-name>back-size-limit</param-name>
               <param-value>8MB</param-value>
            </init-param>
         </init-params>
      </cache-mapping>
      <cache-mapping>
         <cache-name>account-*</cache-name>
         <scheme-name>distributed</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>
   ...
</cache-config>
```

See "Using Cache Scheme Properties" on page 12-11 for more information on how cache scheme properties are configured for a cache scheme definition.

Defining Cache Schemes

Cache schemes are used to define the caches that are available to an application. Cache schemes provide a declarative mechanism that allows caches to be defined independent of the applications that use them. This removes the responsibility of defining caches from the application and allows caches to change without having to change an application's code. Cache schemes also promote cache definition reuse by allowing many applications to use the same cache definition.

Cache schemes are defined within the `<caching-schemes>` element. Each cache type (distributed, replicated, and so on) has a corresponding scheme element and properties that are used to define a cache of that type. Cache schemes can also be nested to allow further customized and composite caches such as near caches. See "caching-schemes" on page B-25 for a detailed reference of the `<caching-schemes>` element.

This section describes how to define cache schemes for the most often used cache types and does not represent the full set of cache types provided by Coherence. Instructions for defining cache schemes for additional cache types are found throughout this guide and are discussed as part of the features that they support. The following topics are included in this section:

- Defining Distributed Cache Schemes
- Defining Replicated Cache Schemes
- Defining Optimistic Cache Schemes
- Defining Local Cache Schemes
- Defining Near Cache Schemes

Defining Distributed Cache Schemes

The <distributed-scheme> element is used to define distributed caches. A distributed cache utilizes a distributed (partitioned) cache service instance. Any number of distributed caches can be defined in a cache configuration file. See "distributed-scheme" on page B-30 for a detailed reference of the <distributed-scheme> element.

Example 12–4 defines a basic distributed cache that uses distributed as the scheme name and is mapped to the cache name example. The <autostart> element is set to true to start the service on a cache server node.

Example 12–4 Sample Distributed Cache Definition

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <caching-scheme-mapping>
      <cache-mapping>
         <cache-name>example</cache-name>
         <scheme-name>distributed</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <distributed-scheme>
         <scheme-name>distributed</scheme-name>
         <backing-map-scheme>
            <local-scheme/>
         </backing-map-scheme>
         <autostart>true</autostart>
      </distributed-scheme>
   </caching-schemes>
</cache-config>
```

In the example, the distributed cache defines a local cache to be used as the backing map. See Chapter 13, "Implementing Storage and Backing Maps" for more information on configuring backing maps.

Defining Replicated Cache Schemes

The <replicated-scheme> element is used to define replicated caches. A replicated cache utilizes a replicated cache service instance. Any number of replicated caches can be defined in a cache configuration file. See "replicated-scheme" on page B-96 for a detailed reference of the <replicated-scheme> element.

Example 12–5 defines a basic replicated cache that uses replicated as the scheme name and is mapped to the cache name example. The <autostart> element is set to true to start the service on a cache server node.

Example 12–5 Sample Replicated Cache Definition

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
```

```
   coherence-cache-config.xsd">
   <caching-scheme-mapping>
      <cache-mapping>
         <cache-name>example</cache-name>
         <scheme-name>replicated</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <replicated-scheme>
         <scheme-name>replicated</scheme-name>
         <backing-map-scheme>
            <local-scheme/>
         </backing-map-scheme>
         <autostart>true</autostart>
      </replicated-scheme>
   </caching-schemes>
</cache-config>
```

In the example, the replicated cache defines a local cache to be used as the backing map. See Chapter 13, "Implementing Storage and Backing Maps" for more information on configuring backing maps.

Defining Optimistic Cache Schemes

The `<optimistic-scheme>` element is used to define optimistic caches. An optimistic cache utilizes an optimistic cache service instance. Any number of optimistic caches can be defined in a cache configuration file. See "optimistic-scheme" on page B-69 for a detailed reference of the `<optimistic-scheme>` element.

Example 12–6 defines a basic optimistic cache that uses `optimistic` as the scheme name and is mapped to the cache name `example`. The `<autostart>` element is set to `true` to start the service on a cache server node.

Example 12–6 Sample Optimistic Cache Definition

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <caching-scheme-mapping>
      <cache-mapping>
         <cache-name>example</cache-name>
         <scheme-name>optimistic</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <optimistic-scheme>
         <scheme-name>optimistic</scheme-name>
         <backing-map-scheme>
            <local-scheme/>
         </backing-map-scheme>
         <autostart>true</autostart>
      </optimistic-scheme>
   </caching-schemes>
</cache-config>
```

In the example, the optimistic cache defines a local cache to be used as the backing map. See Chapter 13, "Implementing Storage and Backing Maps" for more information on configuring backing maps.

Defining Local Cache Schemes

The `<local-scheme>` element is used to define local caches. Local caches are generally nested within other cache schemes, for instance as the front-tier of a near cache. Thus, this element can appear as a sub-element of any of the following elements: `<caching-schemes>`, `<distributed-scheme>`, `<replicated-scheme>`, `<optimistic-scheme>`, `<near-scheme>`, `<overflow-scheme>`, `<read-write-backing-map-scheme>`, and `<backing-map-scheme>`. See "local-scheme" on page B-60 for a detailed reference of the `<local-scheme>` element.

Example 12–7 defines a local cache that uses `local` as the scheme name and is mapped to the cache name `example`.

> **Note:** A local cache is not typically used as a standalone cache on a cache server; moreover, a cache server does not start if the only cache definition in the cache configuration file is a local cache.

Example 12–7 Sample Local Cache Definition

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>example</cache-name>
      <scheme-name>local</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <local-scheme>
      <scheme-name>local</scheme-name>
      <eviction-policy>LRU</eviction-policy>
      <high-units>32000</high-units>
      <low-units>10</low-units>
      <unit-calculator>FIXED</unit-calculator>
      <expiry-delay>10ms</expiry-delay>
    </local-scheme>
  </caching-schemes>
</cache-config>
```

See "Defining a Local Cache for C++ Clients" and "Configuring a Local Cache for .NET Clients" in the *Oracle Coherence Client Guide* when using Coherence*Extend.

Controlling the Growth of a Local Cache

As shown in Table 12–7, the `<local-scheme>` provides several optional sub-elements that control the growth of the cache. For example, the `<low-units>` and `<high-units>` sub-elements limit the cache in terms of size. When the cache reaches its maximum allowable size it prunes itself back to a specified smaller size,

choosing which entries to evict according to a specified eviction-policy (<eviction-policy>). The entries and size limitations are measured in terms of units as calculated by the scheme's unit-calculator (<unit-calculator>).

Local caches use the <expiry-delay> cache configuration element to configure the amount of time that items may remain in the cache before they expire. Client threads initiate these actions while accessing the cache. Therefore, the <expiry-delay> may be reached, but not initiated until a client thread accesses the cache. For example, if the <expiry-delay> value is set at 10 seconds (10s) and a client accesses the cache after 15 seconds, then expiry occurs after 15 seconds.

Note: The client thread performs the evictions, not a background thread. In addition, the expiry delay parameter (cExpiryMillis) is defined as an integer and is expressed in milliseconds. Therefore, the maximum amount of time can never exceed Integer.MAX_VALUE (2147483647) milliseconds or approximately 24 days.

Defining Near Cache Schemes

The <near-scheme> element is used to define a near cache. A near cache is a composite cache because it contains two caches: the <front-scheme> element is used to define a local (front-tier) cache and the <back-scheme> element is used to define a (back-tier) cache. Typically, a local cache is used for the front-tier, however, the front-tier can also use schemes based on Java Objects (using the <class-scheme>) and non-JVM heap-based caches (using <external-scheme> or <paged-external-scheme>). The back-tier cache is described by the <back-scheme> element. A back-tier cache can be any clustered cache type and any of the standalone cache types. See "near-scheme" on page B-63 for a detailed reference of the <near-scheme> element.

Example 12–8 defines of a near cache that uses near as the scheme name and is mapped to the cache name example. The front-tier is a local cache and the back-tier is a distributed cache.

Note: Near caches are used for cache clients and are not typically used on a cache server; moreover, a cache server does not start if the only cache definition in the cache configuration file is a near cache.

Example 12–8 Sample Near Cache Definition

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <caching-scheme-mapping>
      <cache-mapping>
         <cache-name>example</cache-name>
         <scheme-name>near</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <near-scheme>
         <scheme-name>near</scheme-name>
```

```
            <front-scheme>
               <local-scheme/>
            </front-scheme>
            <back-scheme>
               <distributed-scheme>
                  <scheme-name>near-distributed</scheme-name>
                  <backing-map-scheme>
                     <local-scheme/>
                  </backing-map-scheme>
                  <autostart>true</autostart>
               </distributed-scheme>
            </back-scheme>
         </near-scheme>
      </caching-schemes>
</cache-config>
```

See "Defining a Near Cache for C++ Clients" and "Defining a Near Cache for .NET Clients" in the *Oracle Coherence Client Guide* when using Coherence*Extend.

Near Cache Invalidation Strategies

The `<invalidation-strategy>` is an optional subelement for a near cache. An invalidation strategy is used to specify how the front-tier and back-tier objects are kept synchronous. A near cache can be configured to listen to certain events in the back cache and automatically update or invalidate entries in the front cache. Depending on the interface that the back cache implements, the near cache provides four different strategies of invalidating the front cache entries that have changed by other processes in the back cache.

Table 12–1 describes the invalidation strategies. You can find more information on the invalidation strategies and the read-through/write-through approach in Chapter 14, "Caching Data Sources."

Table 12–1 Near Cache Invalidation Strategies

Strategy Name	Description
None	This strategy 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 for the front cache.
Present	This strategy instructs a near cache to listen to the back cache events related only to the items currently present in the front cache. This strategy works best when each instance of a front cache contains distinct subset of data relative to the other front cache instances (for example, sticky data access patterns).
All	This strategy instructs a near cache to listen to all back cache events. This strategy is optimal for read-heavy tiered access patterns where there is significant overlap between the different instances of front caches.
Auto	This strategy instructs a near cache to switch automatically between `Present` and `All` strategies based on the cache statistics.

Using Scheme Inheritance

Scheme inheritance allows cache schemes to be created by inheriting another scheme and selectively overriding the inherited scheme's properties as required. This

flexibility enables cache schemes to be easily maintained and promotes cache scheme reuse. The <scheme-ref> element is used within a cache scheme definition and specifies the name of the cache scheme from which to inherit.

Example 12–9 creates two distributed cache schemes that are equivalent. The first explicitly configures a local scheme to be used for the backing map. The second definition use the <scheme-ref> element to inherit a local scheme named LocalSizeLimited:

Example 12–9 Using Cache Scheme References

```
<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>
```

In Example 12–9, the first distributed scheme definition is more compact; however, the second definition offers the ability to easily reuse the LocalSizeLimited scheme within multiple schemes. Example 12–10 demonstrates multiple schemes reusing the same LocalSizeLimited base definition and overriding the expiry-delay property.

Example 12–10 Multiple Cache Schemes Using Scheme Inheritance

```
<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>
```

Using Cache Scheme Properties

Cache scheme properties modify cache behavior as required for a particular application. Each cache scheme type contains its own set of properties that are valid for the cache. Cache properties are set within a cache scheme definition using their respective elements. See Appendix B, "Cache Configuration Elements," for a reference of all the properties that are supported for each cache scheme type.

Many cache properties use default values unless a different value is explicitly given within the cache scheme definition. The clustered caches (distributed, replicated and optimistic) use the default values as specified by their respective cache service definition. Cache services are defined in the operational deployment descriptor. While it is possible to change property values using an operational override file, cache properties are most often set within the cache scheme definition.

Example 12–11 creates a basic distributed cache scheme that sets the service thread count property and the request timeout property. In addition, the local scheme that is used for the backing map sets properties to limit the size of the local cache. Instructions for using cache scheme properties are found throughout this guide and are discussed as part of the features that they support.

Example 12–11 Setting Cache Properties

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <caching-scheme-mapping>
      <cache-mapping>
         <cache-name>example</cache-name>
         <scheme-name>DistributedInMemoryCache</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <distributed-scheme>
         <scheme-name>DistributedInMemoryCache</scheme-name>
         <service-name>DistributedCache</service-name>
         <thread-count>4</thread-count>
         <request-timeout>60s</request-timeout>
```

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

Using Parameter Macros

The cache configuration deployment descriptor 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 12–2 describes the parameter macros that may be specified:

Table 12–2 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 12–2 (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 12–2 (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.

13

Implementing Storage and Backing Maps

This chapter provides information on storage using backing maps. The following sections are included in this chapter:

Cache Layers

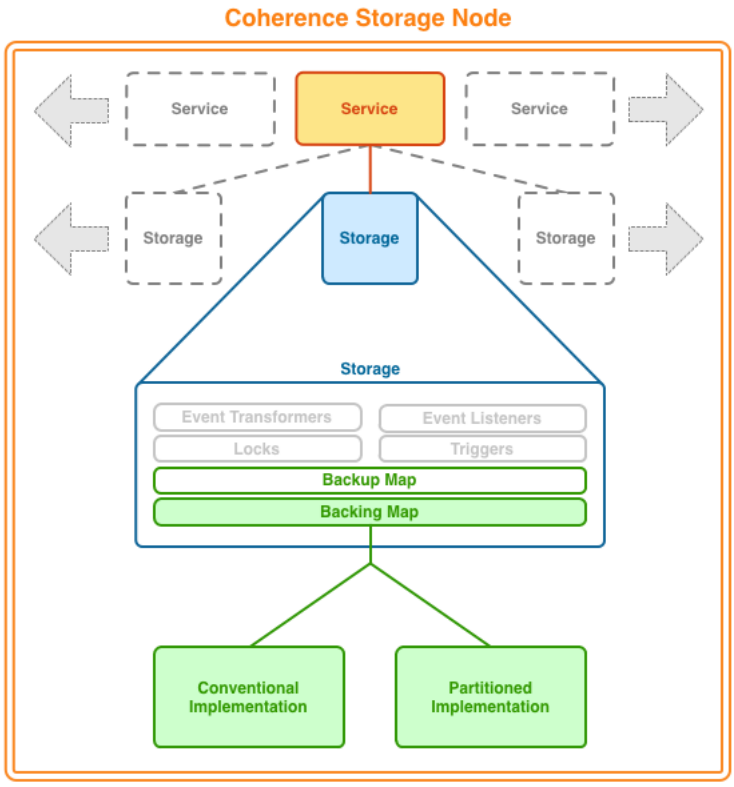
Partitioned (Distributed) cache service in Coherence has three distinct layers:

- **Client View** – The client view represents a virtual layer that provides access to the underlying partitioned data. Access to this tier is provided using the `NamedCache` interface. In this layer you can also create synthetic data structures such as `NearCache` or `ContinuousQueryCache`.
- **Storage Manager** – The storage manager is the server-side tier that is responsible for processing cache-related requests from the client tier. It manages the data structures that hold the actual cache data (primary and backup copies) and information about locks, event listeners, map triggers, and son.
- **Backing Map** – The Backing Map is the server-side data structure that holds actual data.

Coherence allows users to configure some out-of-the-box backing map implementations and custom ones. Basically, the only constraint that all these Map implementation have to be aware of is the understanding that the Storage Manager provides all keys and values in internal (Binary) format. To deal with conversions of that internal data to and from an Object format, the Storage Manager can supply Backing Map implementations with a `BackingMapManagerContext` reference.

Figure 13–1 shows a conceptual view of backing maps.

Figure 13–1 Backing Map Storage



Local Storage

Local storage refers to the data structures that actually store or cache the data that is managed by Coherence. For an object to provide local storage, it must support the same standard collections interface, `java.util.Map`. When a local storage implementation is used by Coherence to store replicated or distributed data, it is called a backing map because Coherence is actually backed by that local storage implementation. The other common uses of local storage is in front of a distributed cache and as a backup behind the distributed cache.

Coherence supports the following local storage implementations:

- **Safe HashMap:** This is the default lossless implementation. A lossless implementation is one, like Java's Hashtable class, that is neither size-limited nor auto-expiring. In other words, it is an implementation that never evicts ("loses") cache items on its own. This particular `HashMap` implementation is optimized for extremely high thread-level concurrency. For the default implementation, use class `com.tangosol.util.SafeHashMap`; when an implementation is required that provides cache events, use `com.tangosol.util.ObservableHashMap`. These implementations are thread-safe.
- **Local Cache:** This is the default size-limiting and auto-expiring implementation. The local cache is covered in more detail below, but the primary points to remember about it are that it can limit the size of the cache, and it can automatically expire cache items after a certain period. For the default implementation, use `com.tangosol.net.cache.LocalCache`; this implementation is thread safe and supports cache events, `com.tangosol.net.CacheLoader`, `CacheStore` and configurable/pluggable eviction policies.

- **Read/Write Backing Map:** This is the default backing map implementation for caches that load from a database on a cache miss. It can be configured as a read-only cache (consumer model) or as either a write-through or a write-behind cache (for the consumer/producer model). The write-through and write-behind modes are intended only for use with the distributed cache service. If used with a near cache and the near cache must be kept synchronous with the distributed cache, it is possible to combine the use of this backing map with a Seppuku-based near cache (for near cache invalidation purposes). For the default implementation, use class `com.tangosol.net.cache.ReadWriteBackingMap`.
- **Binary Map (Java NIO):** This is a backing map implementation that can store its information in memory but outside of the Java heap, or even in memory-mapped files, 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. This implementation is also available for distributed cache backups, which is particularly useful for read-mostly and read-only caches that require backup for high availability purposes, because it means that the backup does not affect the Java heap size yet it is immediately available in case of failover.
- **Serialization Map:** This is a backing map implementation that translates its data to a form that can be stored on disk, referred to as a serialized form. It requires a separate `com.tangosol.io.BinaryStore` object into which it stores the serialized form of the data; usually, this is the built-in LH disk store implementation, but the Serialization Map supports any custom implementation of `BinaryStore`. For the default implementation of Serialization Map, use `com.tangosol.net.cache.SerializationMap`.
- **Serialization Cache:** This is an extension of the `SerializationMap` that supports an LRU eviction policy. For example, a serialization cache can limit the size of disk files. For the default implementation of Serialization Cache, use `com.tangosol.net.cache.SerializationCache`.
- **Overflow Map:** An overflow map does not actually provide storage, but it deserves mention in this section because it can combine two local storage implementations so that when the first one fills up, it overflows into the second. For the default implementation of `OverflowMap`, use `com.tangosol.net.cache.OverflowMap`.

Operations

There are number of operation types performed against the Backing Map:

- Natural access and update operations caused by the application usage. For example, `NamedCache.get()` call naturally causes a `Map.get()` call on a corresponding Backing Map; the `NamedCache.invoke()` call may cause a sequence of `Map.get()` followed by the `Map.put()`; the `NamedCache.keySet(filter)` call may cause an `Map.entrySet().iterator()` loop, and so on.
- Remove operations caused by the time-based expiry or the size-based eviction. For example, a `NamedCache.get()` or `NamedCache.size()` call from the client tier could cause a `Map.remove()` call due to an entry expiry timeout; or `NamedCache.put()` call causing some `Map.remove()` calls (for different keys) caused by the total amount data in a backing map reaching the configured high water-mark value.
- Insert operations caused by a `CacheStore.load()` operation (for backing maps configured with read-through or read-ahead features)

- Synthetic access and updates caused by the partition distribution (which in turn could be caused by cluster nodes fail over or fail back). In this case, without any application tier call, some entries could be inserted or removed from the backing map.

Capacity Planning

Depending on the actual implementation, the Backing Map stores the cache data in the following ways:

- on-heap memory
- off-heap memory
- disk (memory-mapped files or in-process DB)
- solid state device (journal files)
- combination of any of the above

Keeping data in memory naturally provides dramatically smaller access and update latencies and is most commonly used.

More often than not, applications must ensure that the total amount of data placed into the data grid does not exceed some predetermined amount of memory. It could be done either directly by the application tier logic or automatically using size- or expiry-based eviction. Quite naturally, the total amount of data held in a Coherence cache equals the sum of data volume in all corresponding backing maps (one per each cluster node that runs the corresponding partitioned cache service in a storage enabled mode).

Consider following cache configuration excerpts:

```
<backing-map-scheme>
  <local-scheme/>
</backing-map-scheme>
```

The backing map above is an instance of `com.tangosol.net.cache.LocalCache` and does not have any pre-determined size constraints and has to be controlled explicitly. Failure to do so could cause the JVM to go out-of-memory.

```
<backing-map-scheme>
  <local-scheme>
    <eviction-policy>LRU</eviction-policy>
    <high-units>100m</high-units>
    <unit-calculator>BINARY</unit-calculator>
  </local-scheme>
</backing-map-scheme>
```

This backing map above is also a `com.tangosol.net.cache.LocalCache` and has a capacity limit of 100MB. As the total amount of data held by this backing map exceeds that high watermark, some entries are removed from the backing map, bringing the volume down to the low watermark value (`<low-units>` configuration element, witch defaults to 75% of the `<high-units>`). The choice of the removed entries is based on the LRU (Least Recently Used) eviction policy. Other options are LFU (Least Frequently Used) and Hybrid (a combination of the LRU and LFU). The value of `<high-units>` is limited to 2GB. To overcome that limitation (but maintain backward compatibility) Coherence uses the `<unit-factor>` element. For example, the `<high-units>` value of `8192` with a `<unit-factor>` of `1048576` results in a high watermark value of `8GB`.

```
<backing-map-scheme>
  <local-scheme>
    <expiry-delay>1h</expiry-delay>
  </local-scheme>
</backing-map-scheme>
```

The backing map above automatically evicts any entries that have not been updated for more than an hour. Note, that such an eviction is a "lazy" one and can happen any time after an hour since the last update happens; the only guarantee Coherence provides is that entries that exceed one hour are not returned to a caller.

The following backing map is an instance of `com.tangosol.net.cache.SerializationCache` which stores values in the extended (nio) memory and has a capacity limit of 100MB (100*1048576).

```
<backing-map-scheme>
  <external-scheme>
    <nio-memory-manager>
      <initial-size>1MB</initial-size>
      <maximum-size>100MB</maximum-size>
    </nio-memory-manager>
    <high-units>100</high-units>
    <unit-calculator>BINARY</unit-calculator>
    <unit-factor>1048576</unit-factor>
  </external-scheme>
</backing-map-scheme>
```

Configure a backup storage for this cache being `off-heap` (or `file-mapped`):

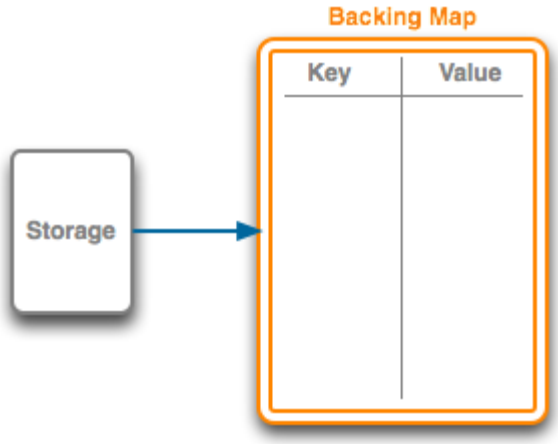
```
<backup-storage>
  <type>off-heap</type>
  <initial-size>1MB</initial-size>
  <maximum-size>100MB</maximum-size>
</backup-storage>
```

Using Partitioned Backing Maps

The conventional backing map implementation contained entries for all partitions owned by the corresponding node. (During partition transfer, it could also hold "in flight" entries that from the clients' perspective are temporarily not owned by anyone).

Figure 13–2 shows a conceptual view of the conventional backing map implementation.

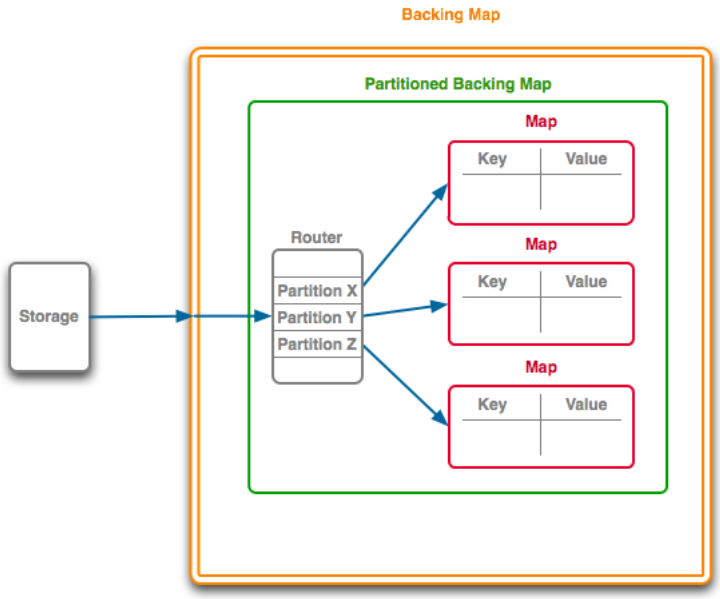
Figure 13–2 Conventional Backing Map Implementation



A partitioned backing map is basically a multiplexer of actual Map implementations, each of which would contain only entries that belong to the same partition.

Figure 13–3 shows a conceptual view of the partitioned backing map implementation.

Figure 13–3 Partitioned Backing Map Implementation



To configure a partitioned backing map, add a `<partitioned>` element with a value of `true`. For example:

```
<backing-map-scheme>
  <partitioned>true</partitioned>
  <external-scheme>
    <nio-memory-manager>
      <initial-size>1MB</initial-size>
      <maximum-size>50MB</maximum-size>
    </nio-memory-manager>
    <high-units>8192</high-units>
    <unit-calculator>BINARY</unit-calculator>
    <unit-factor>1048576</unit-factor>
```

```
  </external-scheme>
</backing-map-scheme>
```

This backing map is an instance of `com.tangosol.net.partition.PartitionSplittingBackingMap`, with individual partition holding maps being instances of `com.tangosol.net.cache.SerializationCache` that each store values in the extended (nio) memory. The individual nio buffers have a limit of 50MB, while the backing map as whole has a capacity limit of 8GB (8192*1048576). Again, you must configure a backup storage for this cache being `off-heap` or `file-mapped`.

Using the Elastic Data Feature to Store Data

The Elastic Data feature is used to seamlessly store data across memory and disk-based devices. This feature is especially tuned to take advantage of fast disk-based devices such as Solid State Disks (SSD) and enables near memory speed while storing and reading data from SSDs. The Elastic Data feature uses a technique called journaling to optimize the storage across memory and disk.

Elastic data contains two distinct components: the RAM journal for storing data in-memory and the flash journal for storing data to disk-based devices. These can be combined in different combinations and are typically used for backing maps and backup storage but can also be used with composite caches (for example, a near cache). The RAM journal always works with the flash journal to enable seamless overflow to disk.

Caches that use RAM and flash journals are configured as part of a cache scheme definition within a cache configuration file. Journaling behavior is configured, as required, by using an operational override file to override the out-of-box configuration.

The following topics are included in this section:

Journaling Overview

Journaling refers to the technique of recording state changes in a sequence of modifications called a journal. As changes occur, the journal records each value for a specific key and a tree structure that is stored in memory keeps track of which journal entry contains the current value for a particular key. To find the value for an entry, you find the key in the tree which includes a pointer to the journal entry that contains the latest value.

As changes in the journal become obsolete due to new values being written for a key, stale values accumulate in the journal. At regular intervals, the stale values are evacuated making room for new values to be written in the journal.

The Elastic Data feature includes a RAM journal implementation and a Flash journal implementation that work seamlessly with each other. If for example the RAM Journal runs out of memory, the Flash Journal automatically accepts the overflow from the RAM Journal, allowing for caches to expand far beyond the size of RAM.

Note: When journaling is enabled, additional capacity planning is required if you are performing data grid operations (such as queries and aggregations) on large result sets. See *Oracle Coherence Administrator's Guide* for details.

A resource manager controls journaling. The resource manager creates and utilizes a binary store to perform operations on the journal. The binary store is implemented by the `JournalBinaryStore` class. All reads and writes through the binary store are handled by the resource manager. There is a resource manager for RAM journals (`RamJournalRM`) and one for flash journals (`FlashJournalRM`). Lastly, journaling uses the `SimpleSerializationMap` class as the backing map implementation. Custom implementation of `SimpleSerializationMap` can be created as required. See *Oracle Coherence Java API Reference* for specific details on these APIs.

Defining Journal Schemes

The `<ramjournal-scheme>` and `<flashjournal-scheme>` elements are used to configure RAM and Flash journals (respectively) in a cache configuration file. See the "ramjournal-scheme" on page B-87 and the "flashjournal-scheme" on page B-41 for detailed configuration options for these scheme types.

The following topics are included in this section:

- Configuring a RAM Journal Backing Map
- Configuring a Flash Journal Backing Map
- Referencing a Journal Scheme
- Using a Journal Scheme for Backup Storage
- Enabling a Custom Map Implementation for a Journal Scheme

Configuring a RAM Journal Backing Map

To configure a RAM journal backing map, add the `<ramjournal-scheme>` element within the `<backing-map-scheme>` element of a cache definition. The following example creates a distributed cache that uses a RAM journal for the backing map. The RAM journal automatically delegates to a flash journal when the RAM journal exceeds the configured memory size. See "Changing Journaling Behavior" on page 13-10 to change memory settings.

```
<distributed-scheme>
   <scheme-name>distributed-journal</scheme-name>
   <service-name>DistributedCacheRAMJournal</service-name>
   <backing-map-scheme>
      <ramjournal-scheme/>
   </backing-map-scheme>
   <autostart>true</autostart>
</distributed-scheme>
```

Configuring a Flash Journal Backing Map

To configure a flash journal backing map, add the `<flashjournal-scheme>` element within the `<backing-map-scheme>` element of a cache definition. The following example creates a distributed scheme that uses a flash journal for the backing map.

```
<distributed-scheme>
   <scheme-name>distributed-journal</scheme-name>
   <service-name>DistributedCacheFlashJournal</service-name>
   <backing-map-scheme>
      <flashjournal-scheme/>
   </backing-map-scheme>
   <autostart>true</autostart>
</distributed-scheme>
```

Referencing a Journal Scheme

The RAM and flash journal schemes both support the use of scheme references to reuse scheme definitions. The following example creates a distributed cache and configures a RAM journal backing map by referencing the RAM scheme definition called `default-ram`.

```
<caching-schemes>
   <distributed-scheme>
      <scheme-name>distributed-journal</scheme-name>
         <service-name>DistributedCacheJournal</service-name>
         <backing-map-scheme>
            <ramjournal-scheme>
               <scheme-ref>default-ram</scheme-ref>
            </ramjournal-scheme>
         </backing-map-scheme>
         <autostart>true</autostart>
   </distributed-scheme>

   <ramjournal-scheme>
      <scheme-name>default-ram</scheme-name>
   </ramjournal-scheme>
</caching-schemes>
```

Using a Journal Scheme for Backup Storage

Journal schemes are used for backup storage as well as for backing maps. By default, a distributed scheme that is configured to use a RAM journal as a backing map also uses a RAM journal for backup storage. Similarly, a distributed scheme that uses a flash journal for a backing map also uses a flash journal for backup storage. This default behavior can be modified by explicitly specifying the storage type within the `<backup-storage>` element. The following configuration uses a RAM journal for the backing map and explicitly configures a flash journal for backup storage:

```
<caching-schemes>
   <distributed-scheme>
      <scheme-name>default-distributed-journal</scheme-name>
         <service-name>DistributedCacheJournal</service-name>
         <backup-storage>
            <type>scheme</type>
            <scheme-name>example-flash</scheme-name>
         </backup-storage>
         <backing-map-scheme>
            <ramjournal-scheme/>
         </backing-map-scheme>
      <autostart>true</autostart>
   </distributed-scheme>

   <flashjournal-scheme>
      <scheme-name>example-flash</scheme-name>
```

```
   </flashjournal-scheme>
</caching-schemes>
```

Enabling a Custom Map Implementation for a Journal Scheme

Journal schemes can be configured to use a custom map as required. Custom map implementations must extend the `SimpleSerializationMap` class and declare the exact same set of public constructors. To enable, a custom implementation, add a `<class-scheme>` element whose value is the fully qualified name of the custom class. Any parameters that are required by the custom class can be defined using the `<init-params>` element. The following example enables a custom map implementation called `MySimpleSerializationMap`.

```
<flashjournal-scheme>
   <scheme-name>example-flash</scheme-name>
   <class-name>package.MySimpleSerializationMap</class-name>
</flashjournal-scheme>
```

Changing Journaling Behavior

A resource manager controls journaling behavior. There is a resource manager for RAM journals (`RamJournalRM`) and a resource manager for Flash journals (`FlashJournalRM`). The resource managers are configured for a cluster in the `tangosol-coherence-override.xml` operational override file. The resource managers' default out-of-box settings are used if no configuration overrides are set.

The following topics are included in this section:

- Configuring the RAM Journal Resource Manager
- Configuring the Flash Journal Resource Manager

Configuring the RAM Journal Resource Manager

The `<ramjournal-manager>` element is used to configure RAM journal behavior. The following lists provides a brief summary of the defaults that are set by the resource manager. See "ramjournal-manager" on page A-53 for details on all settings that are available and their defaults.

- Binary values are limited by default to 64KB (and maximum 4MB)
- An individual buffer (a journal file) is limited by default to 2MB (and maximum 2GB)
- A journal is composed of up to 512 files
- The total memory used by the journal is limited to 1GB by default (and maximum 64GB)

> **Note:** A flash journal is automatically used if the binary value setting, or memory setting, or both are exceeded.

To configure a RAM journal resource manager, add a `<ramjournal-manager>` element within a `<journaling-config>` element and define any subelements that are to be overridden. The following example demonstrates overriding each of the available subelements:

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

```
<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <journaling-config>
         <ramjournal-manager>
            <maximum-value-size>64K</maximum-value-size>
            <maximum-size>2G</maximum-size>
         </ramjournal-manager>
      </journaling-config>
   </cluster-config>
</coherence>
```

Configuring the Flash Journal Resource Manager

The `<flashjournal-manager>` element is used to configure flash journal behavior. The following lists provides a brief summary of the defaults that are set by the resource manager. See "flashjournal-manager" on page A-15 for details on all settings that are available and their defaults.

- Binary values are limited by default to 64MB
- An individual buffer (a journal file) is limited by default to 2GB (and maximum 4GB)
- A journal is composed of up to 512 files.
- A journal is thus limited by default to 1TB, with a theoretical maximum of 2TB.

To configure a flash journal resource manager, add a `<flashjournal-manager>` element within a `<journaling-config>` element and define any subelements that are to be overridden. The following example demonstrates overriding each of the available subelements:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <journaling-config>
         <flashjournal-manager>
            <maximum-value-size>64K</maximum-value-size>
            <maximum-file-size>8M</maximum-file-size>
            <block-size>512K</block-size>
            <maximum-pool-size>32M</maximum-pool-size>
            <directory>/coherence_storage</directory>
            <async-limit>32M</async-limit>
         </flashjournal-manager>
      </journaling-config>
   </cluster-config>
</coherence>
```

Note: The directory specified for storing journal files must exist. If the directory does not exist, a warning is logged and the default temporary file directory, as designated by the JVM, is used.

Using Delta Backup

Delta backup is a technique that is used to apply changes to a backup binary entry rather than replacing the whole entry when the primary entry changes. Delta backup is ideal in situations where the entry being updated is large but only small changes are being made. In such cases, the cost for changing only a small portion of the entry is often less than the cost associated with rewriting the whole entry and results in better performance. Entries that change by more than 50% typically demonstrate little or no performance gain by using delta backup.

Delta backup uses a compressor that compares two in-memory buffers containing an old and a new value and produces a result (called a delta) that can be applied to the old value to create the new value. Coherence provides standard delta compressors for POF and non-POF formats. Custom compressors can also be created and configured as required.

Enabling Delta Backup

Delta backup is only available for distributed caches and is disabled by default. Delta backup is enabled either individually for each distributed cache or for all instances of the distributed cache service type.

To enable delta backup for a distributed cache, add a `<compressor>` element, within a `<distributed-scheme>` element, that is set to `standard`. For example:

```
<distributed-scheme>
   ...
   <compressor>standard</compressor>
   ...
</distributed-scheme>
```

To enable delta backup for all instances of the distributed cache service type, override the partitioned cache service's `compressor` initialization parameter in an operational override file. For example:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <services>
         <service id="3">
            <init-params>
               <init-param id="22">
                  <param-name>compressor</param-name>
                  <param-value
                     system-property="tangosol.coherence.distributed.compressor">
                     standard</param-value>
               </init-param>
            </init-params>
         </service>
      </services>
   </cluster-config>
</coherence>
```

The `tangosol.coherence.distributed.compressor` system property is used to enable delta backup for all instances of the distributed cache service type instead of using the operational override file. For example:

```
-Dtangosol.coherence.distributed.compressor=standard
```

Enabling a Custom Delta Backup Compressor

To use a custom compressor for performing delta backup, include an `<instance>` subelement and provide a fully qualified class name that implements the `DeltaCompressor` interface. See "instance" on page B-49 for detailed instructions on using the `<instance>` element. The following example enables a custom compressor that is implemented in the `MyDeltaCompressor` class.

```
<distributed-scheme>
   ...
   <compressor>
      <instance>
         <class-name>package.MyDeltaCompressor</class-name>
      </instance>
   </compressor>
   ...
</distributed-scheme>
```

As an alternative, the `<instance>` element supports the use of a `<class-factory-name>` element to use a factory class that is responsible for creating `DeltaCompressor` instances, and a `<method-name>` element to specify the static factory method on the factory class that performs object instantiation. The following example gets a custom compressor instance using the `getCompressor` method on the `MyCompressorFactory` class.

```
<distributed-scheme>
   ...
   <compressor>
      <instance>
         <class-factory-name>package.MyCompressorFactory</class-factory-name>
         <method-name>getCompressor</method-name>
      </instance>
   </compressor>
   ...
</distributed-scheme>
```

Any initialization parameters that are required for an implementation can be specified using the `<init-params>` element. The following example sets the `iMaxTime` parameter to `2000`.

```
<distributed-scheme>
   ...
   <compressor>
      <instance>
         <class-name>package.MyDeltaCompressor</class-name>
         <init-params>
            <init-param>
               <param-name>iMaxTime</param-name>
               <param-value>2000</param-value>
            </init-param>
         </init-params>
      </instance>
   </compressor>
   ...
</distributed-scheme>
```

14

Caching Data Sources

This chapter provides instructions for caching data sources to use Coherence as a temporary system-of-record. The chapter includes samples and implementation considerations.

The following sections are included in this chapter:

Overview of Caching Data Sources

Coherence supports transparent read/write caching of any data source, including databases, web services, packaged applications and file systems; however, databases are the most common use case. As shorthand, "database" is used to describe any back-end data source. Effective caches must support both intensive read-only **and** read/write operations, and for read/write operations, the cache and database must be kept fully synchronized. To accomplish caching of data sources, Coherence supports **Read-Through, Write-Through, Refresh-Ahead and Write-Behind** caching.

> **Note:** Read-through/write-through caching (and variants) are intended for use only with the Partitioned (Distributed) cache topology (and by extension, Near cache). Local caches support a subset of this functionality. Replicated and Optimistic caches should not be used.

The following topics are include in this section:

Pluggable Cache Store

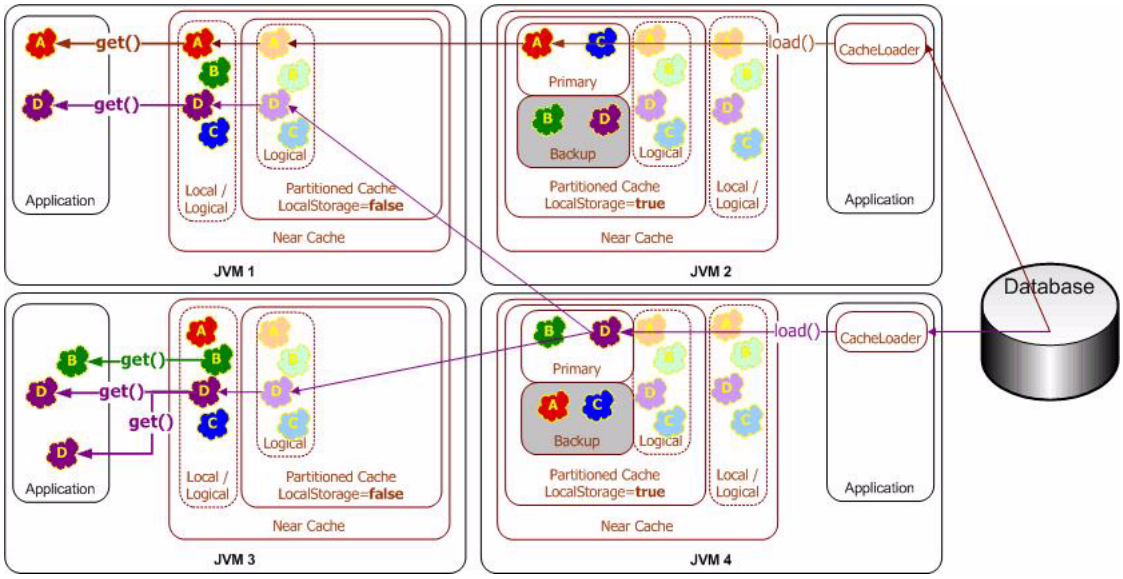
A `CacheStore` is an application-specific adapter used to connect a cache to a underlying data source. The `CacheStore` implementation accesses the data source by using a data access mechanism (for example, *Hibernate*, *Toplink* Essentials, *JPA*, application-specific JDBC calls, another application, mainframe, another cache, and so on). The `CacheStore` understands how to build a Java object using data retrieved from the data source, map and write an object to the data source, and erase an object from the data source.

Both the data source connection strategy and the data source-to-application-object mapping information are specific to the data source schema, application class layout, and operating environment. Therefore, this mapping information must be provided by the application developer in the form of a `CacheStore` implementation. See "Creating a CacheStore Implementation" for more information.

Read-Through Caching

When an application asks the cache for an entry, for example the `key X`, and `X` is not in the cache, Coherence automatically delegates to the CacheStore and ask it to load `X` from the underlying data source. If `X` exists in the data source, the `CacheStore` loads it, returns it to Coherence, then Coherence places it in the cache for future use and finally returns `X` to the application code that requested it. This is called **Read-Through** caching. Refresh-Ahead Cache functionality may further improve read performance (by reducing perceived latency). See "Refresh-Ahead Caching" for more information.

Figure 14–1 Read-Through Caching

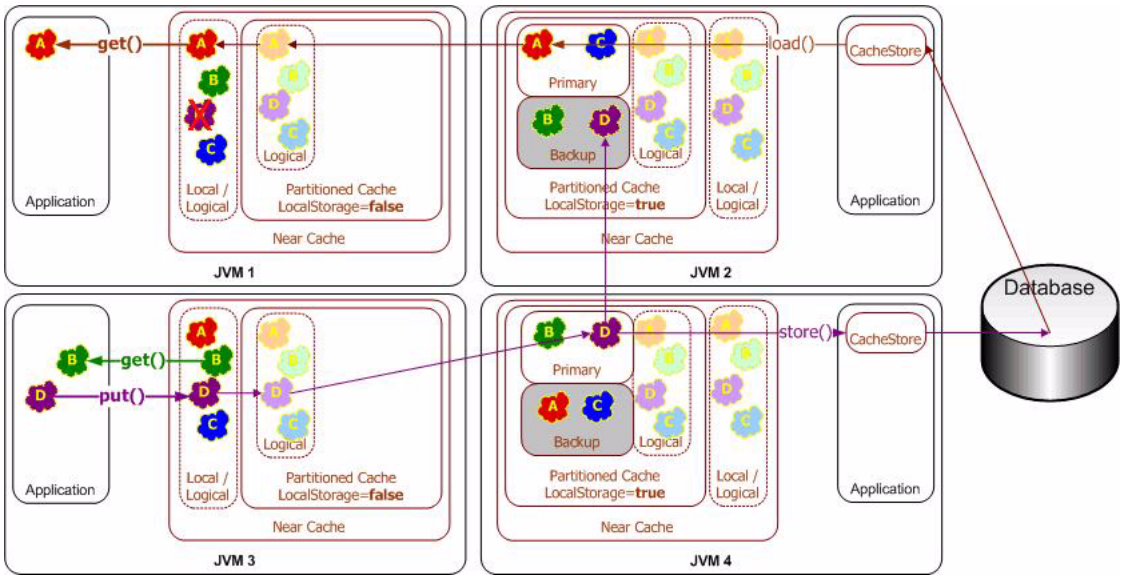


Write-Through Caching

Coherence can handle updates to the data source in two distinct ways, the first being **Write-Through**. In this case, when the application updates a piece of data in the cache (that is, calls `put`(...) to change a cache entry,) the operation does not complete (that is, the `put` does not return) until Coherence has gone through the `CacheStore` and successfully stored the data to the underlying data source. This does not improve write performance at all, since you are still dealing with the latency of the write to the

data source. Improving the write performance is the purpose for the *Write-Behind Cache* functionality. See "Write-Behind Caching" for more information.

Figure 14–2 Write-Through Caching



Write-Behind Caching

In the **Write-Behind** scenario, modified cache entries are asynchronously written to the data source after a configured delay, whether after 10 seconds, 20 minutes, a day, a week or even longer. Note that this only applies to cache inserts and updates - cache entries are removed synchronously from the data source. For **Write-Behind** caching, Coherence maintains a write-behind queue of the data that must be updated in the data source. When the application updates `X` in the cache, `X` is added to the write-behind queue (if it is not there; otherwise, it is replaced), and after the specified write-behind delay Coherence calls the CacheStore to update the underlying data source with the latest state of `X`. Note that the write-behind delay is relative to the first of a series of modifications—in other words, the data in the data source never lags behind the cache by more than the write-behind delay.

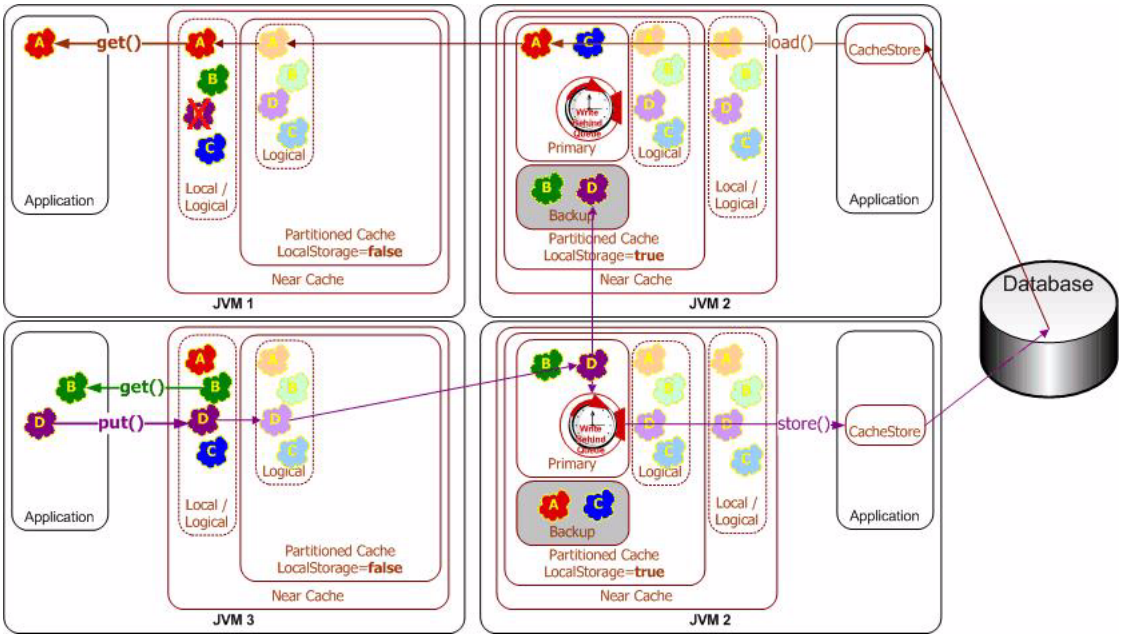
The result is a "read-once and write at a configured interval" (that is, much less often) scenario. There are four main benefits to this type of architecture:

- The application improves in performance, because the user does not have to wait for data to be written to the underlying data source. (The data is written later, and by a different execution thread.)
- The application experiences drastically reduced database load: Since the amount of both read and write operations is reduced, so is the database load. The reads are reduced by caching, as with any other caching approach. The writes, which are typically much more expensive operations, are often reduced because multiple changes to the same object within the write-behind interval are "coalesced" and only written once to the underlying data source ("write-coalescing"). Additionally, writes to multiple cache entries may be combined into a single database transaction ("write-combining") by using the `CacheStore.storeAll()` method.
- The application is somewhat insulated from database failures: the **Write-Behind** feature can be configured in such a way that a write failure results in the object being re-queued for write. If the data that the application is using is in the

Coherence cache, the application can continue operation without the database being up. This is easily attainable when using the Coherence Partitioned Cache, which partitions the entire cache across all participating cluster nodes (with local-storage enabled), thus allowing for enormous caches.

- Linear Scalability: For an application to handle more concurrent users you need only increase the number of nodes in the cluster; the effect on the database in terms of load can be tuned by increasing the write-behind interval.

Figure 14–3 Write-Behind Caching



Write-Behind Requirements

While enabling write-behind caching is simply a matter of adjusting one configuration setting, ensuring that write-behind works as expected is more involved. Specifically, application design must address several design issues up-front.

The most direct implication of write-behind caching is that database updates occur outside of the cache transaction; that is, the cache transaction usually completes before the database transaction(s) begin. This implies that the database transactions must never fail; if this cannot be guaranteed, then rollbacks must be accommodated.

As write-behind may re-order database updates, referential integrity constraints must allow out-of-order updates. Conceptually, this is similar to using the database as ISAM-style storage (primary-key based access with a guarantee of no conflicting updates). If other applications share the database, this introduces a new challenge—there is no way to guarantee that a write-behind transaction does not conflict with an external update. This implies that write-behind conflicts must be handled heuristically or escalated for manual adjustment by a human operator.

As a rule of thumb, mapping each cache entry update to a logical database transaction is ideal, as this guarantees the simplest database transactions.

Because write-behind effectively makes the cache the system-of-record (until the write-behind queue has been written to disk), business regulations must allow cluster-durable (rather than disk-durable) storage of data and transactions.

Refresh-Ahead Caching

In the **Refresh-Ahead** scenario, Coherence allows a developer to configure a cache to automatically and asynchronously reload (refresh) any recently accessed cache entry from the cache loader before its expiration. The result is that after a frequently accessed entry has entered the cache, the application does not feel the impact of a read against a potentially slow cache store when the entry is reloaded due to expiration. The asynchronous refresh is only triggered when an object that is sufficiently close to its expiration time is accessed—if the object is accessed after its expiration time, Coherence performs a synchronous read from the cache store to refresh its value.

The refresh-ahead time is expressed as a percentage of the entry's expiration time. For example, assume that the expiration time for entries in the cache is set to 60 seconds and the refresh-ahead factor is set to 0.5. If the cached object is accessed after 60 seconds, Coherence performs a *synchronous* read from the cache store to refresh its value. However, if a request is performed for an entry that is more than 30 but less than 60 seconds old, the current value in the cache is returned and Coherence schedules an *asynchronous* reload from the cache store.

Refresh-ahead is especially useful if objects are being accessed by a large number of users. Values remain fresh in the cache and the latency that could result from excessive reloads from the cache store is avoided.

The value of the refresh-ahead factor is specified by the `<refresh-ahead-factor>` subelement of the `<read-write-backing-map-scheme>` element in the `coherence-cache-config.xml` file. Refresh-ahead assumes that you have also set an expiration time (`<expiry-delay>`) for entries in the cache.

Example 14–1 configures a refresh-ahead factor of 0.5 and an expiration time of 20 seconds for entries in the local cache. If an entry is accessed within 10 seconds of its expiration time, it is scheduled for an asynchronous reload from the cache store.

Example 14–1 Specifying a Refresh-Ahead Factor

```
<distributed-scheme>
   <scheme-name>categories-cache-all-scheme</scheme-name>
   <service-name>DistributedCache</service-name>
   <backing-map-scheme>

      <read-write-backing-map-scheme>
         <scheme-name>categoriesLoaderScheme</scheme-name>
         <internal-cache-scheme>
            <local-scheme>
               <scheme-ref>categories-eviction</scheme-ref>
            </local-scheme>
         </internal-cache-scheme>

         <cachestore-scheme>
            <class-scheme>
               <class-name>
                  com.demo.cache.coherence.categories.CategoryCacheLoader
               </class-name>
            </class-scheme>
         </cachestore-scheme>
         <refresh-ahead-factor>0.5</refresh-ahead-factor>
      </read-write-backing-map-scheme>
   </backing-map-scheme>
   <autostart>true</autostart>
</distributed-scheme>
<local-scheme>
```

```
    <scheme-name>categories-eviction</scheme-name>
    <expiry-delay>20s</expiry-delay>
</local-scheme>
```

Selecting a Cache Strategy

This section compares and contrasts the benefits of several caching strategies.

- Read-Through/Write-Through versus Cache-Aside
- Refresh-Ahead versus Read-Through
- Write-Behind versus Write-Through

Read-Through/Write-Through versus Cache-Aside

There are two common approaches to the cache-aside pattern in a clustered environment. One involves checking for a cache miss, then querying the database, populating the cache, and continuing application processing. This can result in multiple database visits if different application threads perform this processing at the same time. Alternatively, applications may perform double-checked locking (which works since the check is atomic for the cache entry). This, however, results in a substantial amount of overhead on a cache miss or a database update (a clustered lock, additional read, and clustered unlock - up to 10 additional network hops, or 6-8ms on a typical gigabit Ethernet connection, plus additional processing overhead and an increase in the "lock duration" for a cache entry).

By using inline caching, the entry is locked only for the 2 network hops (while the data is copied to the backup server for fault-tolerance). Additionally, the locks are maintained locally on the partition owner. Furthermore, application code is fully managed on the cache server, meaning that only a controlled subset of nodes directly accesses the database (resulting in more predictable load and security). Additionally, this decouples cache clients from database logic.

Refresh-Ahead versus Read-Through

Refresh-ahead offers reduced latency compared to read-through, but only if the cache can accurately predict which cache items are likely to be needed in the future. With full accuracy in these predictions, refresh-ahead offers reduced latency and no added overhead. The higher the rate of inaccurate prediction, the greater the impact is on throughput (as more unnecessary requests are sent to the database) - potentially even having a negative impact on latency should the database start to fall behind on request processing.

Write-Behind versus Write-Through

If the requirements for write-behind caching can be satisfied, write-behind caching may deliver considerably higher throughput and reduced latency compared to write-through caching. Additionally write-behind caching lowers the load on the database (fewer writes), and on the cache server (reduced cache value deserialization).

Creating a CacheStore Implementation

`CacheStore` implementations are pluggable and depending on the cache's usage of the data source must implement one of two interfaces:

- `CacheLoader` for read-only caches
- `CacheStore` which extends `CacheLoader` to support read/write caches

These interfaces are located in the `com.tangosol.net.cache` package. The `CacheLoader` interface has two main methods: `load(Object key)` and `loadAll(Collection keys)`, and the `CacheStore` interface adds the methods `store(Object key, Object value)`, `storeAll(Map mapEntries)`, `erase(Object key)`, and `eraseAll(Collection colKeys)`.

See "Sample CacheStore" on page 14-8 and "Sample Controllable CacheStore" on page 14-13 for example `CacheStore` implementations.

Plugging in a CacheStore Implementation

To plug in a `CacheStore` module, specify the `CacheStore` implementation class name within the `distributed-scheme`, `backing-map-scheme`, `cachestore-scheme`, or `read-write-backing-map-scheme`, cache configuration element.

The `read-write-backing-map-scheme` configures a `com.tangosol.net.cache.ReadWriteBackingMap`. This backing map is composed of two key elements: an internal map that actually caches the data (see `internal-cache-scheme`), and a `CacheStore` module that interacts with the database (see `cachestore-scheme`).

Example 14–2 illustrates a cache configuration that specifies a `CacheStore` module. The `<init-params>` element contains an ordered list of parameters that is passed into the `CacheStore` constructor. The `{cache-name}` configuration macro is used to pass the cache name into the `CacheStore` implementation, allowing it to be mapped to a database table. For a complete list of available macros, see "Using Parameter Macros" on page 12-12.

For more detailed information on configuring write-behind and refresh-ahead, see the `read-write-backing-map-scheme`, taking note of the `write-batch-factor`, `refresh-ahead-factor`, `write-requeue-threshold`, and `rollback-cachestore-failures` elements.

Example 14–2 Example Cachestore Module

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <caching-scheme-mapping>
      <cache-mapping>
         <cache-name>com.company.dto.*</cache-name>
         <scheme-name>distributed-rwbm</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <distributed-scheme>
         <scheme-name>distributed-rwbm</scheme-name>
         <backing-map-scheme>
            <read-write-backing-map-scheme>

            <internal-cache-scheme>
```

```
            <local-scheme/>
         </internal-cache-scheme>

         <cachestore-scheme>
            <class-scheme>
               <class-name>com.company.MyCacheStore</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>
      </backing-map-scheme>
   </distributed-scheme>
</caching-schemes>
</cache-config>
```

Note: Thread Count: The use of a `CacheStore` module substantially increases the consumption of cache service threads (even the fastest database select is orders of magnitude slower than updating an in-memory structure). Consequently, the cache service thread count must be increased (typically in the range 10-100). The most noticeable symptom of an insufficient thread pool is increased latency for cache requests (without corresponding behavior in the backing database).

Sample CacheStore

This section provides a very basic implementation of the `com.tangosol.net.cache.CacheStore` interface. The implementation in Example 14–3 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 17-4 provides an example of a database cache configuration.

Tip: Save processing effort by bulk loading the cache. The following example use the `put` method to write values to the cache store. Often, performing bulk loads with the `putAll` method results in a savings in processing effort and network traffic. For more information on bulk loading, see Chapter 20, "Pre-Loading a Cache."

Example 14–3 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 does 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 it writes updated values 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 requirement 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 14–4 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 require storage. 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 indicate that an object added to the cache should not be stored in the database.

See "Cache of a Database" on page 17-4 for a sample cache configurations.

Example 14–4 provides a listing of the `Main.java` interface.

Example 14–4 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
    {

    /**
     * A cache controlled CacheStore implementation
     */
    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;
                }

            }

        /**
         * a valued controlled CacheStore implementation that
         * implements 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 14–5 provides a listing of the `CacheStoreAware.java` interface.

Example 14–5 CacheStoreAware.java interface

```
public interface CacheStoreAware
    {
    public boolean isSkipStore();
    }
```

Implementation Considerations

Please keep the following in mind when implementing a `CacheStore`.

Idempotency

All `CacheStore` operations should be designed to be idempotent (that is, repeatable without unwanted side-effects). For write-through and write-behind caches, this allows Coherence to provide low-cost fault-tolerance for partial updates by re-trying the database portion of a cache update during failover processing. For write-behind caching, idempotency also allows Coherence to combine multiple cache updates into a single `CacheStore` invocation without affecting data integrity.

Applications that have a requirement for write-behind caching but which must avoid write-combining (for example, for auditing reasons), should create a "versioned" cache key (for example, by combining the natural primary key with a sequence id).

Write-Through Limitations

Coherence does not support two-phase `CacheStore` operations across multiple `CacheStore` instances. In other words, if two cache entries are updated, triggering calls to `CacheStore` modules sitting on separate cache servers, it is possible for one database update to succeed and for the other to fail. In this case, it may be preferable to use a cache-aside architecture (updating the cache and database as two separate components of a single transaction) with the application server transaction manager. In many cases it is possible to design the database schema to prevent logical commit failures (but obviously not server failures). Write-behind caching avoids this issue as "puts" are not affected by database behavior (as the underlying issues have been addressed earlier in the design process).

Cache Queries

Cache queries only operate on data stored in the cache and do not trigger the `CacheStore` to load any missing (or potentially missing) data. Therefore, applications that query `CacheStore`-backed caches should ensure that all necessary data required for the queries has been pre-loaded. For efficiency, most bulk load operations should be done at application startup by streaming the data set directly from the database into the cache (batching blocks of data into the cache by using `NamedCache.putAll()`. The loader process must use a "Controllable Cachestore" pattern to disable circular updates back to the database. The `CacheStore` may be controlled by using an Invocation service (sending agents across the cluster to modify a local flag in each JVM) or by setting the value in a Replicated cache (a different cache service) and reading it in every `CacheStore` method invocation (minimal overhead compared to the typical database operation). A custom MBean can also be used, a simple task with Coherence's clustered JMX facilities.

Re-entrant Calls

The `CacheStore` implementation must not call back into the hosting cache service. This includes ORM solutions that may internally reference Coherence cache services. Note that calling into another cache service instance is allowed, though care should be taken to avoid deeply nested calls (as each call "consumes" a cache service thread and could result in deadlock if a cache service thread pool is exhausted).

Cache Server Classpath

The classes for cache entries (also known as Value Objects, Data Transfer Objects, and so on) must be in the cache server classpath (as the cache server must serialize-deserialize cache entries to interact with the `CacheStore` module.

CacheStore Collection Operations

The `CacheStore.storeAll` method is most likely to be used if the cache is configured as write-behind and the `<write-batch-factor>` is configured. The `CacheLoader.loadAll` method is also used by Coherence. For similar reasons, its first use likely requires refresh-ahead to be enabled.

Connection Pools

Database connections should be retrieved from the container connection pool (or a third party connection pool) or by using a thread-local lazy-initialization pattern. As dedicated cache servers are often deployed without a managing container, the latter

may be the most attractive option (though the cache service thread-pool size should be constrained to avoid excessive simultaneous database connections).

15

Serialization Paged Cache

This chapter provides general information about caching large amounts of binary data off-heap.

The following sections are included in this chapter:

Understanding Serialization Paged Cache

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 are rarely accessed, or whose accesses tolerates a higher latency if the cached data has been paged to disk. See Chapter 13, "Implementing Storage and Backing Maps."

Serialization Paged Cache is defined as follows:

- *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.
- *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 manages 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 can efficiently expire 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 manages, and the length of time a page is active. 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 Berkley DB (referred to as "BDB") and the various NIO implementations.

Serialization paged caches are configured within the `<external-scheme>` and `<paged-external-scheme>` element in the cache configuration file. See "external-scheme" on page B-37 and "paged-external-scheme" on page B-76 for details.

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. See the <external-scheme> and <paged-external-scheme> cache configuration elements.

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 value is one) for a partitioned cache service, and using the Serialization Paged Cache as the primary backing storage for the service, it is a best practice to also use the Serialization Paged Cache as the backup store, and configure the backup with the passive option. See the <paged-external-scheme> cache configuration elements.

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. A high partition count breaks 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, twenty thousand partitions breaks the cache up into units averaging about 50MB. If a unit (the size of a partition) is too large, it causes an out-of-memory condition when load-balancing the cache. (Remember to ensure 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. See the <external-scheme> and <paged-external-scheme> cache configuration elements.

When the cache is used for large amounts of data, the pages are typically disk-backed. Since the cache eventually expires each page, thus releasing the disk resources, the cache uses a virtual erase optimization by default. 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 is 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) tends 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.)

16

Using Quorum

The following sections are included in this chapter:

Overview

A quorum, in Coherence, refers to the minimum number of service members that are required in a cluster before a service action is allowed or disallowed. Quorums are beneficial because they automatically provide assurances that a cluster behaves in an expected way when member thresholds are reached. For example, a partitioned cache backup quorum might require at least 5 storage-enabled members before the partitioned cache service is allowed to back up partitions.

Quorums are service-specific and defined within a quorum policy; there is a cluster quorum policy for the Cluster service, a partitioned quorum policy for the Partitioned Cache service, and a proxy quorum policy for the Proxy service. Quorum thresholds are set on the policy using a cache configuration file.

Each quorum provides benefits for its particular service. However, in general, quorums:

- control service behavior at different service member levels
- mandate the minimum service member levels that are required for service operations
- ensure an optimal cluster and cache environment for a particular application or solution

Using the Cluster Quorum

The cluster quorum policy defines a single quorum (the timeout survivor quorum) for the Cluster Service. The timeout survivor quorum mandates the minimum number of cluster members that must remain in the cluster when the cluster service is terminating suspect members. A member is considered suspect if it has not responded to network communications and is in imminent danger of being disconnected from the cluster. The quorum can be specified generically across all members or constrained to members that have a specific role in the cluster, such as client or server members. See

the <role-name> element in "member-identity" on page A-36 for more information on defining role names for cluster members.

This quorum is typically used in environments where network performance varies. For example, intermittent network outages may cause a high number of cluster members to be removed from the cluster. Using this quorum, a certain number of members are maintained during the outage and are available when the network recovers. This behavior also minimizes the manual intervention required to restart members. Naturally, requests that require cooperation by the nodes that are not responding are not able to complete and are either blocked for the duration of the outage or are timed out.

Configuring the Cluster Quorum Policy

The timeout survivor quorum threshold is configured in an operational override file using the <timeout-survivor-quorum> element and optionally the role attribute. This element must be used within a <cluster-quorum-policy> element. The following example demonstrates configuring the timeout survivor quorum threshold to ensure that5 cluster members with the server role are always kept in the cluster while removing suspect members:

```
<cluster-config>
   <member-identity>
      <role-name>server</role-name>
   </member-identity>
   <cluster-quorum-policy>
      <timeout-survivor-quorum role="Server">5</timeout-survivor-quorum>
   </cluster-quorum-policy>
</cluster-config>
```

Using the Partitioned Cache Quorums

The partitioned cache quorum policy defines four quorums for the partitioned cache service (DistributedCache) that mandate how many service members are required before different partitioned cache service operations can be performed:

- **Distribution Quorum** – This quorum mandates the minimum number of storage-enabled members of a partitioned cache service that must be present before the partitioned cache service is allowed to perform partition distribution.
- **Restore Quorum** – This quorum mandates the minimum number of storage-enabled members of a partitioned cache service that must be present before the partitioned cache service is allowed to restore lost primary partitions from backup.
- **Read Quorum** – This quorum mandates the minimum number of storage-enabled members of a partitioned cache service that must be present to process read requests. A read request is any request that does not mutate the state or contents of a cache.
- **Write Quorum** – This quorum mandates the minimum number of storage-enabled members of a partitioned cache service that must be present to process write requests. A write request is any request that may mutate the state or contents of a cache.

These quorums are typically used to indicate at what service member levels different service operations are best performed given the intended use and requirements of a distributed cache. For example, a small distributed cache may only require three

storage-enabled members to adequately store data and handle projected request volumes. While; a large distributed cache may require 10, or more, storage-enabled members to adequately store data and handle projected request volumes. Optimal member levels are tested during development and then set accordingly to ensure that the minimum service member levels are provisioned in a production environment.

If the number of storage-enabled nodes running the service drops below the configured level of read or write quorum, the corresponding client operation are rejected by throwing the `com.tangosol.net.RequestPolicyException`. If the number of storage-enabled nodes drops below the configured level of distribution quorum, some data may become "endangered" (no backup) until the quorum is reached. Dropping below the restore quorum may cause some operation to be blocked until the quorum is reached or to be timed out.

Configuring the Partitioned Cache Quorum Policy

Partitioned cache quorums are configured in a cache configuration file within the `<partitioned-quorum-policy-scheme>` element. The element must be used within a `<distributed-scheme>` element. The following example demonstrates configuring thresholds for the partitioned cache quorums. Ideally, the threshold values would indicate the minimum amount of service members that are required to perform the operation.

```
<distributed-scheme>
   <scheme-name>partitioned-cache-with-quorum</scheme-name>
   <service-name>PartitionedCacheWithQuorum</service-name>
   <backing-map-scheme>
      <local-scheme/>
   </backing-map-scheme>
   <partitioned-quorum-policy-scheme>
      <distribution-quorum>4</distribution-quorum>
      <restore-quorum>3</restore-quorum>
      <read-quorum>3</read-quorum>
      <write-quorum>5</write-quorum>
   </partitioned-quorum-policy-scheme>
   <autostart>true</autostart>
</distributed-scheme>
```

The `<partitioned-quorum-policy-scheme>` element also supports the use of scheme references. In the below example, a `<partitioned-quorum-policy-scheme>`, with the name `partitioned-cache-quorum`, is referenced from within the `<distributed-scheme>` element:

```
<distributed-scheme>
   <scheme-name>partitioned-cache-with-quorum</scheme-name>
   <service-name>PartitionedCacheWithQuorum</service-name>
   <backing-map-scheme>
      <local-scheme/>
   </backing-map-scheme>
   <partitioned-quorum-policy-scheme>
      <scheme-ref>partitioned-cache-quorum</scheme-ref>
   </partitioned-quorum-policy-scheme>
   <autostart>true</autostart>
</distributed-scheme>

<distributed-scheme>
   <scheme-name>dist-example</scheme-name>
   <service-name>DistributedCache</service-name>
```

```
   <backing-map-scheme>
      <local-scheme/>
   </backing-map-scheme>
   <partitioned-quorum-policy-scheme>
      <scheme-name>partitioned-cache-quorum</scheme-name>
      <distribution-quorum>4</distribution-quorum>
      <restore-quorum>3</restore-quorum>
      <read-quorum>3</read-quorum>
      <write-quorum>5</write-quorum>
   </partitioned-quorum-policy-scheme>
   <autostart>true</autostart>
</distributed-scheme>
```

Using the Proxy Quorum

The proxy quorum policy defines a single quorum (the connection quorum) for the proxy service. The connection quorum mandates the minimum number of proxy service members that must be available before the proxy service can allow client connections.

This quorum is typically used to ensure enough proxy service members are available to optimally support a given set of TCP clients. For example, a small number of clients may efficiently connect to a cluster using two proxy services. While; a large number of clients may require 3 or more proxy services to efficiently connect to a cluster. Optimal levels are tested during development and then set accordingly to ensure that the minimum service member levels are provisioned in a production environment.

Configuring the Proxy Quorum Policy

The connection quorum threshold is configured in a cache configuration file within the `<proxy-quorum-policy-scheme>` element. The element must be used within a `<proxy-scheme>` element. The following example demonstrates configuring the connection quorum threshold to ensures that `3` proxy service members are present in the cluster before the proxy service is allowed to accept TCP client connections:

```
<proxy-scheme>
   <scheme-name>proxy-with-quorum</scheme-name>
   <service-name>TcpProxyService</service-name>
   <acceptor-config>
      <tcp-acceptor>
         <local-address>
            <address>localhost</address>
            <port>32000</port>
         </local-address>
      </tcp-acceptor>
   </acceptor-config>
   <proxy-quorum-policy-scheme>
      <connect-quorum>3</connect-quorum>
   </proxy-quorum-policy-scheme>
   <autostart>true</autostart>
</proxy-scheme>
```

The `<proxy-quorum-policy-scheme>` element also supports the use of scheme references. In the below example, a `<proxy-quorum-policy-scheme>`, with the name `proxy-quorum`, is referenced from within the `<proxy-scheme>` element:

```
<proxy-scheme>
   <scheme-name>proxy-with-quorum</scheme-name>
```

```
   <service-name>TcpProxyService</service-name>
   ...
   <proxy-quorum-policy-scheme>
      <scheme-ref>proxy-quorum</scheme-ref>
   </proxy-quorum-policy-scheme>
   <autostart>true</autostart>
</proxy-scheme>
<proxy-scheme>
   <scheme-name>proxy-example</scheme-name>
   <service-name>TcpProxyService</service-name>
   ...
   <proxy-quorum-policy-scheme>
      <scheme-name>proxy-quorum</scheme-name>
      <connect-quorum>3</connect-quorum>
   </proxy-quorum-policy-scheme>
   <autostart>true</autostart>
</proxy-scheme>
```

Enabling Custom Action Policies

Custom action policies can be used instead of the default quorum policies for the Cluster service, Partitioned Cache service, and Proxy service. Custom action policies must implement the `com.tangosol.net.ActionPolicy` interface.

To enable a custom policy, add a `<class-name>` element within a quorum policy scheme element that contains the fully qualified name of the implementation class. The following example adds a custom action policy to the partitioned quorum policy for a distributed cache scheme definition:

```
<distributed-scheme>
   <scheme-name>partitioned-cache-with-quorum</scheme-name>
   <service-name>PartitionedCacheWithQuorum</service-name>
   <backing-map-scheme>
     <local-scheme/>
   </backing-map-scheme>
   <partitioned-quorum-policy-scheme>
      <class-name>package.MyCustomAction</class-name>
   </partitioned-quorum-policy-scheme>
   <autostart>true</autostart>
</distributed-scheme>
```

As an alternative, a factory class can create custom action policy instances. To define a factory class, use the `<class-factory-name>` element to enter the fully qualified class name and the `<method-name>` element to specify the name of a static factory method on the factory class which performs object instantiation. For example.

```
<distributed-scheme>
   <scheme-name>partitioned-cache-with-quorum</scheme-name>
   <service-name>PartitionedCacheWithQuorum</service-name>
   <backing-map-scheme>
     <local-scheme/>
   </backing-map-scheme>
   <partitioned-quorum-policy-scheme>
      <class-factory-name>package.Myfactory</class-factory-name>
      <method-name>createPolicy</method-name>
   </partitioned-quorum-policy-scheme>
   <autostart>true</autostart>
</distributed-scheme>
```

17

Cache Configurations by Example

This section provides a series of basic cache scheme definitions that can be used or modified as required. See Chapter 12, "Configuring Caches," for detailed instructions on how to configure caches. In addition, the samples in this chapter build upon one another and often use a `<scheme-ref>` element to reuse other samples as nested schemes. See "Using Scheme Inheritance" on page 12-9 for details on using the `<scheme-ref>` element. Lastly, these samples only specify a minimum number of settings, follow the embedded links to a 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
  - 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 are used as building blocks for clustered caches. See "Clustered Caches (accessible from multiple JVMs)" on page 17-5.

In-memory Cache

Example 17–1 uses a `local-scheme` to define an in-memory cache. The cache stores as much as the JVM heap allows.

Example 17–1 Configuration for a Local, In-memory Cache

```
<local-scheme>
   <scheme-name>SampleMemoryScheme</scheme-name>
</local-scheme>
```

NIO In-memory Cache

Example 17–2 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 17–2 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>` determines which elements to evict from the cache.

Example 17–3 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>` causes cache entries to automatically expire if they are not updated for a given time interval. When expired the cache invalidates the entry, and remove it from the cache.

Example 17–4 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 17–5 uses an `external-scheme` to define an on disk cache. The cache stores as much as the file system allows.

> **Note:** This example uses the `lh-file-manager` for its on disk storage implementation. See external-scheme for additional external storage options.

Example 17–5 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 B-76 for an alternate size limited external caching approach.

Example 17–6 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 17–7 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 B-38) 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 17-7 sample.

The `{cache-name}` macro is used to specify the name of the file the data is stored in. See "Using Parameter Macros" on page 12-12 for more information on this macro.

Example 17–7 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 17–8 illustrates using Berkeley DB rather then LH.

Example 17–8 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 17–9 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 are moved to the on disk (`<back-scheme>`). The front-scheme's `<eviction-policy>` determines 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 17-3. to implement the front and back of the cache.

Example 17–9 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 17–10 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 are read-through to the database, and cache writes are 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 "Sample CacheStore" on page 14-8 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 backs. See "Using Parameter Macros" on page 12-12 for more information on this macro.

Example 17–10 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 17-1). The cache service provides the capability to access local caches from other cluster nodes.

Replicated Cache

Example 17–11 uses the `replicated-scheme` element to define a clustered cache in which a copy of each cache entry is stored on all cluster nodes.

The sample in "In-memory Cache" on page 17-2 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 17–11 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 17-4, each cluster node would have a local overflow cache allowing for much greater storage capacity.

Example 17–12 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 17–13 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 17-2 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 B-30.

Example 17–13 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 17-4, 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 17–14 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 17–15 reuses the "Cache of a Database" on page 17-4 to define the database access.

Example 17–15 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 17–16 uses the serializer element in `distributed-scheme` to define a serializer that is used to serialize and deserialize user types. In this case, the partitioned cache uses 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 D, "POF User Type Configuration Elements" for more information on POF elements.

Example 17–16 Configuration for a Partitioned Cache with a Serializer

```
<distributed-scheme>
  <scheme-name>SamplePartitionedPofScheme</scheme-name>
  <service-name>PartitionedPofCache</service-name>
  <serializer>
    <instance>
      <class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>
    </instance>
  </serializer>
  <backing-map-scheme>
    <local-scheme/>
  </backing-map-scheme>
  <autostart>true</autostart>
</distributed-scheme>
```

Serializers that are defined in the `tangosol-coherence.xml` deployment descriptor can also be referenced.

Example 17–17 Partitioned Cache that References a Serializer

```
<distributed-scheme>
  <scheme-name>SamplePartitionedPofScheme</scheme-name>
  <service-name>PartitionedPofCache</service-name>
  <serializer>pof</serializer>
  <backing-map-scheme>
    <local-scheme/>
  </backing-map-scheme>
  <autostart>true</autostart>
</distributed-scheme>
```

Lastly a default serializer can be defined for all cache schemes and alleviates having to explicitly include a `<serializer>` element in each cache scheme definition. The global serializer definitions can also reference serializers that are defined in the `tangosol-coherence.xml` deployment descriptor

Example 17–18 Defining a Default Serializer

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <defaults>
      <serializer>pof</serializer>
   </defaults>
   ...
```

Near Cache

Example 17–19 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 maintains 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 17-2 sample is reused to define the "near" (`<front-scheme>`) cache, while the "Partitioned Cache" on page 17-6 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 17–19 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>
```

Part IV

Using the Programming API

Part IV contains the following chapters:

18

Serializing Objects

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. Because serialization is often the most expensive part of clustered data management, Coherence provides the following options for serializing/deserializing data:

- `com.tangosol.io.pof.PofSerializer` – The Portable Object Format (also referred to as POF) is a language agnostic binary format. POF was designed to be incredibly efficient in both space and time and has become the recommended serialization option in Coherence. See Chapter 19, "Using Portable Object Format."
- `java.io.Serializable` – The simplest, but slowest option.
- `java.io.Externalizable` – This requires developers to implement serialization manually, but can provide significant performance benefits. Compared to `java.io.Serializable`, this can cut serialized data size by a factor of two or more (especially helpful with Distributed caches, as they generally cache data in serialized form). Most importantly, CPU usage is dramatically reduced.
- `com.tangosol.io.ExternalizableLite` – This is very similar to `java.io.Externalizable`, but offers better performance and less memory usage by using a more efficient I/O stream implementation.
- `com.tangosol.run.xml.XmlBean` – A default implementation of `ExternalizableLite` (For more details, see the API Javadoc for `XmlBean`).

> **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.

19

Using Portable Object Format

Using Portable Object Format (POF) has many advantages ranging from performance benefits to language independence. It's recommended that you look closely at POF as your serialization solution when working with Coherence. For information on how to work with POF when building .NET extend clients, see "Building Integration Objects for .NET Clients" in *Oracle Coherence Client Guide*. For information on how to work with POF when building C++ extend clients, see "Building Integration Objects for C++ Clients" in *Oracle Coherence Client Guide*.

The following sections are included in this chapter:

Overview of POF Serialization

Serialization is the process of encoding an object into a binary format. It is a critical component to working with Coherence as data must be moved around the network. The Portable Object Format (also referred to as POF) is a language agnostic binary format. POF was designed to be incredibly efficient in both space and time and has become a cornerstone element in working with Coherence. For more information on the POF binary stream, see Appendix E, "The PIF-POF Binary Format."

There are several options available for serialization including standard Java serialization, POF, and your own custom serialization routines. Each has their own trade-offs. Standard Java serialization is easy to implement, supports cyclic object graphs and preserves object identity. Unfortunately, it's also comparatively slow, has a verbose binary format, and restricted to only Java objects.

The Portable Object Format has the following advantages:

- It's language independent with current support for Java, .NET, and C++.
- It's very efficient, in a simple test class with a `String`, a `long`, and three `ints`, (de)serialization was seven times faster, and the binary produced was one sixth the size compared with standard Java serialization.
- It's versionable, objects can evolve and have forward and backward compatibility.
- It supports the ability to externalize your serialization logic.
- It's indexed which allows for extracting values without deserializing the whole object. See "Using POF Extractors and POF Updaters" on page 19-11.

Using the POF API to Serialize Objects

POF requires serialization routines that know how to serialize and deserialize an object. There are two interfaces available for serializing objects: the `com.tangosol.io.pof.PortableObject` interface and the `com.tangosol.io.pof.PofSerializer` interface. POF also supports annotations that automatically implement serialization with out having to implement the `PortableObject` or `PofSerializer` interfaces. See "Using POF Annotations to Serialize Objects" on page 19-9 for details.

The following topics are included in this section:

- Implementing the PortableObject Interface
- Implementing the PofSerializer Interface
- Guidelines for Assigning POF Indexes
- Using POF Object References
- Registering POF Objects
- Configuring Coherence to Use the ConfigurablePofContext Class

Implementing the PortableObject Interface

The `PortableObject` interface is an interface made up of two methods:

- `public void readExternal(PofReader reader)`
- `public void writeExternal(PofWriter writer)`

POF elements are indexed by providing a numeric value for each element that you write or read from the POF stream. It's important to keep in mind that the indexes must be unique to each element written and read from the POF stream, especially when you have derived types involved because the indexes must be unique between the super class and the derived class. The following example demonstrates implementing the `PortableObject` interface:

```
public void readExternal(PofReader in)
   throws IOException
   {
   m_symbol    = (Symbol) in.readObject(0);
   m_ldtPlaced = in.readLong(1);
   m_fClosed   = in.readBoolean(2);
   }

public void writeExternal(PofWriter out)
   throws IOException
   {
   out.writeObject(0, m_symbol);
   out.writeLong(1, m_ldtPlaced);
   out.writeBoolean(2, m_fClosed);
   }
```

Implementing the PofSerializer Interface

The `PofSerializer` interface provides a way to externalize the serialization logic from the classes you want to serialize. This is particularly useful when you do not want to change the structure of your classes to work with POF and Coherence. The `PofSerializer` interface is also made up of two methods:

- `public Object deserialize(PofReader in)`
- `public void serialize(PofWriter out, Object o)`

As with the `PortableObject` interface, all elements written to or read from the POF stream must be uniquely indexed. Below is an example implementation of the `PofSerializer` interface:

Example 19–1 Implementation of the PofSerializer Interface

```
public Object deserialize(PofReader in)
   throws IOException
   {
   Symbol symbol    = (Symbol)in.readObject(0);
   long   ldtPlaced = in.readLong(1);
   bool   fClosed   = in.readBoolean(2);

   // mark that reading the object is done
   in.readRemainder(null);

   return new Trade(symbol, ldtPlaced, fClosed);
   }

public void serialize(PofWriter out, Object o)
   throws IOException
   {
   Trade trade = (Trade) o;
   out.writeObject(0, trade.getSymbol());
   out.writeLong(1, trade.getTimePlaced());
   out.writeBoolean(2, trade.isClosed());

   // mark that writing the object is done
   out.writeRemainder(null);
   }
```

Guidelines for Assigning POF Indexes

Use the following guidelines when assigning POF indexes to an object's attributes:

- Order your reads and writes: start with the lowest index value in the serialization routine and finish with the highest. When deserializing a value, perform reads in the same order as writes.
- Non-contiguous indexes are acceptable but must be read/written sequentially.
- When Subclassing reserve index ranges: index's are cumulative across derived types. As such, each derived type must be aware of the POF index range reserved by its super class.
- Do not re-purpose indexes: to support Evolvable, it's imperative that indexes of attributes are not re-purposed across class revisions.
- Label indexes: indexes that are labeled with a `public static final int`, are much easier to work with, especially when using POF Extractors and POF Updaters. See "Using POF Extractors and POF Updaters" on page 19-11. Indexes that are labeled must still be read and written out in the same order as mentioned above.

Using POF Object References

POF supports the use of object identities and references for objects that occur more than once in a POF stream. Objects are labeled with an identity and subsequent instances of a labeled object within the same POF stream are referenced by its identity. Object references are only supported for user defined object types.

Using references avoids encoding the same object multiple times and helps reduce the data size. References are typically used when a large number of sizeable objects are created multiple times or when objects use nested or circular data structures. However, for applications that contain large amounts of data but only few repeats, the use of object references provides minimal benefits due to the overhead incurred in keeping track of object identities and references.

The following topics are included in this section:

- Enabling POF Object References
- Registering POF Object Identities for Circular and Nested Objects

Enabling POF Object References

Object references are not enabled by default and must be enabled either within a `pof-config.xml` configuration file or programmatically when using the `SimplePofContext` class.

To enable object references in the POF configuration file, include the `<enable-references>` element, within the `<pof-config>` element, and set the value to `true`. For example:

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

<pof-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-pof-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-pof-config
   coherence-pof-config.xsd">
   ...
   <enable-references>true</enable-references>
</pof-config>
```

To enable object references when using the `SimplePofContext` class, call the `setReferenceEnabled` method with a property set to `true`. For example:

```
SimplePofContext ctx = new SimplePofContext();
ctx.setReferenceEnabled(true);
```

Note: Objects that have been written out with a POF context that does not support references cannot be read by a POF context that supports references. The opposite is also true.

Registering POF Object Identities for Circular and Nested Objects

Circular or nested objects must manually register an identity when creating the object. Otherwise, a child object that references the parent object will not find the identity of the parent in the reference map. Object identities can be registered from a serializer during the deserialization routine using the `com.tangosol.io.pof.PofReader.registerIdentity` method.

The following examples demonstrate two objects (`Customer` and `Product`) that contain a circular reference and a serializer implementation that registers an identity on the `Customer` object.

The `Customer` object is defined as follows:

```
public class Customer
   {
      private String m_sName;
      private Product m_product;

   public Customer(String sName)
      {
      m_sName = sName;
      }

   public Customer(String sName, Product product)
      {
      m_sName = sName;
      m_product = product;
      }

   public String getName()
      {
      return m_sName;
      }

   public Product getProduct()
      {
      return m_product;
      }

   public void setProduct(Product product)
      {
      m_product = product;
      }
   }
```

The `Product` object is defined as follows:

```
public class Product
   {
      private Customer m_customer;

   public Product(Customer customer)
      {
      m_customer = customer;
      }

   public Customer getCustomer()
      {
      return m_customer;
      }
   }
```

The serializer implementation registers an idetity during deserialization and is defined as follows:

```
public class CustomerSerializer implements PofSerializer
   {
   @Override
```

```
public void serialize(PofWriter pofWriter, Object o) throws IOException
{
   Customer customer = (Customer) o;
   pofWriter.writeString(0, customer.getName());
   pofWriter.writeObject(1, customer.getProduct());
   pofWriter.writeRemainder(null);
}

@Override
public Object deserialize(PofReader pofReader) throws IOException
   {
      String sName = pofReader.readString(0);
      Customer customer = new Customer(sName);

      pofReader.registerIdentity(customer);
      customer.setProduct((Product) pofReader.readObject(1));
      pofReader.readRemainder();
      return customer;
   }
}
```

Registering POF Objects

Coherence provides the `com.tangosol.io.pof.ConfigurablePofContext` serializer class which is responsible for mapping a POF serialized object to an appropriate serialization routine (either a `PofSerializer` implementation or by calling through the `PortableObject` interface).

Once your classes have serialization routines, the classes are registered with the `ConfigurablePofContext` class using a `pof-config.xml` configuration file. The POF configuration file has a `<user-type-list>` element that contains a list of classes that implement `PortableObject` or have a `PofSerializer` associated with them. The `<type-id>` for each class must be unique, and must match across all cluster instances (including extend clients). See Appendix D, "POF User Type Configuration Elements," for detailed reference of the POF configuration elements.

The following is an example of a POF configuration file:

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

<pof-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-pof-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-pof-config
   coherence-pof-config.xsd">
   <user-type-list>
      <include>coherence-pof-config.xml</include>

      <!-- User types must be above 1000 -->
      <user-type>
         <type-id>1001</type-id>
         <class-name>com.examples.MyTrade</class-name>
         <serializer>
            <class-name>com.examples.MyTradeSerializer</class-name>
         </serializer>
      </user-type>

      <user-type>
        <type-id>1002</type-id>
        <class-name>com.examples.MyPortableTrade</class-name>
```

```
      </user-type>
   </user-type-list>
</pof-config>
```

Note: Coherence reserves the first 1000 type-id's for internal use. As shown in the above example, the `<user-type-list>` includes the `coherence-pof-config.xml` file that is located in the root of the `coherence.jar` file. This is where Coherence specific user types are defined and should be included in all of your POF configuration files.

Configuring Coherence to Use the ConfigurablePofContext Class

Coherence can be configured to use the `ConfigurablePofContext` serializer class in three different ways based on the level of granularity that is required:

- Per Service – Each service provides a full `ConfigurablePofContext` serializer class configuration or references a predefined configuration that is included in the operational configuration file.
- All Services – All services use a global `ConfigurablePofContext` serializer class configuration. Services that provide their own configuration override the global configuration. The global configuration can also be a full configuration or reference a predefined configuration that is included in the operational configuration file.
- JVM – The `ConfigurablePofContext` serializer class is enabled for the whole JVM.

Configure the ConfigurablePofContext Class Per Service

To configure a service to use the `ConfigurablePofContext` class, add a `<serializer>` element to a cache scheme in a cache configuration file. See "serializer" on page B-100 for a complete reference of the `<serializer>` element.

The following example demonstrates a distributed cache that is configured to use the `ConfigurablePofContext` class and defines a custom POF configuration file:

```
<distributed-scheme>
  <scheme-name>example-distributed</scheme-name>
  <service-name>DistributedCache</service-name>
  <serializer>
     <instance>
        <class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>
        <init-params>
           <init-param>
              <param-type>String</param-type>
              <param-value>my-pof-config.xml</param-value>
           </init-param>
        </init-params>
     </instance>
  </serializer>
</distributed-scheme>
```

The following example references the default definition in the operational configuration file. Refer to "serializer" on page A-56 to see the default `ConfigurablePofContext` serializer definition.

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

```
    <serializer>pof</serializer>
 </distributed-scheme>
```

Configure the ConfigurablePofContext Class for All Services

To globally configure the `ConfigurablePofContext` class for all services, add a `<serializer>` element within the `<defaults>` element in a cache configuration file. Both of the below examples globally configure a serializer for all cache scheme definitions and do not require any additional configuration within individual cache scheme definitions. See "defaults" on page B-29 for a complete reference of the `<defaults>` element.

The following example demonstrates a global configuration for the `ConfigurablePofContext` class and defines a custom POF configuration file:

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <defaults>
      <serializer>
         <instance>
            <class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>
            <init-params>
               <init-param>
                  <param-type>String</param-type>
                  <param-value>my-pof-config.xml</param-value>
               </init-param>
            </init-params>
         </instance>
      </serializer>
   </defaults>
   ...
```

The following example references the default definition in the operational configuration file. Refer to "serializer" on page A-56 to see the default `ConfigurablePofContext` serializer definition.

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <defaults>
      <serializer>pof</serializer>
   </defaults>
   ...
```

Configure the ConfigurablePofContext Class For the JVM

An entire JVM instance can be configured to use POF using the following system properties:

- `tangosol.pof.enabled=true` - Enables POF for the entire JVM instance.
- `tangosol.pof.config=`*`CONFIG_FILE_PATH`* - The path to the POF configuration file you want to use. If the files is not in the classpath, then it must

be presented as a file resource (for example, `file:///opt/home/coherence/mycustom-pof-config.xml`).

Using POF Annotations to Serialize Objects

POF annotations provide an automated way to implement the serialization and deserialization routines for an object. POF annotations are serialized and deserialized using the `PofAnnotationSerializer` class which is an implementation of the `PofSerializer` interface. Annotations offer an alternative to using the `PortableObject` and `PofSerializer` interfaces and reduce the amount of time and code that is required to make objects serializable.

The following topics are included in this section:

- Annotating Objects for POF Serialization
- Registering POF Annotated Objects
- Enabling Automatic Indexing
- Providing a Custom Codec

Annotating Objects for POF Serialization

Two annotations are available to indicate that a class and its properties are POF serializable:

- `@Portable` – Marks the class as POF serializable. The annotation is only permitted at the class level and has no members.
- `@PortableProperty` – Marks a member variable or method accessor as a POF serialized attribute. Annotated methods must conform to accessor notation (`get`, `set`, `is`). Members can be used to specify POF indexes as well as custom codecs that are executed before or after serialization or deserialization. Index values may be omitted and automatically assigned. If a custom codec is not entered, the default codec is used.

The following example demonstrates annotating a class, method, and properties and assigning explicit property index values. See "Guidelines for Assigning POF Indexes" on page 19-3 for additional details on POF indexing.

```
@Portable
public class Person
   {
   @PortableProperty(0)
   public String getFirstName()
      {
      return m_firstName;
      }

   private String m_firstName;

   @PortableProperty(1)
   private String m_lastName;

   @PortableProperty(2)
   private int m_age;
}
```

Registering POF Annotated Objects

POF annotated objects must be registered in a `pof-config.xml` file within a `<user-type>` element. See Appendix D, "POF User Type Configuration Elements," for a detailed reference of the POF configuration elements. POF annotated objects use the `PofAnnotationSerializer` serializer if an object does not implement `PortableObject` and is annotated as `Portable`; however, the serializer is automatically assumed if an object is annotated and does not need to be included in the user type definition. The following example registers a user type for an annotated `Person` object:

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

<pof-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-pof-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-pof-config
   coherence-pof-config.xsd">
   <user-type-list>
      <include>coherence-pof-config.xml</include>

      <!-- User types must be above 1000 -->
      <user-type>
         <type-id>1001</type-id>
         <class-name>com.examples.Person</class-name>
      </user-type>
</pof-config>
```

Enabling Automatic Indexing

POF annotations support automatic indexing which alleviates the need to explicitly assign and manage index values. The index value can be omitted whenever defining the `@PortableProperty` annotation. Any property that does assign an explicit index value is not assigned an automatic index value. The automatic index algorithm can be described as follows:

Name	Explicit Index	Determined Index
c	1	1
a	omitted	0
b	omitted	2

Note: Automatic indexing does not currently support evolvable classes.

To enable automatic indexing, the `PofAnnotationSerializer` serializer class must be explicitly defined when registering the object as a user type in the POF configuration file. The `fAutoIndex` boolean parameter in the constructor enables automatic indexing and must be set to `true`. For example:

```
<user-type>
   <type-id>1001</type-id>
   <class-name>com.examples.Person</class-name>
   <serializer>
      <class-name>com.tangosol.io.pof.PofAnnotationSerializer</class-name>
         <init-params>
         <init-param>
```

```
            <param-type>int</param-type>
            <param-value>{type-id}</param-value>
         </init-param>
         <init-param>
            <param-type>class</param-type>
            <param-value>{class}</param-value>
         </init-param>
         <init-param>
            <param-type>boolean</param-type>
            <param-value>true</param-value>
         </init-param>
      </init-params>
   </serializer>
</user-type>
```

Providing a Custom Codec

Codecs allow code to be executed before or after serialization or deserialization. The codec defines how to encode and decode a portable property using the `PofWriter` and `PofReader` interfaces. Codecs are typically used for concrete implementations that could get lost when being deserialized or to explicitly call a specific method on the `PofWriter` interface before serializing an object.

To create a codec, create a class that implements the `com.tangosol.io.pof.reflect.Codec` interface. The following example demonstrates a codec that defines the concrete implementation of a linked list type:

```
public static class LinkedListCodec implements Codec
   {
   public Object decode(PofReader in, int index) throws IOException
      {
      return (List<String>) in.readCollection(index, new LinkedList<String>());
      }
   public void encode(PofWriter out, int index, Object value) throws IOException
      {
      out.writeCollection(index, (Collection) value);
      {
   }
```

To assign a codec to a property, enter the codec as a member of the `@PortableProperty` annotation. If a codec is not specified, a default codec (`DefaultCodec`) is used. The following example demonstrates assigning the above `LinkedListCodec` codec:

```
@PortableProperty(codec = LinkedListCodec.class)
private List<String> m_aliases;
```

Using POF Extractors and POF Updaters

In Coherence, the `ValueExtractor` and `ValueUpdater` interfaces are used to extract and update values of objects that are stored in the cache. The `PofExtractor` and `PofUpdater` interfaces take advantage of the POF indexed state to extract or update an object without the requirement to go through the full serialization/deserialization routines.

`PofExtractor` and `PofUpdater` adds flexibility in working with non-primitive types in Coherence. For many extend client cases, a corresponding Java classes in the grid is no longer required. Because POF extractors and POF updaters can navigate the

binary, the entire key/value does not have to be deserialized into Object form. This implies that indexing can be achieved by simply using POF extractors to pull a value to index on. However, a corresponding Java class is still required when using a cache store. In this case, the deserialized version of the key and value is passed to the cache store to write to the back end.

Navigating a POF object

Due to the fact that POF is indexed, it's possible to quickly traverse the binary to a specific element for extraction or updating. It's the responsibility of the `PofNavigator` interface to traverse a POF value object and return the desired POF value object. Out of the box, Coherence provides a `SimplePofPath` class that can navigate a POF value based on integer indexes. In the simplest form, provide the index of the attribute to be extracted/updated.

Consider the following example:

```
public class Contact
        implements PortableObject
    {
    ...
    // ----- PortableObject interface ---------------------------------------

    /**
    * {@inheritDoc}
    */
    public void readExternal(PofReader reader)
            throws IOException
        {
        m_sFirstName     = reader.readString(FIRSTNAME);
        m_sLastName      = reader.readString(LASTNAME);
        m_addrHome       = (Address) reader.readObject(HOME_ADDRESS);
        m_addrWork       = (Address) reader.readObject(WORK_ADDRESS);
        m_mapPhoneNumber = reader.readMap(PHONE_NUMBERS, null);
        }

    /**
    * {@inheritDoc}
    */
    public void writeExternal(PofWriter writer)
            throws IOException
        {
        writer.writeString(FIRSTNAME, m_sFirstName);
        writer.writeString(LASTNAME, m_sLastName);
        writer.writeObject(HOME_ADDRESS, m_addrHome);
        writer.writeObject(WORK_ADDRESS, m_addrWork);
        writer.writeMap(PHONE_NUMBERS, m_mapPhoneNumber);
        }

    ....

    // ----- constants ------------------------------------------------------

    /**
    * The POF index for the FirstName property
    */
    public static final int FIRSTNAME = 0;

    /**
    * The POF index for the LastName property
```

```
    */
    public static final int LASTNAME = 1;

    /**
    * The POF index for the HomeAddress property
    */
    public static final int HOME_ADDRESS = 2;

    /**
    * The POF index for the WorkAddress property
    */
    public static final int WORK_ADDRESS = 3;

    /**
    * The POF index for the PhoneNumbers property
    */
    public static final int PHONE_NUMBERS = 4;

    ...
}
```

Notice that there's a constant for each data member that is being written to and from the POF stream. This is an excellent practice to follow as it simplifies both writing your serialization routines and makes it easier to work with POF extractors and POF updaters. By labeling each index, it becomes much easier to think about the index. As mentioned above, in the simplest case, the work address can be pulled out of the contact by using the `WORK_ADDRESS` index. The `SimplePofPath` also allows using an `Array` of `int`s to traverse the `PofValues`. For example, to get the zip code of the work address use `[WORK_ADDRESS, ZIP]`. The example are discussed in more detail below.

Using POF Extractors

POF extractors are typically used when querying a cache and improves query performance. For example, using the class demonstrated above, to query the cache for all contacts with the last names `Jones`, the query is as follows:

```
ValueExtractor veName = new PofExtractor(String.class, Contact.LASTNAME);
Filter         filter = new EqualsFilter(veName, "Jones");

// find all entries that have a last name of Jones
Set setEntries = cache.entrySet(filter);
```

In the above case, `PofExtractor` has a convenience constructor that uses a `SimplePofPath` to retrieve a singular index, in our case the `Contact.LASTNAME` index. To find all contacts with the area code `01803`, the query is as follows:

```
ValueExtractor veZip = new PofExtractor(
   String.class, new SimplePofPath(new int[] {Contact.WORK_ADDRESS,
Address.ZIP}));

Filter filter = new EqualsFilter(veZip, "01803");

// find all entries that have a work address in the 01803 zip code
Set setEntries  = cache.entrySet(filter);
```

Notice that in the previous examples, the `PofExtractor` constructor has a first argument with the class of the extracted value or `null`. The reason for passing type information is that POF uses a compact form in the serialized value when possible. For example, some numeric values are represented as special POF intrinsic types in which

the type implies the value. As a result, POF requires the receiver of a value to have implicit knowledge of the type. `PofExtractor` uses the class supplied in the constructor as the source of the type information. If the class is `null`, `PofExtractor` infers the type from the serialized state, but the extracted type may differ from the expected type. `String` types, in fact, can be correctly inferred from the POF stream, so `null` is sufficient in the previous examples. In general, however, `null` should not be used.

Using POF Updaters

POF updaters work in the same way as POF extractors except that they update the value of an object rather than extract it. To change all entries with the last name of `Jones` to `Smith`, use the `UpdaterProcessor` class as follows:

```
ValueExtractor veName  = new PofExtractor(String.class, Contact.LASTNAME);
Filter         filter  = new EqualsFilter(veName, "Jones");
ValueUpdater   updater = new PofUpdater(Contact.LASTNAME);

// find all Contacts with the last name Jones and change them to have the last
// name "Smith"

cache.invokeAll(filter, new UpdaterProcessor(updater, "Smith"));
```

Note: while these examples operate on `String` based values, this functionality works on any POF encoded value.

20

Pre-Loading a Cache

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

The following sections are included in this chapter:

- Performing Bulk Loading and Processing
- Performing Distributed Bulk Loading

Performing Bulk Loading and Processing

Example 20–5, `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 20–1:

Example 20–1 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 returns the object it

just replaced in the cache (per the `java.util.Map` interface) which adds more unnecessary overhead. Loading the cache can be made much more efficient by using the `ConcurrentMap.putAll` method instead. This is illustrated in Example 20–2:

Example 20–2 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 20–3 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 20–4 illustrates a pattern to process query results in batches to avoid this problem:

Example 20–4 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 process results in parts, similar to the example above. However `LimitFilter` is meant for scenarios where the results are 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 20–5 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 20–5 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 thrashes 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)
                {
                // 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%");

    // 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 20–6 illustrates the terminal output from Coherence when you compile and run the example:

Example 20–6 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 allows 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 must decide on the following:

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

The application should consider the load that is 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 is 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 20–7 Retrieving Storage-Enabled Members of the Cache

```
protected Set getStorageMembers(NamedCache cache)
        {
        return ((PartitionedService) cache.getCacheService())
           .getOwnershipEnabledMembers();
        }
```

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 20–8 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 performs the load on each member. For example, use Coherence's `InvocationService` to launch the task. In this case, the implementation of `LoaderInvocable` must iterate through `memberFileNames` and process each file, loading its contents into the cache. The cache operations normally performed on the client must execute through the `LoaderInvocable`.

Example 20–9 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);
            }
        }
```

21

Using Cache Events

Coherence provides cache events using the JavaBean Event model. It is extremely simple to receive the events that you need, where you need them, regardless of where the changes are actually occurring in the cluster. Developers that are familiar with the JavaBean model should have no difficulties working with events, even in a complex cluster.

The following sections are included in this chapter:

Listener Interface and Event Object

In the JavaBeans Event model, there is an `EventListener` interface that all listeners must extend. Coherence provides a `MapListener` interface, which allows application logic to receive events when data in a Coherence cache is added, modified or removed.

Example 21–1 illustrates an excerpt from the `com.tangosol.util.MapListener` API.

Example 21–1 Excerpt from the MapListener API

```
public interface MapListener
        extends EventListener
    {
    /**
    * Invoked when a map entry has been inserted.
```

```
    *
    * @param evt  the MapEvent carrying the insert information
    */
    public void entryInserted(MapEvent evt);

    /**
    * Invoked when a map entry has been updated.
    *
    * @param evt  the MapEvent carrying the update information
    */
    public void entryUpdated(MapEvent evt);

    /**
    * Invoked when a map entry has been removed.
    *
    * @param evt  the MapEvent carrying the delete information
    */
    public void entryDeleted(MapEvent evt);
    }
```

An application object that implements the `MapListener` interface can sign up for events from any Coherence cache or class that implements the `ObservableMap` interface, simply by passing an instance of the application's `MapListener` implementation to a `addMapListener()` method.

The `MapEvent` object that is passed to the `MapListener` carries all of the necessary information about the event that has occurred, including the *source* (`ObservableMap`) that raised the event, the *identity* (key) that the event is related to, what the *action* was against that identity (insert, update or delete), what the old value was and what the new value is:

Example 21–2 illustrates an excerpt from the `com.tangosol.util.MapEvent` API.

Example 21–2 Excerpt from the MapEvent API

```
public class MapEvent
        extends EventObject
    {
    /**
    * Return an ObservableMap object on which this event has actually
    * occurred.
    *
    * @return an ObservableMap object
    */
    public ObservableMap getMap()

    /**
    * Return this event's id. The event id is an ENTRY_*
    * enumerated constants.
    *
    * @return an id
    */
    public int getId()

    /**
    * Return a key associated with this event.
    *
    * @return a key
    */
    public Object getKey()
```

```
/**
* Return an old value associated with this event.
* <p>
* The old value represents a value deleted from or updated in a map.
* It is always null for "insert" notifications.
*
* @return an old value
*/
public Object getOldValue()

/**
* Return a new value associated with this event.
* <p>
* The new value represents a new value inserted into or updated in
* a map. It is always null for "delete" notifications.
*
* @return a new value
*/
public Object getNewValue()


// ----- Object methods ---------------------------------------------------

/**
* Return a String representation of this MapEvent object.
*
* @return a String representation of this MapEvent object
*/
public String toString()


// ----- constants --------------------------------------------------------

/**
* This event indicates that an entry has been added to the map.
*/
public static final int ENTRY_INSERTED = 1;

/**
* This event indicates that an entry has been updated in the map.
*/
public static final int ENTRY_UPDATED  = 2;

/**
* This event indicates that an entry has been removed from the map.
*/
public static final int ENTRY_DELETED  = 3;
}
```

Understanding Event Guarantees

The partitioned cache service guarantees that under normal circumstances an event is delivered only once. However, there are two scenarios that could break this guarantee:

- A catastrophic cluster failure that caused the data loss (for example, simultaneous crash of two machines holding data). In this case, the `PARTITION_LOST` event is emitted to all registered `PartitionListener` instances on the server side.

- Client disconnect. In this case, the `MEMBER_LEFT` event is emitted to all registered `MemberListener` instances on the client side.

Caches and Classes that Support Events

All Coherence caches implement `ObservableMap`; in fact, the `NamedCache` interface that is implemented by all Coherence caches extends the `ObservableMap` interface. That means that an application can sign up to receive events from any cache, regardless of whether that cache is local, partitioned, near, replicated, using read-through, write-through, write-behind, overflow, disk storage, and so on.

Note: Regardless of the cache topology and the number of servers, and even if the modifications are being made by other servers, the events are delivered to the application's listeners.

In addition to the Coherence caches (those objects obtained through a Coherence cache factory), several other supporting classes in Coherence also implement the `ObservableMap` interface:

- `ObservableHashMap`
- `LocalCache`
- `OverflowMap`
- `NearCache`
- `ReadWriteBackingMap`
- `AbstractSerializationCache`, `SerializationCache`, and `SerializationPagedCache`
- `WrapperObservableMap`, `WrapperConcurrentMap`, and `WrapperNamedCache`

For a full list of published implementing classes, see the Coherence Javadoc for `ObservableMap`.

Signing Up for All Events

To sign up for events, simply pass an object that implements the `MapListener` interface to a `addMapListener` method on `ObservableMap`. The `addMapListener` methods are illustrated in Example 21–3.

Example 21–3 Methods on the ObservableMap API

```
public void addMapListener(MapListener listener);
public void addMapListener(MapListener listener, Object oKey, boolean fLite);
public void addMapListener(MapListener listener, Filter filter, boolean fLite);
```

Let's create an example `MapListener` implementation. Example 21–4 illustrates a sample `MapListener` implementation that prints each event as it receive.

Example 21–4 Sample MapListener Implementation

```
/**
* A MapListener implementation that prints each event as it receives
* them.
*/
```

```
public static class EventPrinter
        extends Base
        implements MapListener
    {
    public void entryInserted(MapEvent evt)
        {
        out(evt);
        }

    public void entryUpdated(MapEvent evt)
        {
        out(evt);
        }

    public void entryDeleted(MapEvent evt)
        {
        out(evt);
        }
    }
```

Using this implementation, it is extremely simple to print all events from any given cache (since all caches implement the `ObservableMap` interface):

```
cache.addMapListener(new EventPrinter());
```

Of course, to be able to later remove the listener, it is necessary to hold on to a reference to the listener:

Example 21–5 Holding a Reference to a Listener

```
Listener listener = new EventPrinter();
cache.addMapListener(listener);
m_listener = listener; // store the listener in a field
```

Later, to remove the listener:

Example 21–6 Removing a Listener

```
Listener listener = m_listener;
if (listener != null)
    {
    cache.removeMapListener(listener);
    m_listener = null; // clean up the listener field
    }
```

Each `addMapListener` method on the `ObservableMap` interface has a corresponding `removeMapListener` method. To remove a listener, use the `removeMapListener` method that corresponds to the `addMapListener` method that was used to add the listener.

Using an Inner Class as a MapListener

When creating an inner class to use as a `MapListener`, or when implementing a `MapListener` that only listens to one or two types of events (inserts, updates or deletes), you can use the `AbstractMapListener` base class. For example, the anonymous inner class in Example 21–7 prints out only the insert events for the cache.

Example 21–7 Inner Class that Prints Only Cache Insert Events

```
cache.addMapListener(new AbstractMapListener()
    {
    public void entryInserted(MapEvent evt)
        {
        out(evt);
        }
    });
```

Another helpful base class for creating a `MapListener` is the `MultiplexingMapListener`, which routes all events to a single method for handling. This class would allow you to simplify the `EventPrinter` example to the code illustrated in Example 21–8. Since only one method must be implemented to capture all events, the `MultiplexingMapListener` can also be very useful when creating an inner class to use as a `MapListener`.

Example 21–8 Routing All Events to a Single Method for Handling

```
public static class EventPrinter
        extends MultiplexingMapListener
    {
    public void onMapEvent(MapEvent evt)
        {
        out(evt);
        }
    }
```

Configuring a MapListener for a Cache

If the listener should always be on a particular cache, then place it into the cache configuration using the `<listener>` element and Coherence automatically adds the listener when it configures the cache.

Signing up for Events on specific identities

Signing up for events that occur against specific identities (keys) is just as simple. For example, to print all events that occur against the Integer key `5`:

```
cache.addMapListener(new EventPrinter(), new Integer(5), false);
```

Thus, the code in Example 21–9 would only trigger an event when the Integer key `5` is inserted or updated:

Example 21–9 Triggering an Event when a Specific Integer Key is Inserted or Updated

```
for (int i = 0; i < 10; ++i)
    {
    Integer key   = new Integer(i);
    String  value = "test value for key " + i;
    cache.put(key, value);
    }
```

Filtering Events

Similar to listening to a particular key, it is possible to listen to particular events. In Example 21–10 a listener is added to the cache with a filter that allows the listener to only receive delete events.

Example 21–10 Adding a Listener with Filter for Deleted Events

```
// Filters used with partitioned caches must be
// Serializable, Externalizable or ExternalizableLite
public class DeletedFilter
        implements Filter, Serializable
    {
    public boolean evaluate(Object o)
        {
        MapEvent evt = (MapEvent) o;
        return evt.getId() == MapEvent.ENTRY_DELETED;
        }
    }

cache.addMapListener(new EventPrinter(), new DeletedFilter(), false);
```

> **Note:** Filtering events versus filtering cached data:
>
> When building a filter for querying, the object that is passed to the evaluate method of the `Filter` is a value from the cache, or - if the filter implements the `EntryFilter` interface - the entire `Map.Entry` from the cache. When building a filter for filtering events for a `MapListener`, the object that is passed to the evaluate method of the filter is of type `MapEvent`.
>
> See "Advanced: Listening to Queries" on page 21-8, for more information on how to use a query filter to listen to cache events, .

If you then make the following sequence of calls:

```
cache.put("hello", "world");
cache.put("hello", "again");
cache.remove("hello");
```

The result would be:

```
CacheEvent{LocalCache deleted: key=hello, value=again}
```

For more information, see the "Advanced: Listening to Queries" on page 21-8.

"Lite" Events

By default, Coherence provides both the old and the new value as part of an event. Consider the following example:

Example 21–11 Inserting, Updating, and Removing a Value from the Cache

```
MapListener listener = new MultiplexingMapListener()
    {
    public void onMapEvent(MapEvent evt)
        {
        out("event has occurred: " + evt);
        out("(the wire-size of the event would have been "
            + ExternalizableHelper.toBinary(evt).length()
            + " bytes.)");
        }
    };
cache.addMapListener(listener);
```

```
// insert a 1KB value
cache.put("test", new byte[1024]);

// update with a 2KB value
cache.put("test", new byte[2048]);

// remove the 2KB value
cache.remove("test");
```

The output from running the test, illustrated inExample 21–12, shows that the first event carries the 1KB inserted value, the second event carries both the replaced 1KB value and the new 2KB value, and the third event carries the removed 2KB value.

Example 21–12 Sample Output

```
event has occurred: CacheEvent{LocalCache added: key=test, value=[B@a470b8}
(the wire-size of the event would have been 1283 bytes.)
event has occurred: CacheEvent{LocalCache updated: key=test, old value=[B@a470b8,
new value=[B@1c6f579}
(the wire-size of the event would have been 3340 bytes.)
event has occurred: CacheEvent{LocalCache deleted: key=test, value=[B@1c6f579}
(the wire-size of the event would have been 2307 bytes.)
```

When an application does not require the old and the new value to be included in the event, it can indicate that by requesting only "lite" events. When adding a listener, you can request lite events by using a `addMapListener` method that takes an additional boolean `fLite` parameter. In Example 21–11, the only change would be:

```
cache.addMapListener(listener, (Filter) null, true);
```

> **Note:** Obviously, a lite event's old value and new value may be null. However, even if you request lite events, the old and the new value *may* be included if there is no additional cost to generate and deliver the event. In other words, requesting that a `MapListener` receive lite events is simply a hint to the system that the `MapListener` does not have to know the old and new values for the event.

Advanced: Listening to Queries

All Coherence caches support querying by any criteria. When an application queries for data from a cache, the result is a point-in-time snapshot, either as a set of identities (`keySet`) or a set of identity/value pairs (`entrySet`). The mechanism for determining the contents of the resulting set is referred to as *filtering*, and it allows an application developer to construct queries of arbitrary complexity using a rich set of out-of-the-box filters (for example, equals, less-than, like, between, and so on), or to provide their own custom filters (for example, `XPath`).

The same filters that are used to query a cache can listen to events from a cache. For example, in a trading system it is possible to query for all open `Order` objects for a particular trader:

Example 21–13 Listening for Events from a Cache

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

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

To receive notifications of new trades being opened for that trader, closed by that trader or reassigned to or from another trader, the application can use the same filter:

Example 21–14 Listening for Events on an Object

```
// receive events for all trade IDs that this trader is interested in
mapTrades.addMapListener(listener, new MapEventFilter(filter), true);
```

The `MapEventFilter` converts a query filter into an event filter.

The `MapEventFilter` has several very powerful options, allowing an application listener to receive only the events that it is specifically interested in. More importantly for scalability and performance, only the desired events have to be communicated over the network, and they are communicated only to the servers and clients that have expressed interest in those specific events. Example 21–15 illustrates these scenarios.

Example 21–15 Using MapEventFilter to Filter on Various Events

```
// receive all events for all trades that this trader is interested in
nMask = MapEventFilter.E_ALL;
mapTrades.addMapListener(listener, new MapEventFilter(nMask, filter), true);

// receive events for all this trader's trades that are closed or
// re-assigned to a different trader
nMask = MapEventFilter.E_UPDATED_LEFT | MapEventFilter.E_DELETED;
mapTrades.addMapListener(listener, new MapEventFilter(nMask, filter), true);

// receive events for all trades as they are assigned to this trader
nMask = MapEventFilter.E_INSERTED | MapEventFilter.E_UPDATED_ENTERED;
mapTrades.addMapListener(listener, new MapEventFilter(nMask, filter), true);

// receive events only fornew trades assigned to this trader
nMask = MapEventFilter.E_INSERTED;
mapTrades.addMapListener(listener, new MapEventFilter(nMask, filter), true);
```

For more information on the various options supported, see the API documentation for `MapEventFilter`.

Filtering Events Versus Filtering Cached Data

When building a `Filter` for querying, the object that is passed to the evaluate method of the `Filter` is a value from the cache, or if the `Filter` implements the `EntryFilter` interface, the entire `Map.Entry` from the cache. When building a `Filter` for filtering events for a `MapListener`, the object that is passed to the evaluate method of the `Filter` is of type `MapEvent`.

The `MapEventFilter` converts a `Filter` that is used to do a query into a `Filter` that is used to filter events for a `MapListener`. In other words, the `MapEventFilter` is constructed from a `Filter` that queries a cache, and the resulting `MapEventFilter` is a filter that evaluates `MapEvent` objects by converting them into the objects that a query Filter would expect.

Advanced: Synthetic Events

Events usually reflect the changes being made to a cache. For example, one server is modifying one entry in a cache while another server is adding several items to a cache while a third server is removing an item from the same cache, all while fifty threads on

each and every server in the cluster is accessing data from the same cache! All the modifying actions produces events that any server within the cluster can choose to receive. We refer to these actions as *client actions*, and the events as being *dispatched to clients*, even though the "clients" in this case are actually servers. This is a natural concept in a true peer-to-peer architecture, such as a Coherence cluster: Each and every peer is both a client and a server, both consuming services from its peers and providing services to its peers. In a typical Java Enterprise application, a "peer" is an application server instance that is acting as a container for the application, and the "client" is that part of the application that is directly accessing and modifying the caches and listening to events from the caches.

Some events originate from within a cache itself. There are many examples, but the most common cases are:

- When entries automatically expire from a cache;
- When entries are evicted from a cache because the maximum size of the cache has been reached;
- When entries are transparently added to a cache as the result of a Read-Through operation;
- When entries in a cache are transparently updated as the result of a Read-Ahead or Refresh-Ahead operation.

Each of these represents a modification, but the modifications represent natural (and typically automatic) operations from within a cache. These events are referred to as *synthetic* events.

When necessary, an application can differentiate between client-induced and synthetic events simply by asking the event if it is synthetic. This information is carried on a sub-class of the `MapEvent`, called `CacheEvent`. Using the previous `EventPrinter` example, it is possible to print only the synthetic events:

Example 21–16 Determining Synthetic Events

```
public static class EventPrinter
        extends MultiplexingMapListener
    {
    public void onMapEvent(MapEvent evt)
        {
        if (evt instanceof CacheEvent && ((CacheEvent) evt).isSynthetic())
            {
            out(evt);
            )
        }
    }
```

For more information on this feature, see the API documentation for `CacheEvent`.

Advanced: Backing Map Events

While it is possible to listen to events from Coherence caches, each of which presents a local view of distributed, partitioned, replicated, near-cached, continuously-queried, read-through/write-through and and write-behind data, it is also possible to peek behind the curtains, so to speak.

For some advanced use cases, it may be necessary to "listen to" the "map" behind the "service". Replication, partitioning and other approaches to managing data in a distributed environment are all distribution *services*. The service still has to have

something in which to actually *manage* the data, and that *something* is called a "backing map".

Backing maps can be configured. If all the data for a particular cache should be kept in object form on the heap, then use an unlimited and non-expiring `LocalCache` (or a `SafeHashMap` if statistics are not required). If only a small number of items should be kept in memory, use a `LocalCache`. If data are to be read on demand from a database, then use a `ReadWriteBackingMap` (which knows how to read and write through an application's DAO implementation), and in turn give the `ReadWriteBackingMap` a backing map such as a `SafeHashMap` or a `LocalCache` to store its data in.

Some backing maps are observable. The events coming from these backing maps are not usually of direct interest to the application. Instead, Coherence translates them into actions that must be taken (by Coherence) to keep data synchronous and properly backed up, and it also translates them when appropriate into clustered events that are delivered throughout the cluster as requested by application listeners. For example, if a partitioned cache has a `LocalCache` as its backing map, and the local cache expires an entry, that event causes Coherence to expire all of the backup copies of that entry. Furthermore, if any listeners have been registered on the partitioned cache, and if the event matches their event filter(s), then that event is delivered to those listeners on the servers where those listeners were registered.

In some advanced use cases, an application must process events on the server where the data are being maintained, and it must do so on the structure (backing map) that is actually managing the data. In these cases, if the backing map is an observable map, a listener can be configured on the backing map or one can be programmatically added to the backing map. (If the backing map is not observable, it can be made observable by wrapping it in an `WrapperObservableMap`.)

Each backing map event is dispatched once and only once. However, multiple backing map events could be generated from a single `put`. For example, if the entry from `put` has to be redistributed, then distributed events (deleted from original node, and inserted in a new node) are created. In this case, the backing map listener is called multiple times for the single `put`.

Lastly, backing map listeners are always synchronous; they are fired on a thread that is doing the modification operation while holding the synchronization monitor for the backing map iteself. Often times for internal backingmap listeners, events are not processed immediately, but are queued and processed later asynchronously.

For more information on this feature, see the API documentation for `BackingMapManager`.

Producing Readable Backing MapListener Events from Distributed Caches

Backing `MapListener` events are returned from replicated caches in readable Java format. However, backing `MapListener` events returned from distributed caches are in internal Coherence format. The Coherence Incubator Common project provides an `AbstractMultiplexingBackingMapListener` class that enables you to obtain readable backing `MapListener` events from distributed caches. See `http://coherence.oracle.com/display/INCUBATOR/Coherence+Common` to download Coherence Common libraries.

To produce readable backing `MapListener` events from distributed caches:

1. Implement the `AbstractMultiplexingBackingMapListener` class.
2. Register the implementation in the `<listener>` section of the `backing-map-scheme` in the `cache-config` file.

3. Start the cache server application file and the client file with the `cacheconfig` Java property:

```
-Dtangosol.coherence.cacheconfig="cache-config.xml"
```

The `AbstractMultiplexingBackingMapListener` class provides an `onBackingMapEvent` method which you can override to specify how you would like the event returned.

The following listing of the `VerboseBackingMapListener` class is a sample implementation of `AbstractMultiplexingBackingMapListener`. The `onBackingMapEvent` method has been over-ridden to send the results to standard output.

Example 21–17 An AbstractMultiplexingBackingMapListener Implementation

```
import com.tangosol.net.BackingMapManagerContext;
import com.tangosol.util.MapEvent;


public class VerboseBackingMapListener extends
AbstractMultiplexingBackingMapListener {

        public VerboseBackingMapListener(BackingMapManagerContext context) {
                super(context);
        }

        @Override
        protected void onBackingMapEvent(MapEvent mapEvent, Cause cause) {

                System.out.printf("Thread: %s Cause: %s Event: %s\n",
Thread.currentThread().getName(), cause, mapEvent);

                try {
                        Thread.currentThread().sleep(5000);
                } catch (InterruptedException e) {
                        // add Auto-generated catch block
                 e.printStackTrace();
                }

        }
}
```

Example 21–18 is an example distributed scheme definition. In the `<listener>` section of the file, the `VerboseBackingMapListener` is identified as being of type `com.tangosol.net.BackingMapManagerContext`.

Example 21–18 Distributed Scheme Specifying a Verbose Backing Map Listener

```
<distributed-scheme>
   <scheme-name>my-dist-scheme</scheme-name>
   <service-name>DistributedCache</service-name>
      <backing-map-scheme>
         <read-write-backing-map-scheme>
            <internal-cache-scheme>
               <local-scheme>
                  <high-units>0</high-units>
                  <expiry-delay>0</expiry-delay>
               </local-scheme>
            </internal-cache-scheme>
```

```
            <cachestore-scheme>
               <class-scheme>
                  <class-name>CustomCacheStore</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>
            <listener>
               <class-scheme>
                  <class-name>VerboseBackingMapListener</class-name>
                     <init-params>
                        <init-param>
                           <param-type>com.tangosol.net.BackingMapManagerContext
                           </param-type>
                           <param-value>{manager-context}</param-value>
                        </init-param>
                     </init-params>
                  </class-scheme>
            </listener>
         </read-write-backing-map-scheme>
      </backing-map-scheme>
   <autostart>true</autostart>
</distributed-scheme>
```

Advanced: Synchronous Event Listeners

Some events are delivered asynchronously, so that application listeners do not disrupt the cache services that are generating the events. In some rare scenarios, asynchronous delivery can cause ambiguity of the ordering of events compared to the results of ongoing operations. To guarantee that the cache API operations and the events are ordered as if the local view of the clustered system were single-threaded, a `MapListener` must implement the `SynchronousListener` marker interface.

One example in Coherence itself that uses synchronous listeners is the Near Cache, which can use events to invalidate locally cached data ("Seppuku").

For more information on this feature, see the API documentation for `MapListenerSupport.SynchronousListener`.

22

Querying Data In a 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.

The following sections are included in this chapter:

Query Overview

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 do not use the `CacheLoader` interface to retrieve additional data that may satisfy the query). Thus, the data set should be loaded entirely into cache before queries are performed. In cases where the data set 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; for dedicated cache server instances, this implies (usually) that application classes must be installed in the cache server's 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).

Query Concepts

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 need only the `ReflectionExtractor` implementation of this interface. The implementation uses reflection to extract an attribute from a value object by referring to a method name which is typically a getter method. For example:

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

Any `void` argument method can be used, including `Object` methods like `toString()` (useful for prototype/debugging). Indexes may be either traditional field indexes (indexing fields of objects) or functional-based 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. See "Using Query Indexes" on page 22-3 for more information on indexes.

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.

```
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`:

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

The following example shows a routine to select the entries of a cache that satisfy a particular 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);
    }
```

The following example uses a filter to select and sort cache entries:

```
// 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.

Performing Simple Queries

Example 22–1 demonstrates how to create a simple query and uses the `GreaterEqualsFilter` filter. Coherence includes many pre-built filters located in the `com.tangosol.util.filter` package. See the *Oracle Coherence Java API Reference* for a complete list of all the pre-built filters.

Example 22–1 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.

Note: Although queries can be executed through a near cache, the query does not use the front portion of a near cache. If using a near cache with queries, the best approach is to use the following sequence:

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

Using Query Indexes

Query indexes allow values (or attributes of those values) and corresponding keys to be correlated within a `QueryMap` to increase query performance.

The following topics are included in this section:

- Creating an Index
- Creating User-Defined Indexes

Creating an Index

The `addIndex` method of the `QueryMap` class is used to create indexes. Any attribute able to be queried may be indexed using this method. The method includes three parameters:

```
addIndex(ValueExtractor extractor, boolean fOrdered, Comparator comparator)
```

Example 22–2 demonstrates how to create an index:

Example 22–2 Sample Code to Create an Index

```
NamedCache cache = CacheFactory.getCache("MyCache");
ValueExtractor extractor = new ReflectionExtractor("getAttribute");
cache.addIndex(extractor, true, null);
```

The `fOrdered` argument specifies whether the index structure is sorted. Sorted indexes are useful for range queries, such as "select all entries that fall between two dates" or "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 provide a custom `java.util.Comparator` for ordering the index.

The `addIndex` 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) exists. It is expected that an application calls this method to suggest an index even if the index may exist, just so that the application is certain that index has been suggested. For example in a distributed environment, each server likely suggests 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 requested the same indexes.

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. See *Oracle Coherence Management Guide* for more information on using JMX with Coherence.

Creating User-Defined Indexes

Applications can choose to create user-defined indexes to control which entries are added to the index. User-defined indexes are typically used to reduce the memory and processing overhead required to maintain an index. To create a user-defined index, an application must implement the `MapIndex` interface and the `IndexAwareExtractor` interfaces. This section also describes the `ConditionalIndex` and `ConditionalExtractor` classes which provide an implementation of the interfaces to create a conditional index that uses an associated filter to evaluate whether an entry should be indexed.

Implementing the MapIndex Interface

The `MapIndex` interface is used to correlate values stored in an indexed `Map` (or attributes of those values) to the corresponding keys in the indexed `Map`. Applications implement this interface to supply a custom index.

The following example implementation defines an index that only adds entries with non-null values. This would be useful in the case where there is a cache with a large number of entries and only a small subset have meaningful, non-null, values.

```
public class CustomMapIndex implements MapIndex
   {
   public void insert(Map.Entry entry)
       {
       if (entry.getValue()!= null)
           {
           ...
           }
       }
   ...
   }
```

In the above example, the value of the entry is checked for `null` before extraction, but it could be done after. If the value of the entry is `null` then nothing is inserted into the index. A similar check for `null` would also be required for the `MapIndex` `update` method. The rest of the `MapIndex` methods must be implemented appropriately as well.

Implementing the IndexAwareExtractor Interface

The `IndexAwareExtractor` interface is an extension to the `ValueExtractor` interface that supports the creation and destruction of a `MapIndex` index. Instances of this interface are intended to be used with the `QueryMap` API to support the creation of custom indexes. The following example demonstrates how to implement this interface and is for the example `CustomMapIndex` class that was created above:

```
public class CustomIndexAwareExtractor
        implements IndexAwareExtractor, ExternalizableLite, PortableObject
    {
    public CustomIndexAwareExtractor(ValueExtractor extractor)
        {
        m_extractor = extractor;
        }

    public MapIndex createIndex(boolean fOrdered, Comparator comparator,
            Map mapIndex)
        {
        ValueExtractor extractor = m_extractor;
        MapIndex       index     = (MapIndex) mapIndex.get(extractor);

        if (index != null)
            {
            throw new IllegalArgumentException(
                    "Repetitive addIndex call for " + this);
            }

        index = new CustomMapIndex(extractor, fOrdered, comparator);
        mapIndex.put(extractor, index);
        return index;
        }

    public MapIndex destroyIndex(Map mapIndex)
        {
        return (MapIndex) mapIndex.remove(m_extractor);
        }
    ...
    }
```

In the above example, an underlying extractor is actually used to create the index and ultimately extracts the values from the cache entries. The `IndexAwareExtractor` implementation is used to manage the creation and destruction of a custom `MapIndex` implementation while preserving the existing `QueryMap` interfaces.

The `IndexAwareExtractor` is passed into the `QueryMap.addIndex` and `QueryMap.removeIndex` calls. Coherence, in turn, calls `createIndex` and `destroyIndex` on the `IndexAwareExtractor`. Also note that it is the responsibility of the `IndexAwareExtractor` to maintain the `Map` of extractor-to-index associations that is passed into `createIndex` and `destroyIndex`.

Using a Conditional Index

A conditional index is a custom index that implements both the `MapIndex` and `IndexAwareExtractor` interfaces as described above and uses an associated filter to evaluate whether an entry should be indexed. An entry's extracted value is only added to the index if the filter evaluates to `true`. The implemented classes are `ConditionalIndex` and `ConditionalExtractor`, respectively.

The `ConditionalIndex` is created by a `ConditionalExtractor`. The filter and extractor used by the `ConditionalIndex` are set on the `ConditionalExtractor`

and passed to the `ConditionalIndex` constructor during the `QueryMap.addIndex` call.

The `ConditionalExtractor` is an `IndexAwareExtractor` implementation that is only used to create a `ConditionalIndex`. The underlying `ValueExtractor` is used for value extraction during index creation and is the extractor that is associated with the created `ConditionalIndex` in the given index map. Using the `ConditionalExtractor` to extract values in not supported. For example:

```
ValueExtractor extractor = new ReflectionExtractor("getLastName");
Filter filter = new NotEqualsFilter("getId", null);
ValueExtractor condExtractor = new ConditionalExtractor(filter, extractor, true);

// add the conditional index which should only contain the last name values for
the
// entries with non-null Ids
cache.addIndex(condExtractor, true, null);
```

Performing Batch Queries

In order to preserve memory on the client issuing a query, there are various techniques that can retrieve query results in batches.

Using the `key set` form of the queries – combined with `getAll()` – reduces memory consumption since the entire entry set is not deserialized on the client simultaneously. It also takes advantage of near caching. For example:

Example 22–3 Using a key set Query Format

```
// key set(Filter filter)
Set setKeys = cache.key set(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();
        }
    }
```

A `LimitFilter` may be used to limit the amount of data sent to the client, and also to provide paging. Example 22–4 demonstrates using a `LimitFilter`:

Example 22–4 Using a Limit Filter

```
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);
```

When using a distributed/partitioned cache, queries can be targeted to partitions and cache servers using a `PartitionedFilter`. This is the most efficient way of batching query results as each query request is targeted to a single cache server, thus reducing the number of servers that must respond to a request and making the most efficient use of the network.

> **Note:** Use of `PartitionedFilter` is limited to cluster members; it cannot be used by Coherence*Extend clients. Coherence*Extend clients may use the two techniques described above, or these queries can be implemented as an `Invocable` and executed remotely by a Coherence*Extend client.

To execute a query partition by partition:

```
DistributedCacheService service =
   (DistributedCacheService) cache.getCacheService();
int cPartitions = service.getPartitionCount();

PartitionSet parts = new PartitionSet(cPartitions);
for (int iPartition = 0; iPartition < cPartitions; iPartition++)
    {
    parts.add(iPartition);
    Filter filterPart = new PartitionedFilter(filter, parts);
    Set setEntriesPart = cache.entrySet(filterPart);

    // process the entries ...
    parts.remove(iPartition);
    }
```

Queries can also be executed on a server by server basis:

```
DistributedCacheService service =
   (DistributedCacheService) cache.getCacheService();
int cPartitions = service.getPartitionCount();

PartitionSet partsProcessed = new PartitionSet(cPartitions);
for (Iterator iter = service.getStorageEnabledMembers().iterator();
        iter.hasNext();)
    {
    Member member = (Member) iter.next();
    PartitionSet partsMember = service.getOwnedPartitions(member);

    // due to a redistribution some partitions may have been processed
    partsMember.remove(partsProcessed);
    Filter filterPart = new PartitionedFilter(filter, partsMember);
    Set setEntriesPart = cache.entrySet(filterPart);

    // process the entries ...
    partsProcessed.add(partsMember);
    }

// due to a possible redistribution, some partitions may have been skipped
if (!partsProcessed.isFull())
    {
    partsProcessed.invert();
    Filter filter = new PartitionedFilter(filter, partsProcessed);
```

```
// process the remaining entries ...
}
```

Performing Queries on Multi-Value Attributes

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

Example 22–5 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
// ...
```

Using Chained Extractors

The `ChainedExtractor` implementation allows chained invocation of zero-argument (accessor) methods. In Example 22–6, the extractor first uses reflection to call `getName()` on each cached `Person` object, and then uses reflection to call `length()` on the returned `String`.

Example 22–6 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`.

Evaluating Query Cost and Effectiveness

This section provides instructions for creating query explain plan records and query trace records in order to view the estimated cost and actual effectiveness of each filter in a query, respectively. The records are used to evaluate how Coherence is running the query and to determine why a query is performing poorly or how it can be modified in order to perform better. See also the `StorageManagerMBean` reference in *Oracle Coherence Management Guide* for details on viewing query-based statistics.

The following topics are included in this section:

- Creating Query Records

Creating Query Records

The `com.tangosol.util.aggregator.QueryRecorder` class produces an explain or trace record for a given filter. The class is an implementation of a parallel aggregator that is capable querying all nodes in a cluster and aggregating the results. The class supports two record types: an `EXPLAIN` record for showing the estimated cost for the filters in a query, and a `TRACE` record for showing the actual effectiveness of each filter in a query.

To create a query record, create a new `QueryRecorder` instance that specifies a `RecordType` parameter. Include the instance and the filter to be tested as parameters of the `aggregate` method. The following example creates an explain record:

```
NamedCache cache = CacheFactory.getCache("mycache");
cache.addIndex(new ReflectionExtractor("getAge"), true, null);

AllFilter filter = new AllFilter(new Filter[]
   {
    new OrFilter(
       new EqualsFilter(new ReflectionExtractor("getAge"), 16),
       new EqualsFilter(new ReflectionExtractor("getAge"), 19)),
    new EqualsFilter(new ReflectionExtractor("getLastName"), "Smith"),
    new EqualsFilter(new ReflectionExtractor("getFirstName"), "Bob"),
    });

QueryRecorder agent = new QueryRecorder(RecordType.EXPLAIN);
Object resultsExplain = cache.aggregate(filter, agent);

System.out.println("\n" + resultsExplain + "\n");
```

To create a trace record, change the `RecordType` parameter to `TRACE`:

```
QueryRecorder agent = new QueryRecorder(RecordType.TRACE);
```

Interpreting Query Records

Query records are used to evaluate the filters and indexes that make up a query. Explain plan records are used to evaluate the estimated cost associated with applying a filter. Trace records are used to evaluate how effective a filter is at reducing a key set.

This section provides a sample explain plan record and a sample trace record and discuss how to read and interpret the record. The records are based on an example query of 1500 entries that were located on a cluster of 4 storage-enabled nodes. The query consists of a filter that finds any people that are either age `16` or `19` with the first name `Bob` and the last name `Smith`. Lastly, and index is added for `getAge`. To run the complete example, see "Running The Query Record Example" on page 22-12.

```
NamedCache cache = CacheFactory.getCache("mycache");
cache.addIndex(new ReflectionExtractor("getAge"), true, null);

AllFilter filter = new AllFilter(new Filter[]
   {
    new OrFilter(
       new EqualsFilter(new ReflectionExtractor("getAge"), 16),
       new EqualsFilter(new ReflectionExtractor("getAge"), 19)),
    new EqualsFilter(new ReflectionExtractor("getLastName"), "Smith"),
```

```
    new EqualsFilter(new ReflectionExtractor("getFirstName"), "Bob"),
    });
```

Query Explain Plan Record

A query explain record provides the estimated cost of evaluating a filter as part of a query operation. The cost takes into account whether or not an index can be used by a filter. The cost evaluation is used to determine the order in which filters are applied when a query is performed. Filters that use an index have the lowest cost and get applied first.

Example 22–7 shows a typical query explain plan record. The record includes an Explain Plain table for evaluating each filter in the query and a Index Lookups table that lists each index that can be used by the filter. The columns are described as follows:

- Name – This column shows the name of each filter in the query. Composite filters show information for each of the filters within the composite filter.
- Index – This column shows whether or not an index can be used with the given filter. If an index is found, the number shown corresponds to the index number on the Index Lookups table. In the example, an ordered simple map index (`0`) was found for `getAge()`.
- Cost – This column shows an estimated cost of applying the filter. If an index can be used, the cost is given as 1. The value of 1 is used since the operation of applying the index requires just a single access to the index content. In the example, there are 4 storage-enabled cluster members and thus the cost reflects accessing the index on all four members. If no index exists, the cost is calculated as *`EVAL_COST * number of keys`*. The *`EVAL_COST`* value is a constant value and is 1000. This is intended to show the relative cost of doing a full scan to reduce the key set using the filter. In the example, there are 1500 cache entries which need to be evaluated. Querying indexed entries is always relatively inexpensive as compared to non-indexed entries but does not necessarily guarantee effectiveness.

The record in Example 22–7 shows that the equal filter for `getAge()` has a low cost because it has an associated index and would be applied before `getLastName()` and `getFirstName()`. However, the `getAge()` filter, while inexpensive, may not be very effective if all entries were either `16` and `19` and only few entries matched `Bob` and `Smith`. In this case, it is more effective to add an index for `getLastName()` and `getFirstName()`. Moreover, the cost (mainly memory consumption) associated with creating an index is wasted if the index does a poor job of reducing the key set.

Example 22–7 Sample Query Explain Plan Record

```
Explain Plan
Name                                    Index        Cost
================================================================================
com.tangosol.util.filter.AllFilter  | ----       | 0
  com.tangosol.util.filter.OrFilter | ----       | 0
    EqualsFilter(.getAge(), 16)     | 0          | 4
    EqualsFilter(.getAge(), 19)     | 0          | 4
  EqualsFilter(.getLastName(), Smit | 1          | 1500000
  EqualsFilter(.getFirstName(), Bob | 2          | 1500000


Index Lookups
Index  Description                                Extractor             Ordered
================================================================================
0      SimpleMapIndex: Extractor=.getAge(), Ord  .getAge()             true
```

```
1       No index found                          .getLastName()        false
2       No index found                          .getFirstName()       false
```

Query Trace Record

A query trace record provides the actual cost of evaluating a filter as part of a query operation. The cost takes into account whether or not an index can be used by a filter. The query is actually performed and the effectiveness of each filter at reducing the key set is shown.

Example 22–8 shows a typical query trace record. The record includes a Trace table that shows the effectiveness of each filter in the query and an Index Lookups table that lists each index that can be used by the filter. The columns are described as follows:

- Name – This column shows the name of each filter in the query. Composite filters show information for each of the filters within the composite filter.
- Index – This column shows whether or not an index can be used with the given filter. If an index is found, the number shown corresponds to the index number on the Index Lookups table. In the example, an ordered simple map index (`0`) was found for `getAge()`.
- Effectiveness – This column shows the amount a key set was actually reduced as a result of each filter. The value is given as *`prefilter_key_set_size`* | *`postfilter_key_set_size`* and is also presented as a percentage. The *`prefilter_key_set_size`* value represents the key set size prior to evaluating the filter or applying an index. The *`postfilter_key_set_size`* value represents the size of the key set remaining after evaluating the filter or applying an index. For a composite filter entry, the value is the overall results for its contained filters. Once a key set size can no longer be reduced based on an index, the resulting key set is deserialized and any non index filters are applied.
- Duration – This column shows the number of milliseconds spent evaluating the filter or applying an index. A value of 0 indicates that the time registered was below the reporting threshold. In the example, the `63` milliseconds is the result of having to deserialize the key set which is incurred on the first filter `getLastName()` only.

The record in Example 22–8 shows that it took approximately 63 milliseconds to reduce 1500 entries to find 100 entries with the first name `Bob`, last name `Smith`, and with an age of `16` or `19`. The key set of 1500 entries was initially reduced to 300 using the index for `getAge()`. The resulting 300 entries (because they could not be further reduced using an index) were then deserialized and reduced to 150 entries based on `getLastName()` and then reduced to 100 using `getFirstName()`. The example shows that an index on `getAge()` is well worth the resources because it was able to effectively reduce the key set by 1200 entries. An index on `getLastName` and `getFirstName` would increase the performance of the overall query but may not be worth the additional resource required to create the index.

Example 22–8 Sample Query Trace Record

```
Trace
Name                                    Index        Effectiveness          Duration
================================================================================
com.tangosol.util.filter.AllFilter  | ----       | 1500|300(80%)        | 0
  com.tangosol.util.filter.OrFilter | ----       | 1500|300(80%)        | 0
    EqualsFilter(.getAge(), 16)     | 0          | 1500|150(90%)        | 0
    EqualsFilter(.getAge(), 19)     | 0          | 1350|150(88%)        | 0
  EqualsFilter(.getLastName(), Smit | 1          | 300|300(0%)          | 0
```

```
  EqualsFilter(.getFirstName(), Bob | 2          | 300|300(0%)          | 0
com.tangosol.util.filter.AllFilter  | ----       | 300|100(66%)         | 63
  EqualsFilter(.getLastName(), Smit | ----       | 300|150(50%)         | 63
  EqualsFilter(.getFirstName(), Bob | ----       | 150|100(33%)         | 0


Index Lookups
Index  Description                                 Extractor             Ordered
================================================================================
0      SimpleMapIndex: Extractor=.getAge(), Ord  .getAge()             true
1      No index found                            .getLastName()        false
2      No index found                            .getFirstName()       false
```

Running The Query Record Example

The following example is a simple class that demonstrates creating query records. The class loads a distributed cache (`mycache`) with 1500 `Person` objects, creates an index on an attribute, performs a query, and creates both a query explain plan record and a query trace record that is emitted to the console before the class exits.

Example 22–9 A Query Record Example

```
import com.tangosol.net.CacheFactory;
import com.tangosol.net.NamedCache;
import com.tangosol.util.Filter;
import com.tangosol.util.aggregator.QueryRecorder;
import static com.tangosol.util.aggregator.QueryRecorder.RecordType;
import com.tangosol.util.extractor.ReflectionExtractor;
import com.tangosol.util.filter.AllFilter;
import com.tangosol.util.filter.EqualsFilter;
import com.tangosol.util.filter.OrFilter;
import java.io.Serializable;
import java.util.Properties;

public class QueryRecordExanple
   {
   public static void main(String[] args) {

      testExplain();
      testTrace();
   }

   public static void testExplain()
   {
      NamedCache cache = CacheFactory.getCache("mycache");
      cache.addIndex(new ReflectionExtractor("getAge"), true, null);
      PopulateCache(cache);

      AllFilter filter = new AllFilter(new Filter[]
      {
         new OrFilter(
            new EqualsFilter(new ReflectionExtractor("getAge"), 16),
            new EqualsFilter(new ReflectionExtractor("getAge"), 19)),
         new EqualsFilter(new ReflectionExtractor("getLastName"), "Smith"),
         new EqualsFilter(new ReflectionExtractor("getFirstName"), "Bob"),
      });

      QueryRecorder agent = new QueryRecorder(RecordType.EXPLAIN);
      Object resultsExplain = cache.aggregate(filter, agent);
```

```
   System.out.println("\nExplain Plan=\n" + resultsExplain + "\n");
}

public static void testTrace()
{
   NamedCache cache = CacheFactory.getCache("hello-example");
   cache.addIndex(new ReflectionExtractor("getAge"), true, null);
   PopulateCache(cache);

   AllFilter filter = new AllFilter(new Filter[]
   {
      new OrFilter(
         new EqualsFilter(new ReflectionExtractor("getAge"), 16),
         new EqualsFilter(new ReflectionExtractor("getAge"), 19)),
      new EqualsFilter(new ReflectionExtractor("getLastName"), "Smith"),
      new EqualsFilter(new ReflectionExtractor("getFirstName"), "Bob"),
   });

   QueryRecorder agent = new QueryRecorder(RecordType.TRACE);
   Object resultsExplain = cache.aggregate(filter, agent);
   System.out.println("\nTrace =\n" + resultsExplain + "\n");
}

private static void PopulateCache(NamedCache cache)
   {
      for (int i = 0; i < 1500; ++i)
         {
            Person person = new Person(i % 3 == 0 ? "Joe" : "Bob",
               i % 2 == 0 ? "Smith" : "Jones", 15 + i % 10);
            cache.put("key" + i, person);
         }
   }

public static class Person implements Serializable
   {
   public Person(String sFirstName, String sLastName, int nAge)
      {
         m_sFirstName = sFirstName;
         m_sLastName = sLastName;
         m_nAge = nAge;
      }

   public String getFirstName()
      {
         return m_sFirstName;
      }

   public String getLastName()
      {
         return m_sLastName;
      }

   public int getAge()
      {
         return m_nAge;
      }

   public String toString()
      {
         return "Person( " +m_sFirstName + " " + m_sLastName + " : " +
```

```
                m_nAge + ")";
            }
        private String m_sFirstName;
        private String m_sLastName;
        private int m_nAge;
    }
}
```

23

Using Continuous Query Caching

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 22, "Querying Data In a Cache."

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.

The following sections are included in this chapter:

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 always be maintained in an up-to-date manner. 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 caches.

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 B, "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 23–1 A Query for a Continuous Query Cache

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

See Chapter 22, "Querying Data In a Cache" for more information on queries.

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

Example 23–2 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 23–3 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 are 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 23–4 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 23–5 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 is automatically 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 23–6 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 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 23–7 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 23–8 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, the same entry can appear 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 receives 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 23–9 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 Chapter 21, "Using Cache Events," 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` have their events delivered asynchronously. However, the `ContinuousQueryCache` does respect the option for synchronous events as provided by the `SynchronousListener` interface. See Chapter 23, "Using Continuous Query Caching," for more information on this option.

Making the ContinuousQueryCache Read-Only

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

Example 23–10 Making the Continuous Query Cache Read-Only

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

A read-only `ContinuousQueryCache` does 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.

24

Processing Data In a Cache

Coherence provides the ideal infrastructure for building Data Grid services and the client and server-based applications that use a Data Grid. At a basic level, Coherence can manage an immense amount of data across a large number of servers in a grid; it can provide close to zero latency access for that data; it supports parallel queries across that data in a map-reduce manner; and it supports integration with database and EIS systems that act as the system of record for that data. Additionally, Coherence provides several services that are ideal for building effective data grids.

The following sections are included in this chapter:

Targeted Execution

Coherence provides for the ability to execute an agent against an entry in any map of data managed by the Data Grid:

```
map.invoke(key, agent);
```

In the case of partitioned data, the agent executes on the grid node that owns the data to execute against. The queuing, concurrency management, agent execution, data access by the agent, and data modification by the agent all occur on that grid node. (Only the synchronous backup of the resultant data modification, if any, requires additional network traffic.) For many processing purposes, it is much more efficient to move the serialized form of the agent (usually only a few hundred bytes, at most) than to handle distributed concurrency control, coherency and data updates.

For request/response processing, the agent returns a result:

```
Object oResult = map.invoke(key, agent);
```

In other words, Coherence as a Data Grid determines the location to execute the agent based on the configuration for the data topology, move the agent there, execute the

agent (automatically handling concurrency control for the item while executing the agent), back up the modifications if any, and return a result.

Parallel Execution

Coherence provides map-reduce functionality which allows agents to be executed in parallel against a collection of entries across all nodes in the grid. Parallel execution allows large amounts of data to be processed by balancing the work across the grid. The `invokeAll` method is used as follows:

```
map.invokeAll(collectionKeys, agent);
```

For request/response processing, the agent returns one result for each key processed:

```
Map mapResults = map.invokeAll(collectionKeys, agent);
```

Coherence determines the optimal location(s) to execute the agent based on the configuration for the data topology, moves the agent there, executes the agent (automatically handling concurrency control for the item(s) while executing the agent), backing up the modifications if any, and returning the coalesced results. See "Data Grid Aggregation" on page 24-6 for instructions on performing aggregation against a result set.

Query-Based Execution

Coherence supports the ability to query across the entire data grid. For example, in a trading system it is possible to query for all open `Order` objects for a particular trader:

Example 24–1 Querying Across a Data Grid

```
NamedCache map    = CacheFactory.getCache("trades");
Filter     filter = new AndFilter(new EqualsFilter("getTrader", traderid),
                                  new EqualsFilter("getStatus", Status.OPEN));
Set setOpenTradeIds = mapTrades.keySet(filter);
```

By combining this feature with Parallel Execution in the data grid, Coherence provides for the ability to execute an agent against a query. As in the previous section, the execution occurs in parallel, and instead of returning the identities or entries that match the query, Coherence executes the agents against the entries:

```
map.invokeAll(filter, agent);
```

For request/response processing, the agent returns one result for each key processed:

```
Map mapResults = map.invokeAll(filter, agent);
```

In other words, Coherence combines its Parallel Query and its Parallel Execution to achieve query-based agent invocation against a Data Grid.

Data-Grid-Wide Execution

Passing an instance of `AlwaysFilter` (or a null) to the `invokeAll` method causes the passed agent to be executed against all entries in the `InvocableMap`:

```
map.invokeAll((Filter) null, agent);
```

As with the other types of agent invocation, request/response processing is supported:

```
Map mapResults = map.invokeAll((Filter) null, agent);
```

An application can process all the data spread across a particular map in the Data Grid with a single line of code.

Agents for Targeted, Parallel and Query-Based Execution

An agent implements the `EntryProcessor` interface, typically by extending the `AbstractProcessor` class.

Several agents are included with Coherence, including:

- `AbstractProcessor` - an abstract base class for building an `EntryProcessor`
- `ExtractorProcessor` - extracts and returns a value (such as a property value) from an object stored in an `InvocableMap`
- `CompositeProcessor` - bundles a collection of `EntryProcessor` objects that are invoked sequentially against the same Entry
- `ConditionalProcessor` - conditionally invokes an `EntryProcessor` if a `Filter` against the Entry-to-process evaluates to true
- `PropertyProcessor` - an abstract base class for `EntryProcessor` implementations that depend on a `PropertyManipulator`
- `NumberIncrementor` - pre- or post-increments any property of a primitive integral type, and `Byte`, `Short`, `Integer`, `Long`, `Float`, `Double`, `BigInteger`, `BigDecimal`
- `NumberMultiplier` - multiplies any property of a primitive integral type, and `Byte`, `Short`, `Integer`, `Long`, `Float`, `Double`, `BigInteger`, `BigDecimal`, and returns either the previous or new value

The `EntryProcessor` interface (contained within the `InvocableMap` interface) contains only two methods:

Example 24–2 Methods in the EntryProcessor Interface

```
/**
* An invocable agent that operates against the Entry objects within a
* Map.
*/
public interface EntryProcessor
        extends Serializable
    {
    /**
    * Process a Map Entry.
    *
    * @param entry  the Entry to process
    *
    * @return the result of the processing, if any
    */
    public Object process(Entry entry);

    /**
    * Process a Set of InvocableMap Entry objects. This method is
    * semantically equivalent to:
    * <pre>
    *   Map mapResults = new ListMap();
    *   for (Iterator iter = setEntries.iterator(); iter.hasNext(); )
    *       {
```

```
*       Entry entry = (Entry) iter.next();
*       mapResults.put(entry.getKey(), process(entry));
*       }
*   return mapResults;
* </pre>
*
* @param setEntries  a read-only Set of InvocableMap Entry objects to
*                    process
*
* @return a Map containing the results of the processing, up to one
*         entry for each InvocableMap Entry that was processed, keyed
*         by the keys of the Map that were processed, with a
*         corresponding value being the result of the processing for
*         each key
*/
public Map processAll(Set setEntries);
}
```

(The `AbstractProcessor` implements the `processAll` method as described in the previous example.)

The `InvocableMap.Entry` that is passed to an `EntryProcessor` is an extension of the `Map.Entry` interface that allows an `EntryProcessor` implementation to obtain the necessary information about the entry and to make the necessary modifications in the most efficient manner possible:

Example 24–3 InvocableMap.Entry API

```
/**
* An InvocableMap Entry contains additional information and exposes
* additional operations that the basic Map Entry does not. It allows
* non-existent entries to be represented, thus allowing their optional
* creation. It allows existent entries to be removed from the Map. It
* supports several optimizations that can ultimately be mapped
* through to indexes and other data structures of the underlying Map.
*/
public interface Entry
        extends Map.Entry
    {
    // ----- Map Entry interface -------------------------------------

    /**
    * Return the key corresponding to this entry. The resultant key does
    * not necessarily exist within the containing Map, which is to say
    * that <tt>InvocableMap.this.containsKey(getKey)</tt> could return
    * false. To test for the presence of this key within the Map, use
    * {@link #isPresent}, and to create the entry for the key, use
    * {@link #setValue}.
     *
    * @return the key corresponding to this entry; may be null if the
    *         underlying Map supports null keys
    */
    public Object getKey();

    /**
    * Return the value corresponding to this entry. If the entry does
    * not exist, then the value is null. To differentiate between
    * a null value and a non-existent entry, use {@link #isPresent}.
    * <p/>
    * <b>Note:</b> any modifications to the value retrieved using this
```

```
* method are not guaranteed to persist unless followed by a
* {@link #setValue} or {@link #update} call.
*
* @return the value corresponding to this entry; may be null if the
*         value is null or if the Entry does not exist in the Map
*/
public Object getValue();

/**
* Store the value corresponding to this entry. If the entry does
* not exist, then the entry is created by invoking this method,
* even with a null value (assuming the Map supports null values).
*
* @param oValue  the new value for this Entry
*
* @return the previous value of this Entry, or null if the Entry did
*         not exist
*/
public Object setValue(Object oValue);

// ----- InvocableMap Entry interface ---------------------------

/**
* Store the value corresponding to this entry. If the entry does
* not exist, then the entry is created by invoking this method,
* even with a null value (assuming the Map supports null values).
* <p/>
* Unlike the other form of {@link #setValue(Object) setValue}, this
* form does not return the previous value, and consequently may be
* significantly less expensive (in terms of cost of execution) for
* certain Map implementations.
*
* @param oValue      the new value for this Entry
* @param fSynthetic  pass true only if the insertion into or
*                    modification of the Map should be treated as a
*                    synthetic event
*/
public void setValue(Object oValue, boolean fSynthetic);

/**
* Extract a value out of the Entry's value. Calling this method is
* semantically equivalent to
* <tt>extractor.extract(entry.getValue())</tt>, but this method may
* be significantly less expensive because the resultant value may be
* obtained from a forward index, for example.
*
* @param extractor  a ValueExtractor to apply to the Entry's value
*
* @return the extracted value
*/
public Object extract(ValueExtractor extractor);

/**
* Update the Entry's value. Calling this method is semantically
* equivalent to:
* <pre>
*   Object oTarget = entry.getValue();
*   updater.update(oTarget, oValue);
*   entry.setValue(oTarget, false);
* </pre>
```

```
    * The benefit of using this method is that it may allow the Entry
    * implementation to significantly optimize the operation, such as
    * for purposes of delta updates and backup maintenance.
    *
    * @param updater  a ValueUpdater used to modify the Entry's value
    */
    public void update(ValueUpdater updater, Object oValue);

    /**
    * Determine if this Entry exists in the Map. If the Entry is not
    * present, it can be created by calling {@link #setValue} or
    * {@link #setValue}. If the Entry is present, it can be destroyed by
    * calling {@link #remove}.
    *
    * @return true iff this Entry is existent in the containing Map
    */
    public boolean isPresent();

    /**
    * Remove this Entry from the Map if it is present in the Map.
    * <p/>
    * This method supports both the operation corresponding to
    * {@link Map#remove} and synthetic operations such as
    * eviction. If the containing Map does not differentiate between
    * the two, then this method must be identical to
    * <tt>InvocableMap.this.remove(getKey())</tt>.
    *
    * @param fSynthetic  pass true only if the removal from the Map
    *                    should be treated as a synthetic event
    */
    public void remove(boolean fSynthetic);
    }
```

Data Grid Aggregation

In addition to *scalar* agents, the `InvocableMap` interface also supports entry aggregators that perform operations against a subset of entries to obtain a single result. Entry aggregation occurs in parallel across the grid to provide map-reduce support when working with large amounts of data.

Example 24–4 Aggregation in the InvocableMap API

```
/**
* Perform an aggregating operation against the entries specified by the
* passed keys.
*
* @param collKeys  the Collection of keys that specify the entries within
*                  this Map to aggregate across
* @param agent     the EntryAggregator that is used to aggregate across
*                  the specified entries of this Map
*
* @return the result of the aggregation
*/
public Object aggregate(Collection collKeys, EntryAggregator agent);

/**
* Perform an aggregating operation against the set of entries that are
* selected by the given Filter.
```

```
* <p/>
* <b>Note:</b> calling this method on partitioned caches requires a
* Coherence Enterprise Edition (or higher) license.
*
* @param filter  the Filter that is used to select entries within this
*                Map to aggregate across
* @param agent   the EntryAggregator that is used to aggregate across
*                the selected entries of this Map
*
* @return the result of the aggregation
*/
public Object aggregate(Filter filter, EntryAggregator agent);
```

A simple `EntryAggregator` processes a set of `InvocableMap.Entry` objects to achieve a result:

Example 24–5 EntryAggregator API

```
/**
* An EntryAggregator represents processing that can be directed to occur
* against some subset of the entries in an InvocableMap, resulting in a
* aggregated result. Common examples of aggregation include functions
* such as min(), max() and avg(). However, the concept of aggregation
* applies to any process that must evaluate a group of entries to
* come up with a single answer.
*/
public interface EntryAggregator
        extends Serializable
    {
    /**
    * Process a set of InvocableMap Entry objects to produce an
    * aggregated result.
    *
    * @param setEntries  a Set of read-only InvocableMap Entry objects to
    *                    aggregate
    *
    * @return the aggregated result from processing the entries
    */
    public Object aggregate(Set setEntries);
    }
```

For efficient execution in a Data Grid, an aggregation process must be designed to operate in a parallel manner.

Example 24–6 ParallelAwareAggregator API for running Aggregation in Parallel

```
/**
* A ParallelAwareAggregator is an advanced extension to EntryAggregator
* that is explicitly capable of being run in parallel, for example in a
* distributed environment.
*/
public interface ParallelAwareAggregator
        extends EntryAggregator
    {
    /**
    * Get an aggregator that can take the place of this aggregator in
    * situations in which the InvocableMap can aggregate in parallel.
    *
    * @return the aggregator that is run in parallel
    */
```

```
    public EntryAggregator getParallelAggregator();

    /**
    * Aggregate the results of the parallel aggregations.
    *
    * @return the aggregation of the parallel aggregation results
    */
    public Object aggregateResults(Collection collResults);
    }
```

Coherence comes with all of the natural aggregation functions, including:

- `Count`
- `DistinctValues`
- `DoubleAverage`
- `DoubleMax`
- `DoubleMin`
- `DoubleSum`
- `LongMax`
- `LongMin`
- `LongSum`

> **Note:** All aggregators that come with Coherence are parallel-aware.

See the `com.tangosol.util.aggregator` package for a list of Coherence aggregators. To implement your own aggregator, see the `AbstractAggregator` abstract base class.

Node-Based Execution

Coherence provides an Invocation Service which allows execution of single-pass agents (called Invocable objects) anywhere within the grid. The agents can be executed on any particular node of the grid, in parallel on any particular set of nodes in the grid, or in parallel on all nodes of the grid.

An invocation service is configured using the `<invocation-scheme>` element in the cache configuration file. Using the name of the service, the application can easily obtain a reference to the service:

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

Agents are simply runnable classes that are part of the application. An example of a simple agent is one designed to request a GC from the JVM:

Example 24–7 Simple Agent to Request Garbage Collection

```
/**
* Agent that issues a garbage collection.
*/
public class GCAgent
        extends AbstractInvocable
    {
```

```
    public void run()
        {
        System.gc();
        }
    }
```

To execute that agent across the entire cluster, it takes one line of code:

```
service.execute(new GCAgent(), null, null);
```

Here is an example of an agent that supports a grid-wide request/response model:

Example 24–8 Agent to Support a Grid-Wide Request and Response Model

```
/**
* Agent that determines how much free memory a grid node has.
*/
public class FreeMemAgent
        extends AbstractInvocable
    {
    public void run()
        {
        Runtime runtime = Runtime.getRuntime();
        int cbFree  = runtime.freeMemory();
        int cbTotal = runtime.totalMemory();
        setResult(new int[] {cbFree, cbTotal});
        }
    }
```

To execute that agent across the entire grid and retrieve all the results from it, **it still takes only one line of code**:

```
Map map = service.query(new FreeMemAgent(), null);
```

While it is easy to do a grid-wide request/response, it takes a bit more code to print the results:

Example 24–9 Printing the Results from a Grid-Wide Request or Response

```
Iterator iter = map.entrySet().iterator();
while (iter.hasNext())
    {
    Map.Entry entry  = (Map.Entry) iter.next();
    Member    member = (Member) entry.getKey();
    int[]     anInfo = (int[]) entry.getValue();
    if (anInfo != null) // nullif member died
        System.out.println("Member " + member + " has "
            + anInfo[0] + " bytes free out of "
            + anInfo[1] + " bytes total");
    }
```

The agent operations can be stateful, which means that their invocation state is serialized and transmitted to the grid nodes on which the agent is to be run.

Example 24–10 Stateful Agent Operations

```
/**
* Agent that carries some state with it.
*/
public class StatefulAgent
        extends AbstractInvocable
```

```
    {
    public StatefulAgent(String sKey)
        {
        m_sKey = sKey;
        }

    public void run()
        {
        // the agent has the key that it was constructed with
        String sKey = m_sKey;
        // ...
        }

    private String m_sKey;
    }
```

Work Manager

Coherence provides a grid-enabled implementation of the *CommonJ Work Manager*. Using a Work Manager, an application can submit a collection of work that must be executed. The Work Manager distributes that work in such a way that it is executed in parallel, typically across the grid. In other words, if there are ten work items submitted and ten servers in the grid, then each server likely processes one work item. Further, the distribution of work items across the grid can be tailored, so that certain servers (for example, one that acts as a gateway to a particular mainframe service) is the first choice to run certain work items, for sake of efficiency and locality of data.

The application can then wait for the work to be completed, and can provide a timeout for how long it can wait. The API for this purpose is quite powerful, allowing an application to wait for the first work item to complete, or for a specified set of the work items to complete. By combining methods from this API, it is possible to do things like "Here are 10 items to execute; for these 7 unimportant items, wait no more than 5 seconds, and for these 3 important items, wait no more than 30 seconds".

Example 24–11 Using a Work Manager

```
Work[] aWork = ...
Collection collBigItems = new ArrayList();
Collection collAllItems = new ArrayList();
for (int i = 0, c = aWork.length; i < c; ++i)
    {
    WorkItem item = manager.schedule(aWork[i]);

    if (i < 3)
        {
        // the first three work items are the important ones
        collBigItems.add(item);
        }

    collAllItems.add(item);
    }

Collection collDone = manager.waitForAll(collAllItems, 5000L);
if (!collDone.containsAll(collBigItems))
    {
    // wait the remainder of 30 seconds for the important work to finish
    manager.waitForAll(collBigItems, 25000L);
    }
```

25

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 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 enables you to make this change without having to modify all the existing 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 is 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` is registered during the very first `CacheFactory.getCache(...)` call for the corresponding cache. Example 25–1 assumes that the `createMapTrigger` method would return a new `MapTriggerListener(new MyCustomTrigger());`:

Example 25–1 Example MapTriggerListener Configuration

```
<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>
```

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 performs 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 25–2, `MapTriggerListener` is used to register the `PersonMapTrigger` with the `People` named cache.

Example 25–2 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 25–3 illustrates a map trigger and how it can be called. In the `PersonMapTrigger` class in Example 25–3, 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 25–3 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 25–4, calls the `PersonMapTrigger`. The new `MapTriggerListener` passes the `PersonMapTrigger` to the `People NamedCache`.

Example 25–4 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");
        }
    }
```

26

Using Coherence Query Language

This chapter describes how to use Coherence Query Language (CohQL) to interact with Coherence caches. CohQL is a light-weight syntax (in the tradition of SQL) that is used to perform cache operations on a Coherence cluster. The language can be used either programmatically or from a command-line tool.

The following sections are included in this chapter:

Note:

- Although the CohQL syntax may appear similar to SQL, it is important to remember that the syntax is *not* SQL and is actually more contextually related to the Java Persistence Query Language (JPQL) standard.
- CQL (Continuous Query Language) is a query language related to Complex Event Processing (CEP) and should not be confused with CohQL.

Understanding Coherence Query Language Syntax

The following sections describe the functionality provided by CohQL. Each section describes a particular statement, its syntax, and an example. You can find more query examples in "Additional Coherence Query Language Examples" on page 26-16.

Note: CohQL does not support subqueries.

The following topics are included in this section:

- Issuing Multiple Query Statements
- Viewing Query Cost and Effectiveness

For reference, Table 26–1 lists the Coherence query statements, clauses, and expressions in alphabetical order.

Table 26–1 Coherence Query Language Statements

For this statement, clause, or expression...	See this section
BACKUP CACHE	"Writing a Serialized Representation of a Cache to a File"
bind variables	"Using Bind Variables"
CREATE CACHE	"Creating a Cache"
CREATE INDEX	"Creating an Index on the Cache"
DELETE	"Deleting Entries in the Cache"
DROP CACHE	"Removing a Cache from the Cluster"
DROP INDEX	"Removing an Index from the Cache"
ENSURE CACHE	"Creating a Cache"
ENSURE INDEX	"Creating an Index on the Cache"
GROUP BY	"Aggregating Query Results"
INSERT	"Inserting Entries in the Cache"
key() pseudo-function	"Using Key and Value Pseudo-Functions"
path-expressions	"Using Path-Expressions"
RESTORE CACHE	"Loading Cache Contents from a File"
SELECT	"Retrieving Data from the Cache"
SOURCE	"Processing Query Statements in Batch Mode"
UPDATE	"Changing Existing Values"
value() pseudo-function	"Using Key and Value Pseudo-Functions"
WHERE	"Filtering Entries in a Result Set"

Query Syntax Basics

This section describes some building blocks of the syntax, such as path expressions, bind variables, and pseudo-functions.

Using Path-Expressions

One of the main building blocks of CohQL are path-expressions. Path expressions are used to navigate through a graph of object instances. An identifier in a path expression is used to represent a property in the Java Bean sense. It is backed by a `ReflectionExtractor` that is created by prepending a `get` and capitalizing the first letter. Elements are separated by the "dot" (`.`) character, that represents object traversal. For example the following path expression is used to navigate an object structure:

```
a.b.c
```

It reflectively invokes these methods:

```
getA().getB().getC()
```

Using Bind Variables

For programmatic uses, the API passes strings to a simple set of query functions. Use bind variables to pass the value of variables without engaging in string concatenation. There are two different formats for bind variables.

- the question mark (?)—Enter a question mark, immediately followed by a number to signify a positional place holder that indexes a collection of objects that are "supplied" before the query is run. The syntax for this form is: `?`*n* where *n* can be any number. Positional bind variables can be used by the `QueryHelper` class in the construction of filters. For example:

  ```
  QueryHelper.createFilter("number = ?1" , new Object[]{new Integer(42)};
  ```

- the colon (:)—Enter a colon, immediately followed by the identifier to be used as a named place holder for the object to be supplied as a key value pair. The syntax for this is `:`*identifier* where *identifier* is an alpha-numeric combination, starting with an alphabetic character. Named bind variables can be used by the `QueryHelper` class in the construction of filters. For example:

  ```
  HashMap env = new HashMap();
  env.put("iNum",new Integer(42));
  QueryHelper.createFilter("number = :iNum" , env};
  ```

See "Building Filters in Java Programs" on page 26-15 for more information on the `QueryHelper` class and constructing filters programmatically.

Using Key and Value Pseudo-Functions

CohQL provides a `key()` pseudo-function because many users store objects with a `key` property. The `key()` represents the cache's key. The query syntax also provides a `value()` pseudo-function. The `value()` is implicit in chains that do not start with `key()`. The `key()` and `value()` pseudo-functions are typically used in `WHERE` clauses, where they test against the key or value of the cache entry. For examples of using `key()` and `value()`, see "Key and Value Pseudo-Function Examples" on page 26-18 and "A Command-Line Example" on page 26-12.

Using Aliases

Although not needed semantically, CohQL supports aliases to make code artifacts as portable as possible to JPQL. CohQL supports aliases attached to the cache name and at the head of dotted path expressions in the `SELECT`, `UPDATE`, and `DELETE` commands. CohQL also allows the cache alias as a substitute for the `value()` pseudo function and as an argument to the `key()` pseudo function.

Using Quotes with Literal Arguments

Generally, you do not have to enclose literal arguments (such as *`cache-name`* or *`service-name`*) in quotes. Quotes (either single or double) would be required only if the argument contains an operator (such as `-`, `+`, `.`, `<`, `>`, `=`, and so on) or whitespace.

Filenames should also be quoted. Filenames often contain path separators (`/` or `\`) and dots to separate the name from the extension.

The compiler throws an error if it encounters an *unquoted* literal argument or filename that contains an offending character.

Retrieving Data

The following sections describe the SELECT statement and the WHERE clause. These entities are the basic building blocks of most cache queries.

Retrieving Data from the Cache

The SELECT statement is the basic building block of a query: it retrieves data from the cache. The clause can take several forms, including simple and complex path expressions, key expressions, transformation functions, multiple expressions, and aggregate functions. The SELECT statement also supports the use of aliases.

The form of the SELECT statement is as follows:

```
SELECT (properties* aggregators* | * | alias)
FROM "cache-name" [[AS] alias]
[WHERE conditional-expression] [GROUP [BY] properties+]
```

The asterisk (*) character represents the full object instead of subparts. It is not required to prefix a path with the *cache-name*. The FROM part of the SELECT statement targets the cache that forms the domain over which the query should draw its results. The *cache-name* is the name of an existing cache.

See "Simple SELECT * FROM Statements that Highlight Filters" on page 26-16 for additional examples.

Example:

- Select all of the items from the cache dept.

```
select * from "dept"
```

Filtering Entries in a Result Set

Use the WHERE clause to filter the entries returned in a result set. One of the key features of CohQL is that they can use path expressions to navigate object structure during expression evaluation. Conditional expressions can use a combination of logical operators, comparison expressions, primitive and function operators on fields, and so on.

In the literal syntax of the WHERE clause, use single quotes to enclose string literals; they can be escaped within a string by prefixing the quote with another single quote. Numeric expressions are defined according to the conventions of the Java programming language. Boolean values are represented by the literals TRUE and FALSE. Date literals are not supported.

Note: CohQL does not have access to type information. If a getter returns a numeric type different than the type of the literal, you may get a false where you would have expected a true on the comparison operators. The work around is to specify the type of the literal with l, for long, d for double, or s for short. The defaults are Integer for literals *without* a period (.) and Float for literals with a period (.).

Operator precedence within the WHERE clause is as follows:

1. Path operator (.)
2. Unary + and –

3. Multiplication (*) and division (/)
4. Addition (+) and subtraction (-)
5. Comparison operators: =, >, >=, <, <=, <>, [NOT] BETWEEN, [NOT] LIKE, [NOT] IN, IS [NOT] NULL, CONTAINS [ALL | ANY]
6. Logical operators (AND, OR, NOT)

The WHERE clause supports only arithmetic at the language level.

The BETWEEN operator can be used in conditional expressions to determine whether the result of an expression falls within an inclusive range of values. Numeric, or string expressions can be evaluated in this way. The form is: BETWEEN *lower* AND *upper*.

The LIKE operator can use the _ and % wildcards. The _ wildcard is used to match exactly one character, while the % wildcard is used to match zero or more occurrences of any characters. To escape the wildcards, precede them with an escape character that is defined using the escape keyword. The following example escapes the % wildcard using the \ escape character in order to select a key literally named k%1.

```
SELECT key(),value() FROM mycache WHERE key() LIKE "k\%1" escape "\"
```

In addition, any character may be defined as the escape character. For example:

```
SELECT key(),value() FROM mycache WHERE key() LIKE "k#%1" escape "#"
```

The IN operator can check whether a single-valued path-expression is a member of a collection. The collection is defined as an inline-list or expressed as a bind variable. The syntax of an inline-list is:

```
"(" literal* ")"
```

CONTAINS [ALL | ANY] are very useful operators because Coherence data models typically use de-normalized data. The CONTAINS operator can determine if a many-valued path-expression contains a given value. For example:

```
e.citys CONTAINS "Boston"
```

The ALL and ANY forms of CONTAINS take a inline-list or bind-variable with the same syntax as the IN operator.

> **Note:** Coherence provides a programmatic API that enables you to create standalone Coherence filters based on the WHERE clause conditional-expression syntax. See "Building Filters in Java Programs" on page 26-15.

See "Simple SELECT * FROM Statements that Highlight Filters" on page 26-16 for additional examples.

Example:

- Select all of the items in the cache dept where the value of the deptno key equals 10.

```
select * from "dept" where deptno = 10
```

Managing the Cache Lifecycle

The following sections describe how to create and remove caches. They also describe how to backup and restore cache contents.

Creating a Cache

Before sending queries, connect to an existing cache or create a new cache using the `CREATE CACHE` or `ENSURE CACHE` statements, respectively. This statement first attempts to connect to a cache with the specified *`cache-name`*. If the cache is not found in the cluster, Coherence attempts to create a cache with the specified name based on the current cache configuration file. This statement is especially useful on the command line. If you are using this statement in a program, you have the option of specifying service and classloader information instead of a name (classloaders cannot be accessed from the command line).

> **Note:** Cache names and service names must be enclosed in quotes (either double-quotes (`" "`) or single-quotes (`' '`)) in a statement.

The syntax is:

```
[ CREATE | ENSURE ] CACHE "cache-name"
[ SERVICE "service-name" ]
```

Example:

- Create a cache named `dept`.

```
create cache "dept"
```

Writing a Serialized Representation of a Cache to a File

Use the `BACKUP CACHE` statement to write a serialized representation of the given cache to a file represented by the given *`filename`*. The *`filename`* is an operating system-dependent path and must be enclosed in single or double quotes. The `BACKUP CACHE` statement is available only in the command-line tool. The syntax is:

```
BACKUP CACHE "cache-name" [ TO ] [ FILE ] "filename"
```

> **Note:** The backup (and subsequent restore) functionality is designed for use in a development and testing environment and should not be used on a production data set as it makes no provisions to ensure data consistency. It is not supported as a production backup, snapshot, or checkpointing utility.
>
> In particular:
>
> - The backup is slow since it only operates on a single node in the cluster.
> - The backup is not atomic. That is, it misses changes to elements which occur during the backup and results in a dirty read of the data.
> - The backup stops if an error occurs and results in an incomplete backup. In such scenarios, an `IOException` is thrown that describes the error.

Example:

- Write a serialized representation of the cache `dept` to the file `textfile`.

```
backup cache "dept" to file "textfile"
```

Loading Cache Contents from a File

Use the RESTORE CACHE statement to read a serialized representation of the given cache from a file represented by the given *filename*. The *filename* is an operating system-dependent path and must be enclosed in single or double quotes. The RESTORE CACHE statement is available only in the command-line tool. The syntax is:

```
RESTORE CACHE "cache-name" [ FROM ] [ FILE ] "filename"
```

Example:

- Restore the cache dept from the file textfile.

```
restore cache "dept" from file "textfile"
```

Removing a Cache from the Cluster

Use the DROP CACHE statement to remove the specified cache completely from the cluster. The cache is removed by a call to the Java destroy() method. If any cluster member holds a reference to the dropped cache and tries to perform any operations on it, then the member receives an IllegalStateException. The syntax for the Coherence query DROP CACHE statement is:

```
DROP CACHE  "cache-name"
```

Example:

- Remove the cache orders from the cluster.

```
drop cache "orders"
```

Working with Cache Data

The following sections describe how to work with data in the cache, such as inserting and deleting cache data and filtering result sets.

Aggregating Query Results

An aggregate query is a variation on the SELECT query. Use an aggregate query when you want to group results and apply aggregate functions to obtain summary information about the results. A query is considered an aggregate query if it uses an aggregate function or has a GROUP BY clause. The most typical form of an aggregate query involves the use of one or more grouping expressions followed by aggregate functions in the SELECT clause paired with the same lead grouping expressions in a GROUP BY clause.

CohQL supports these aggregate functions: COUNT, AVG, MIN, MAX, and SUM.

See "Complex Queries that Feature Projection, Aggregation, and Grouping" on page 26-17 for additional examples.

Example:

- Select the total amount and average price for items from the orders cache, grouped by supplier.

```
select supplier,sum(amount),avg(price) from "orders" group by supplier
```

Changing Existing Values

Use the UPDATE statement to change an existing value in the cache. The syntax is:

```
UPDATE "cache-name" [[AS] alias]
SET update-statement {, update-statement}*
```

```
[ WHERE conditional-expression ]
```

Each *update-statement* consists of a path expression, assignment operator (=), and an expression. The expression choices for the assignment statement are restricted. The right side of the assignment must resolve to a literal, a bind-variable, a static method, or a new Java-constructor with only literals or bind-variables. The UPDATE statement also supports the use of aliases.

See "UPDATE Examples" on page 26-18 for additional examples.

Example:

- For employees in the employees cache whose ranking is above grade 7, update their salaries to 1000 and vacation hours to 200.

```
update "employees" set salary = 1000, vacation = 200 where grade > 7
```

Inserting Entries in the Cache

Use the INSERT statement to store the given VALUE under the given KEY. If the KEY clause is not provided, then the newly created object is sent the message getKey(), if possible. Otherwise, the value object is used as the key.

Note that the INSERT statement operates on Maps of Objects. The syntax is:

```
INSERT INTO "cache-name"
[ KEY (literal | new java-constructor | static method) ]
VALUE (literal | new java-constructor | static method)
```

Example:

- Insert the key writer with the value David into the employee cache.

```
insert into "employee" key "writer" value "David"
```

Deleting Entries in the Cache

Use the DELETE statement to delete specified entries in the cache. The syntax is:

```
DELETE FROM "cache-name" [[AS] alias]
[WHERE conditional-expression]
```

The WHERE clause for the DELETE statement functions the same as it would for a SELECT statement. All *conditional-expressions* are available to filter the set of entities to be removed. The DELETE statement also supports the use of aliases.

> **Be Careful:** If the WHERE clause is not present, then all entities in the given cache are removed.

Example:

- Delete the entry from the cache employee where bar.writer key is not David.

```
delete from "employee" where bar.writer IS NOT "David"
```

Working with Indexes

The following sections describe how to create and remove indexes on cache data. Indexes are a powerful tool that allows Coherence's built-in optimizer to more quickly and efficiently analyze queries and return results.

Creating an Index on the Cache

Use the `CREATE INDEX` or the `ENSURE INDEX` statement to create indexes on an identified cache. The syntax is:

```
[ CREATE | ENSURE ] INDEX [ON] "cache-name" (value-extractor-list)
```

The *`value-extractor-list`* is a comma-delimited list that uses path expressions to create `ValueExtractors`. If multiple elements exist, then a `MultiExtractor` is used. To create a `KeyExtractor`, then start the path expression with a `key()` pseudo-function.

Natural ordering for the index is assumed.

Example:

- Create a index on the attribute `lastname` in the `orders` cache.

```
create index "orders" lastname
```

Removing an Index from the Cache

The `DROP INDEX` statement removes the index based on the given `ValueExtractor`. This statement is available only for the command-line tool. The syntax is:

```
DROP INDEX [ON] "cache-name" (value-extractor-list)
```

Example:

- Remove the index on the `lastname` attribute in the `orders` cache.

```
drop index "orders" lastname
```

Issuing Multiple Query Statements

The following section describes how to more efficiently issue multiple query statements to the cache.

Processing Query Statements in Batch Mode

The `SOURCE` statement allows for the "batch" processing of statements. The `SOURCE` statement opens and reads one or more query statements from a file represented by the given *`filename`*. The *`filename`* is an operating system-dependent path and must be enclosed in single or double quotes. Each query statement in the file must be separated by a semicolon (`;`) character. Sourcing is available only in the command-line tool, where you naturally want to load files consisting of sequences of commands. Source files may source other files. The syntax is:

```
SOURCE FROM [ FILE ] "filename"
```

`SOURCE` can be abbreviated with an "at" symbol (`@`) as in `@"`*`filename`*". On the command command line only, a "period" symbol '.' can be used as an abbreviation for '@' but must no contain quotes around the filename.

Example:

- Process the statements in the file `command_file`.

```
source from file "command_file"
```

or,

```
@ "command_file"
```

or,

```
. command_file
```

Viewing Query Cost and Effectiveness

The `EXPLAIN PLAN FOR` and `TRACE` commands are used to create and output query records that are used to determine the cost and effectiveness of a query. A query explain record provides the estimated cost of evaluating a filter as part of a query operation. A query trace record provides the actual cost of evaluating a filter as part of a query operation. Both query records take into account whether or not an index can be used by a filter. See "Interpreting Query Records" on page 22-9 for additional details on understanding the data provided in an explain plan record and trace record. The syntax for the commands are:

Query Explain Plan:

```
EXPLAIN PLAN FOR select statement | update statement | delete statement
```

Trace:

```
TRACE select statement | update statement | delete statement
```

Example:

```
EXPLAIN PLAN FOR select * from "mycache" where age=19 and firstName=Bob
```

or,

```
TRACE SELECT * from "MyCache" WHERE age=19
```

Using the CohQL Command-Line Tool

The CohQL command-line tool provides a non-programmatic way to interact with caches by allowing statements to be issued from the command line. The tool can be run using the `com.tangosol.coherence.dslquery.QueryPlus` class or, for convenience, a startup script is available to run the tool and is located in the *COHERENCE_HOME*`/bin/` directory. The script is available for both Windows (`query.cmd`) and UNIX (`query.sh`).

The script starts a cluster node in console mode; that is, storage is not enabled on the node. This is the suggested setting for production environments and assumes that the node joins a cluster that contains storage-enabled cache servers. However, a storage-enabled node can be created for testing by changing the `storage_enabled` setting in the script to `true`.

Note: As configured, the startup script uses the default operational configuration file (`tangosol-coherence.xml`) and the default cache configuration file (`coherence-cache-config.xml`) that are located in the `coherence.jar` when creating/joining a cluster and configuring caches. For more information on configuring Coherence, see Chapter 3, "Understanding Configuration."

The script provides the option for setting the COHERENCE_HOME environment variable. If COHERENCE_HOME is not set on the computer, set it in the script to the location where Coherence was installed.

CohQL uses JLine for enhanced command-line editing capabilities, such as having the up and down arrows move through the command history. However, JLine is not required to use CohQL. The script automatically uses the jline.jar library that is located in the *COHERENCE_HOME*/lib/ directory. A different location can be specified by modifying the JLINE_HOME variable and classpath in the script. If the JLine library is not found, a message displays and CohQL starts without JLine capabilities.

Starting the Command-line Tool

The following procedure demonstrates how to start the CohQL command-line tool using the startup script and assumes that the storage_enabled setting in the script is set to false (the default):

1. Start a cache server cluster node or ensure that an existing cache server cluster node is started.

   To start a cache server cluster node, open a command prompt or shell and execute the cache server startup script that is located in the /bin directory: cache-server.cmd on the Windows platform or cache-server.sh for UNIX platforms. The cache server starts and output is emitted that provides information about this cluster member.

2. Open a command prompt or shell and execute the CohQL command-line startup script that is located in the /bin directory: query.cmd on the Windows platform or query.sh for UNIX platforms. Information about the Java environment displays. The command-line tool prompt (CohQL>) is returned.

   Note: When joining an existing cache server node, modify the startup script to use the same cluster settings as the existing cache server node, including the same cache configuration.

3. Enter help at the prompt to view the complete command-line help. Enter commands to list the help without detailed descriptions.

   See "A Command-Line Example" on page 26-12 for a series of query statements that exercise the command-line tool.

Using Command-Line Tool Arguments

The CohQL command-line tool includes a set of arguments that are read and executed before the CohQL> prompt returns. This is useful when using the script as part of a larger script– for example, as part of a build process or to pipe I/O. Enter help at the CohQL> prompt to view help for the arguments within the command-line tool.

Table 26–2 Coherence Query Language Command-Line Tool Arguments

Argument	Description
-t	enable trace mode to print debug information.
-c	Exit the command-line tool after processing the command-line arguments. This argument should not be used when redirecting from standard input; in which case, the tool exits as soon as the command line arguments are finished being processed and the redirected input is never read.

Table 26–2 (Cont.) Coherence Query Language Command-Line Tool Arguments

Argument	Description
`-s`	Run the command-line tool in silent mode to remove extraneous verbiage. This allows the command line tool to be used in pipes or filters by redirecting standard input (`<myInput`) and standard output (`>myOuput`).
`-e`	Run the command-line tool in extended language mode. This mode allows object literals in update and insert commands. See the command-line help for complete usage information.
`-l` *statement*	Execute the given statement. Statements must be enclosed in single or double quotes. Any number of `-l` arguments can be used.
`-f` *filename*	Process the statements in the given file. The statements in the file must be separated by a semicolon (;). The file is an operating system-dependent path and must be enclosed in single or double quotes. Any number of `-f` arguments can be used.

Examples

Return all entries in the `contact` cache and print the entries to the standard out then exit the command-line tool.

```
query.sh -c -l "select * from contact"
```

Return all entries in the `dist-example` cache and print the entries (suppressing extra verbiage) to the file named `myOutput` then exit the command-line tool.

```
query.cmd -s -c -l "select * from 'dist-example'" >myOutput
```

Process all the segments in the file named `myStatements` then exit the command-line tool.

```
query.sh -c -f myStatements
```

Read the commands from the `myInput` file and print the output (suppressing extra verbiage) to the file named `myOutput`.

```
query.sh -s <myInput >myOutput
```

A Command-Line Example

Example 26–1 illustrates a simple example that exercises the command-line tool on Windows. This example is intended to test statements against a local cache, so the `storage_enabled` setting in the startup script is set to `true`. The example illustrates creating and dropping a cache, storing and retrieving entries, and restoring the cache from a backup file. It also highlights the use of the `key()` and `value()` pseudo-functions.

When you start `query.cmd` at the command prompt, information about the Java environment, the Coherence version and edition, and Coherence cache server is displayed. You then receive a prompt (`CohQL>`) where you can enter your query statements.

Annotations that describe the commands and their output have been added to the example in bold-italic font. Here is an example:

```
< This is an annotation. >
```

Example 26–1 A Command-Line Query Exercise

```
C:/coherence/bin/query.cmd
```

```
** Starting storage enabled console **
java version "1.6.0_14"
Java(TM) SE Runtime Environment (build 1.6.0_14-b08)
Java HotSpot(TM) Server VM (build 14.0-b16, mixed mode)

2010-01-27 16:54:07.501/0.265 Oracle Coherence 3.6.0.0 Internal <Info> (thread=main, member=n/a):
Loaded operational configuration from
"jar:file:/C:/coherence360/coherence/lib/coherence.jar!/tangosol-coherence.xml"
2010-01-27 16:54:07.501/0.265 Oracle Coherence 3.6.0.0 Internal <Info> (thread=main, member=n/a):
Loaded operational overrides from
"jar:file:/C:/coherence360/coherence/lib/coherence.jar!/tangosol-coherence-override-dev.xml"
2010-01-27 16:54:07.501/0.265 Oracle Coherence 3.6.0.0 Internal <D5> (thread=main, member=n/a):
Optional configuration override "/tangosol-coherence-override.xml" is not specified
2010-01-27 16:54:07.517/0.281 Oracle Coherence 3.6.0.0 Internal <D5> (thread=main, member=n/a):
Optional configuration override "/custom-mbeans.xml" is not specified

Oracle Coherence Version 3.6.0.0 Internal Build 0
 Grid Edition: Development mode
Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.

2010-01-27 16:54:09.173/1.937 Oracle Coherence GE 3.6.0.0 Internal <D5> (thread=Cluster,
member=n/a): Service Cluster joined the cluster with senior service member n/a
2010-01-27 16:54:12.423/5.187 Oracle Coherence GE 3.6.0.0 Internal <Info> (thread=Cluster,
member=n/a): Created a new cluster "cluster:0xC4DB" with Member(Id=1, Timestamp=2010-01-27
16:54:08.032, Address=130.35.99.213:8088, MachineId=49877,
Location=site:us.oracle.com,machine:tpfaeffl-lap7,process:4316, Role=TangosolCoherenceQueryPlus,
Edition=Grid Edition, Mode=Development, CpuCount=2, SocketCount=1)
UID=0x822363D500000126726BBBA0C2D51F98
2010-01-27 16:54:12.501/5.265 Oracle Coherence GE 3.6.0.0 Internal <D5>
(thread=Invocation:Management, member=1): Service Management joined the cluster with senior service
member 1

   < Create a cache named "employees". >
CohQL> create cache "employees"

2010-01-27 16:54:26.892/19.656 Oracle Coherence GE 3.6.0.0 Internal <Info> (thread=main, member=1):
Loaded cache configuration from
"jar:file:/C:/coherence360/coherence/lib/coherence.jar!/coherence-cache-config.xml"
2010-01-27 16:54:27.079/19.843 Oracle Coherence GE 3.6.0.0 Internal <D5> (thread=DistributedCache,
member=1): Service DistributedCache joined the cluster with senior service member 1
2010-01-27 16:54:27.095/19.859 Oracle Coherence GE 3.6.0.0 Internal <D5> (thread=DistributedCache,
member=1): Service DistributedCache: sending Config
Sync to all
Result

   < Insert an entry (key-value pair) into the cache. >
CohQL> insert into "employees" key "David" value "ID-5070"

   < Insert an object into the cache. >
CohQL> insert into "employess" value new com.my.Employee("John", "Doe", "address", 34)

   < Change the value of the key. >
CohQL> update employees set value() = "ID-5080" where key() like "David"
Result
David, true

   < Retrieve the values in the cache. >
CohQL> select * from "employees"
Result
ID-5080
```

```
   < Retrieve the value of a key that does not exist. An empty result set is returned >
CohQL> select key(), value() from "employees" where key() is "Richard"
Result

   < Delete an existing key in the cache. An empty result set is returned. >
CohQL> delete from employees where key() = "David"
Result

  < Delete the contents of the employees cache. An empty result set is returned. >
CohQL> delete from "employees"
Result

   < Destroy the employees cache. >
CohQL> drop cache "employees"

   < Re-create the employees cache. >
CohQL> create cache "employees"
Result

   < Insert more entries into the cache. >
CohQL> insert into "employees" key "David" value "ID-5080"

CohQL> insert into "employees" key "Julie" value "ID-5081"

CohQL> insert into "employees" key "Mike" value "ID-5082"

CohQL> insert into "employees" key "Mary" value "ID-5083"

   < Retrieve the keys and value in the enployees cache. >
CohQL> select key(), value() from "employees"
Result
Julie, ID-5081
Mike, ID-5082
Mary, ID-5083
David, ID-5080

   < Save a serialized representation of the cache in a file. >
CohQL> backup cache "employees" to "emp.bkup"

   < Delete a key from the cache. >
CohQL> delete from "employees" where key() = "David"
Result

   < Retreive the cache contents again, notice that the deleted key and value are not present. >
CohQL> select key(), value() from "employees"
Result
Julie, ID-5081
Mike, ID-5082
Mary, ID-5083

   < Delete the contents of the cache. >
CohQL> delete from "employees"
Result

   < Retrieve the contents of the cache. An empty result set is returned. >
CohQL> select * from "employees"
Result

   < Restore the cache contents from the backup file. >
```

```
CohQL> restore cache "employees" from file "emp.bkup"

   < Retrieve the cache contents. Note that all of the entries have been restored and returned. >
CohQL> select key(), value() from "employees"
Result
Julie, ID-5081
Mike, ID-5082
Mary, ID-5083
David, ID-5080

   < Destroy the employees cache. >
CohQL> drop cache "employees"

   < Exit the command-line tool. >
CohQL> bye
```

Building Filters in Java Programs

The `FilterBuilder` API is a string-oriented way to filter a result set from within a Java program, without having to remember details of the Coherence API. The API provides a set of four overloaded `createFilter` factory methods in the `com.tangosol.util.QueryHelper` class.

The following list describes the different forms of the `createFilter` method. The passed string uses the Coherence query `WHERE` clause syntax (as described in "Filtering Entries in a Result Set" on page 26-4), but without the literal `WHERE`. The forms that take an `Object` array or `Map` are for passing objects that are referenced by bind variables. Each form constructs a filter from the provided Coherence query string.

- `public static Filter createFilter(String s)`—where `s` is a `String` in the Coherence query representing a `Filter`.
- `public static Filter createFilter(String s, Object[] aBindings)`—where `s` is a `String` in the Coherence query representing a `Filter` and `aBindings` is an array of `Objects` to use for bind variables.
- `public static Filter createFilter(String s, Map bindings)`—where `s` is a `String` in the Coherence query representing a `Filter` and `bindings` is a `Map` of `Objects` to use for bind variables.
- `public static Filter createFilter(String s, Object[] aBindings, Map bindings)`—where `s` is a `String` in the Coherence query representing a `Filter`, `aBindings` is an array of `Objects` to use for bind variables, and `bindings` is a `Map` of `Objects` to use for bind variables.

These factory methods throw a `FilterBuildingException` if there are any malformed, syntactically incorrect expressions, or semantic errors. Since this exception is a subclass of `RuntimeException`, catching the error is not required, but the process could terminate if you do not.

Example

The following statement uses the `createFilter(String s)` form of the method. It constructs a filter for employees who live in Massachusetts but work in another state.

```
..
QueryHelper.createFilter("homeAddress.state = 'MA'  and workAddress.state !=
'MA'")
...
```

This statement is equivalent to the following filter/extractor using the Coherence API:

```
AndFilter(EqualsFilter(ChainedExtractor(#getHomeAddress[], #getState[]), MA),
NotEqualsFilter(ChainedExtractor(#getWorkAddress[], #getState[]), MA)))
```

The `QueryHelper` class also provides a `createExtractor` method that enables you to create value extractors when building filters. The extractor is used to both extract values (for example, for sorting or filtering) from an object, and to provide an identity for that extraction. The following example demonstrates using `createExtractor` when creating an index:

```
cache.addIndex(QueryHelper.createExtractor("key().lastName"),/*fOrdered*/ true,
  /*comparator*/ null);
```

Additional Coherence Query Language Examples

This section provides additional examples and shows their equivalent Coherence API calls with instantiated Objects (`Filters`, `ValueExtractors`, `Aggregators`, and so on). The simple `select *` examples that highlight `Filters` can understand the translation for `FilterBuilder` API if you focus only on the `Filter` part. Use the full set of examples to understand the translation for the `QueryBuilder` API and the command-line tool.

The examples use an abbreviated form of the path syntax where the cache name to qualify an identifier is dropped.

The Java language form of the examples also use `ReducerAggregator` instead of `EntryProcessors` for projection. Note also that the use of `KeyExtractor` should no longer be needed given changes to `ReflectionExtractor` in Coherence 3.5.

Simple SELECT * FROM Statements that Highlight Filters

- Select the items from the cache `orders` where `40` is greater than the value of the `price` key.

  ```
  select * from "orders" where 40 > price
  ```

- Select the items from the cache `orders` where the value of the `price` key exactly equals 100, and the value of `insurance` key is less than 10 or the value of the `shipping` key is greater than or equal to 20.

  ```
  select * from "orders" where price is 100 and insurance < 10 or shipping >= 20
  ```

- Select the items from the cache `orders` where the value of the `price` key exactly equals 100, and *either* the value of `insurance` key is less than 10 or the value of the `shipping` key is greater than or equal to 20.

  ```
  select * from "orders" where price is 100 and (insurance < 10 or shipping >=
  20)
  ```

- Select the items from the cache `orders` where either the value of the `price` key equals `100`, or the `bar` key equals `20`.

  ```
  select * from "orders" where price = 100 or shipping = 20
  ```

- Select the items from the cache `orders` where the value of the `insurance` key is not null.

  ```
  select * from "orders" where insurance is not null
  ```

- Select the items from the cache `employees` where the `emp_id` key has a value between 1 and 1000 or the `bar.emp` key is not "`Smith`".

```
select * from "employees" where emp_id between 1 and 1000 or bar.emp is not
"Smith"
```

- Select items from the cache `orders` where the value of `item` key is similar to the value "`coat`".

```
select * from "orders" where item like "coat%"
```

- Select items from the cache `employees` where the value of `emp_id` is in the set 5, 10, 15, or 20.

```
select * from "employees" where emp_id in (5,10,15,20)
```

- Select items from the cache `employees` where `emp_id` contains the list `5`, `10`, `15`, and `20`.

```
select * from "employees" where emp_id contains (5,10,15,20)
```

- Select items from the cache `employees` where `emp_id` contains the all of the items `5`, `10`, `15`, and `20`.

```
select * from "employees" where emp_id contains all (5,10,15,20)
```

- Select items from the cache `employees` where `emp_id` contains any of the items `5`, `10`, `15`, or `20`.

```
select * from "employees" where emp_id contains any (5,10,15,20)
```

- Select items from cache `employees` where the value of `foo` key is less than 10 and occurs in the set 10, 20.

```
select * from "employees" where emp_id < 10 in (10,20)
```

Complex Queries that Feature Projection, Aggregation, and Grouping

- Select the home `state` and `age` of employees in the cache `ContactInfoCache`, and group by `state` and `age`.

```
select homeAddress.state, age, count() from "ContactInfoCache" group by
homeAddress.state, age
```

- Select the spurious `frobit` key from the `orders` cache. Note, an empty result set is returned.

```
select frobit,supplier,sum(amount),avg(price) from "orders" group by supplier
```

- For the items in the `orders` cache that are greater than $1000, select the items, their prices and colors.

```
select item_name,price,color from "orders" where price > 1000
```

- Select the total `amount` for items from the `orders` cache.

```
select sum(amount) from "orders"
```

- Select the total `amount` for items from the `orders` cache where the `color` attribute is `red` or `green`.

```
select sum(amount) from "orders" where color is "red" or color is "green"
```

- Select the total `amount` and average `price` for items from the `orders` cache

```
select sum(amount),avg(price) from "orders"
```

- Select one copy of the `lastname` and `city` from possible duplicate rows from the `employees` cache, where the `state` is California.

```
select distinct lastName,city from "employees" where state = "CA"
```

UPDATE Examples

- For employees in the `employees` cache whose ranking is above grade 7, increase their salaries by 10% and add 50 hours of vacation time.

```
update "employees" set salary = salary*1.10, vacation = vacation + 50 where
grade > 7
```

Key and Value Pseudo-Function Examples

This section provides examples of how to use the `key()` and `value()` pseudo-functions. For additional examples, see "A Command-Line Example" on page 26-12.

- Select the employees from the `ContactInfoCache` whose home address is in Massachusetts, but work out of state.

```
select key().firstName, key().lastName from "ContactInfoCache"
homeAddress.state is 'MA' and workAddress.state != "MA"
```

- Select the employees from the `ContactInfoCache` cache whose age is greater than 42.

```
select key().firstName, key().lastName, age from "ContactInfoCache" where age >
42
```

27

Performing Transactions

This chapter provides instructions for using Coherence's transaction and data concurrency features. Users should be familiar with transaction principles before reading this chapter. In addition, the Coherence Resource Adapter requires knowledge of J2EE Connector Architecture (J2CA), Java Transaction API (JTA) and Java EE deployment.

The following sections are included in this chapter:

Overview of Transactions

Transactions ensure correct outcomes in systems that undergo state changes by allowing a programmer to scope multiple state changes into a unit of work. The state changes are committed only if each change can complete without failure; otherwise, all changes must be rolled back to their previous state.

Transactions attempt to maintain a set of criteria that are commonly referred to as ACID properties (Atomicity, Consistency, Isolation, Durability):

- **Atomic** - The changes that are performed within the transaction are either all committed or all rolled back to their previous state.
- **Consistent** - The results of a transaction must leave any shared resources in a valid state.
- **Isolated** - The results of a transaction are not visible outside of the transaction until the transaction has been committed.
- **Durable** - The changes that are performed within the transaction are made permanent.

Sometimes ACID properties cannot be maintained solely by the transaction infrastructure and may require customized business logic. For instance, the consistency property requires program logic to check whether changes to a system are valid. In addition, strict adherence to the ACID properties can directly affect infrastructure and application performance and must be carefully considered.

Coherence offers various transaction options that provide different transaction guarantees. The options should be selected based on an application's or solution's transaction requirements.

Table 27–1 summarizes the various transactions option that Coherence offers.

Table 27–1 Coherence Transaction Options

Option Name	Description
Explicit locking	The `ConcurrentMap` interface (which is extended by the `NamedCache` interface) supports explicit locking operations. The locking API guarantees data concurrency but does not offer atomic guarantees. For detailed information on this option, see "Using Explicit Locking for Data Concurrency" on page 27-2.
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. This option offers high-level concurrency control but does not offer atomic guarantees. For detailed information on this option, see "Using Entry Processors for Data Concurrency" on page 27-3.
Transaction Framework API	Coherence Transaction Framework API is a connection-based API that provides atomic transaction guarantees across partitions and caches even with a client failure. The framework supports the use of `NamedCache` operations, queries, aggregation, and entry processors within the context of a transaction. For detailed information on this option, see "Using the Transaction Framework API" on page 27-5.
Coherence Resource Adapter	The Coherence resource adapter leverages the Coherence Transaction Framework API and allows Coherence to participate as a resource in XA transactions that are managed by a JavaEE container's transaction manager. This transaction option offers atomic guarantees. For detailed information on this option, see "Using the Coherence Resource Adapter" on page 27-21.

Using Explicit Locking for Data Concurrency

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 27–1 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
    // do not 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 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 blocks indefinitely until a lock can be obtained, 0 returns immediately, and a value greater than 0 waits 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 value is zero. 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".

Using Entry Processors for Data Concurrency

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 `ConcurrentMap` explicit locking API, `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 different than the explicit `lock(key)` functionality provided by `ConcurrentMap` API.

`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` is executed against the cache entries. Each entry that matches the `Filter` criteria has the `EntryProcessor` executed against it. For more information on Filters, see Chapter 22, "Querying Data In a Cache".

Entry processors are 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 computers (as opposed to serially on one computer) and the network overhead of obtaining and releasing locks is eliminated.

> **Note:** EntryProcessor classes must be available in the classpath for each cluster node.

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

Example 27–2 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 27–3 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()` do not execute as one atomic unit. As soon as an individual `EntryProcessor` has completed, any updates made to the cache is immediately visible while the other `EntryProcessors` are executing. Furthermore, an uncaught exception in an `EntryProcessor` does not prevent the others from executing. Should the primary node for an entry fail while executing an `EntryProcessor`, the backup node performs the execution instead. However if the node fails after the completion of an `EntryProcessor`, the `EntryProcessor` is not 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"*.

Using the Transaction Framework API

The Transaction Framework API allows TCMP clients to perform operations and use queries, aggregators, and entry processors within the context of a transaction. The transactions provide read consistency and atomic guarantees across partitions and caches even with client failure. The framework uses its own concurrency strategy and storage implementation and its own recovery manager for failed transactions.

Note: The `TransactionMap` API has been deprecated and is superseded by the Transaction Framework API. The two APIs are mutually exclusive.

Known Limitations

The Transaction Framework API has the following limitations:

- Database Integration – For existing Coherence users, the most noticeable limitation is the lack of support for database integration as compared to the existing Partitioned `NamedCache` implementation.
- Server-Side Functionality – Transactional caches do not support eviction or expiry, though they support garbage collection of older object versions. Backing map listeners, triggers, and `CacheStore` modules are not supported.

- Explicit Locking and Pessimistic Transactions – Pessimistic/explicit locking (`ConcurrentMap` interface) are not supported.
- Filters – Filters, such as `PartitionedFilter`, `LimitFilter` and `KeyAssociationFilter`, are not supported.
- Synchronous Listener – The `SynchronousListener` interface is not supported.
- Near Cache – Wrapping a near cache around a transactional cache is not supported.
- Key Partitioning Strategy – You cannot specify a custom `KeyPartitioningStrategy` for a transactional cache; although, `KeyAssociation` or a custom `KeyAssociator` works.

The following topics are included in this section:

The Transaction Framework API is also the underling transaction framework for the Coherence JCA resource adapter. For details on using the resource adapter, see "Using the Coherence Resource Adapter" on page 27-21.

Defining Transactional Caches

Transactional caches are specialized distributed caches that provide transactional guarantees. Transactional caches are required whenever performing a transaction using the Transaction Framework API. Transactional caches are not interoperable with non-transactional caches.

At run-time, transactional caches are automatically used with a set of internal transactional caches that provide transactional storage and recovery. Transactional caches also allow default transaction behavior (including the default behavior of the internal transactional caches) to be overridden at run-time.

Transactional caches are defined within a cache configuration file using a `<transactional-scheme>` element. A transaction scheme includes many of the same elements and attributes that are available to a distributed cache scheme. For detailed information about the `<transactional-scheme>` element and all its subelements, see "transactional-scheme" on page B-110.

Note: The use of transaction schemes within near cache schemes is currently not supported.

The following example demonstrates defining a transactional cache scheme in a cache configuration file. The cache is named `MyTxCache` and maps to a

<transactional-scheme> that is named example-transactional. The cache name can also use the tx-* convention which allows multiple cache instances to use a single mapping to a transactional cache scheme.

> **Note:** The <service-name> element, as shown in the example below, is optional. If no <service-name> element is included in the transactional cache scheme, TransactionalCache is used as the default service name. In this case, applications must connect to a transactional service using the default service name. See "Creating Transactional Connections" on page 27-10.

Example 27–4 Example Transactional Cache Definition

```
<caching-scheme-mapping>
   <cache-mapping>
      <cache-name>MyTxCache</cache-name>
      <scheme-name>example-transactional</scheme-name>
   </cache-mapping>
</caching-scheme-mapping>

<caching-schemes>
<!-- Transactional caching scheme. -->
   <transactional-scheme>
      <scheme-name>example-transactional</scheme-name>
      <service-name>TransactionalCache</service-name>
      <thread-count>10</thread-count>
      <request-timeout>30000</request-timeout>
      <autostart>true</autostart>
   </transactional-scheme>
</caching-schemes>
```

The <transactional-scheme> element also supports the use of scheme references. In the below example, a <transactional-scheme> with the name example-transactional references a <transactional-scheme> with the name base-transactional:

```
<caching-scheme-mapping>
   <cache-mapping>
      <cache-name>tx-*</cache-name>
      <scheme-name>example-transactional</scheme-name>
   </cache-mapping>
</caching-scheme-mapping>

<caching-schemes>
   <transactional-scheme>
      <scheme-name>example-transactional</scheme-name>
      <scheme-ref>base-transactional</scheme-ref>
      <thread-count>10</thread-count>
   </transactional-scheme>

   <transactional-scheme>
      <scheme-name>base-transactional</scheme-name>
      <service-name>TransactionalCache</service-name>
      <request-timeout>30000</request-timeout>
      <autostart>true</autostart>
   </transactional-scheme>
</caching-schemes>
```

Performing Cache Operations within a Transaction

Applications perform cache operations within a transaction in one of three ways:

- Using the NamedCache API – Applications use the `NamedCache` API to implicitly perform cache operations within a transaction.
- Using the Connection API – Applications use the `Connection` API to explicitly perform cache operations within a transaction.
- Using the Coherence Resource Adapter – Java EE applications use the Coherence Resource Adapter to connect to a Coherence data cluster and perform cache operations as part of a distributed (global) transaction.

Using the NamedCache API

The `NamedCache` API can perform cache operations implicitly within the context of a transaction. However, this approach does not allow an application to change default transaction behavior. For example, transactions are in auto-commit mode when using the `NamedCache` API approach. Each operation is immediately committed when it successfully completes; multiple operations cannot be scoped into a single transaction. Applications that require more control over transactional behavior must use the `Connection` API. See "Using Transactional Connections" on page 27-11 for a detailed description of a transaction's default behaviors.

The `NamedCache` API approach is ideally suited for ensuring atomicity guarantees when performing single operations such as `putAll`. The following example demonstrates a simple client that creates a `NamedCache` instance and uses the `CacheFactory.getCache()` method to get a transactional cache. The example uses the transactional cache that was defined in Example 27–4. The client performs a `putAll` operation that is only committed if all the `put` operations succeed. The transaction is automatically rolled back if any `put` operation fails.

```
...
String key = "k";
String key2 = "k2";
String key3 = "k3";
String key4 = "k4";

CacheFactory.ensureCluster();
NamedCache cache = CacheFactory.getCache("MyTxCache");

Map map = new HashMap();
map.put(key, "value");
map.put(key2, "value2");
map.put(key3, "value3");
map.put(key4, "value4");

//operations performed on the cache are atomic
cache.putAll(map);

CacheFactory.shutdown();
...
```

Using the Connection API

The `Connection` API is used to perform cache operations within a transaction and provides the ability to explicitly control transaction behavior. For example, applications can enable or disable auto-commit mode or change transaction isolation levels.

The examples in this section demonstrate how to use the `Connection` interface, `DefaultConnectionFactory` class, and the `OptimisticNamedCache` interface which are located in the `com.tangosol.coherence.transaction` package. The examples use the transactional cache that was defined in Example 27–4. The `Connection` API is discussed in detail following the examples.

Example 27–5 demonstrates an auto-commit transaction; where, two `insert` operations are each executed as separate transactions.

Example 27–5 Performing an Auto-Commit Transaction

```
...
Connection con = new DefaultConnectionFactory().
   createConnection("TransactionalCache");

OptimisticNamedCache cache = con.getNamedCache("MytxCache");

cache.insert(key, value);
cache.insert(key2, value2);

con.close();
...
```

Example 27–6 demonstrates a non auto-commit transaction; where, two `insert` operations are performed within a single transaction. Applications that use non auto-commit transactions must manually demarcate transaction boundaries.

Example 27–6 Performing a Non Auto-Commit Transaction

```
...
Connection con = new DefaultConnectionFactory().
   createConnection("TransactionalCache");

con.setAutoCommit(false);

try
   {
   OptimisticNamedCache cache = con.getNamedCache("MyTxCache");

   cache.insert(key, value);
   cache.insert(key2, value2);
   con.commit();

catch (Exception e)
   {
   con.rollback();
   throw e;
   }

finally
   {
   con.close();
   }
...
```

Example 27–7 demonstrates performing a transaction that spans multiple caches. Each transactional cache must be defined in a cache configuration file.

Example 27–7 Transaction Across Multiple Caches

```
...
Connection con = new DefaultConnectionFactory().
   createConnection("TransactionalCache");

con.setAutoCommit(false);
OptimisticNamedCache cache = con.getNamedCache("MyTxCache");
OptimisticNamedCache cache1 = con.getNamedCache("MyTxCache1");

cache.insert(key, value);
cache1.insert(key2, value2);

con.commit();

con.close();
...
```

Note: Transactions can span multiple partitions and caches within the same service but cannot span multiple services.

Creating Transactional Connections

The `com.tangosol.coherence.transaction.DefaultConnectionFactory` class is used to create `com.tangosol.coherence.transaction.Connection` instances. The following code from Example 27–5 demonstrates creating a `Connection` instance using the factory's no argument constructor:

```
Connection con = new DefaultConnectionFactory().
   createConnection("TransactionalCache");
```

In this example, the first cache configuration file found on the classpath (or specified using the `-Dtangosol.coherence.cacheconfig` system property) is used by this `Connection` instance. Optionally, a URI can be passed as an argument to the factory class that specifies the location and name of a cache configuration file. For example, the following code demonstrates constructing a connection factory that uses a cache configuration file named `cache-config.xml` that is located in a `config` directory found on the classpath.

```
Connection con = new DefaultConnectionFactory("config/cache-config.xml").
   createConnection("TransactionalCache");
```

The `DefaultConnectionFactory` class provides methods for creating connections:

- `createConnection()` – The no-argument method creates a connection that is a member of the default transactional service, which is named `TransactionalCache`. Use the no-argument method when the `<transactional-scheme>` element being used does not include a specific`<service-name>` element. For details on defining transactional cache schemes and specifying the service name, see "Defining Transactional Caches" on page 27-6.
- `createConnection(ServiceName)` – This method creates a connection that is a member of a transactional service. The service name is a `String` that indicates the transactional service to which this connection belongs. The `ServiceName` maps to a `<service-name>` element that is defined within a `<transactional-scheme>` element in the cache configuration file. If no service name is used, the default name (`TransactionalCache`) is used as the service

name. For details on defining transactional cache schemes and specifying the service name, see "Defining Transactional Caches" on page 27-6.

- `createConnection(ServiceName, loader)` – This method also creates a connection that is a member of a transactional service. In addition, it specifies the class loader to use. In the above example, the connection is created by only specifying a service name; in which case, the default class loader is used.

Using Transactional Connections

The `com.tangosol.coherence.transaction.Connection` interface represents a logical connection to a Coherence service. An active connection is always associated with a transaction. A new transaction implicitly starts when a connection is created and also when a transaction is committed or rolled back.

Transactions that are derived from a connection have several default behaviors that are listed below. The default behaviors balance ease-of-use with performance.

- A transaction is automatically committed or rolled back for each cache operation. See "Using Auto-Commit Mode" below.
- A transaction uses the read committed isolation level. See "Setting Isolation Levels" below.
- A transaction immediately performs operations on the cache. See "Using Eager Mode" below.
- A transaction has a default timeout of 300 seconds. See "Setting Transaction Timeout" below.

A connection's default behaviors can be changed using the `Connection` instance's methods as required.

Using Auto-Commit Mode

Auto-commit mode allows an application to choose whether each cache operation should be associated with a separate transaction or whether multiple cache operations should be executed as a single transaction. Each cache operation is executed in a distinct transaction when auto-commit is enabled; the framework automatically commits or rolls back the transaction after an operation completes and then the connection is associated with a new transaction and the next operation is performed. By default, auto-commit is enabled when a `Connection` instance is created.

The following code from Example 27–5 demonstrates `insert` operations that are each performed as a separate transaction:

```
OptimisticNamedCache cache = con.getNamedCache("MyTxCache");

cache.insert(key, value);
cache.insert(key2, value2);
```

Multiple operations are performed as part of a single transaction by disabling auto-commit mode. If auto-commit mode is disabled, an application must manually demarcate transaction boundaries. The following code from Example 27–6 demonstrates `insert` operations that are performed within a single transaction:

```
con.setAutoCommit(false);

OptimisticNamedCache cache = con.getNamedCache("MyTxCache");

cache.insert(key, value);
cache.insert(key2, value2);
```

```
con.commit();
```

An application cannot use the `commit()` or `rollback()` method when auto-commit mode is enabled. Moreover, if auto-commit mode is enabled while in an active transaction, any work is automatically rolled back.

Setting Isolation Levels

Isolation levels help control data concurrency and consistency. The Transaction Framework uses implicit write-locks and does not implement read-locks. Any attempt to write to a locked entry results in an `UnableToAcquireLockException`; the request does not block. When a transaction is set to eager mode, the exception is thrown immediately. In non-eager mode, exceptions may not be thrown until the statement is flushed, which is typically at the next read or when the transaction commits. See "Using Eager Mode" on page 27-13.

The Coherence Transaction Framework API supports the following isolation levels:

- `READ_COMMITTED` – This is the default isolation level if no level is specified. This isolation level guarantees that only committed data is visible and does not provide any consistency guarantees. This is the weakest of the isolation levels and generally provides the best performance at the cost of read consistency.
- `STMT_CONSISTENT_READ` – This isolation level provides statement-scoped read consistency which guarantees that a single operation only reads data for the consistent read version that was available at the time the statement began. The version may or may not be the most current data in the cache. See the note below for additional details.
- `STMT_MONOTONIC_CONSISTENT_READ` – This isolation level provides the same guarantees as `STMT_CONSISTENT_READ`, but reads are also guaranteed to be monotonic. A read is guaranteed to return a version equal or greater than any version that was previously encountered while using the connection. Due to the monotinic read guarantee, reads with this isolation may block until the necessary versions are available.
- `TX_CONSISTENT_READ` – This isolation level provides transaction-scoped read consistency which guarantees that all operations performed in a given transaction read data for the same consistent read version that was available at the time the transaction began. The version may or may not be the most current data in the cache. See the note below for additional details.
- `TX_MONOTONIC_CONSISTENT_READ` – This isolation level provides the same guarantees as `TX_CONSISTENT_READ`, but reads are also guaranteed to be monotonic. A read is guaranteed to return a version equal or greater than any version that was previously encountered while using the connection. Due to the monotinic read guarantee, the initial read in a transaction with this isolation may block until the necessary versions are available.

> **Note:** Consistent read isolation levels (statement or transaction) may lag slightly behind the most current data in the cache. If a transaction writes and commits a value, then immediately reads the same value in the next transaction with a consistent read isolation level, the updated value may not be immediately visible. If reading the most recent value is critical, then the `READ_COMMITTED` isolation level is required.

Isolation levels are set on a `Connection` instance and must be set before starting an active transaction. For example:

```
...
Connection con = new DefaultConnectionFactory().
createConnection("TransactionalCache");

con.setIsolationLevel(STMT_CONSISTENT_READ);
...
```

Using Eager Mode

Eager mode allows an application to control when cache operations are performed on the cluster. If eager mode is enabled, cache operations are immediately performed on the cluster. If eager mode is disabled, cache operations are deferred, if possible, and queued to be performed as a batch operation. Typically, an operation can only be queued if it does not return a value. An application may be able to increase performance by disabling eager mode.

By default, eager mode is enabled and cache operations are immediately performed on the cluster. The following example demonstrates disabling eager mode.

```
...
Connection con = new DefaultConnectionFactory().
createConnection("TransactionalCache");

con.setEager(false);
...
```

Setting Transaction Timeout

The transaction timeout allows an application to control how long a transaction can remain active before it is rolled back. The transaction timeout is associated with the current transaction and any new transactions that are associated with the connection.

The timeout value is specified in seconds. The default timeout value is `300` seconds. The following example demonstrates setting the transaction timeout value.

```
...
Connection con = new DefaultConnectionFactory().
createConnection("TransactionalCache");

con.setTransactionTimeout(420);
...
```

Using the OptimisticNamedCache Interface

The `com.tangosol.coherence.transaction.OptimisticNamedCache` interface extends the `NamedCache` interface and adds the operations: `update()`, `delete()`, and `insert()`.

All transactional caches are derived from this type. This cache type ensures that an application use the framework's concurrency and data locking implementations.

> **Note:** `OptimisticNamedCache` does not extend any operations from the `ConcurrentMap` interface since it uses its own locking strategy.

The following code sample from Example 27–5 demonstrates getting a transactional cache called `MyTxCache` and performs operations on the cache. For this example, a transactional cache that is named `MyTxCache` must be located in the cache configuration file at run-time. For details on defining a transactional cache, see "Defining Transactional Caches" on page 27-6.

```
OptimisticNamedCache cache = con.getNamedCache("MyTxCache");

cache.insert(key, value);
cache.insert(key2, value2);
```

Configuring POF When Performing Transactions

Transactional caches support Portable Object Format (POF) serialization within transactions. POF is enabled within a transactional cache scheme using the `<serializer>` element. The following example demonstrates enabling POF serialization in a transactional cache scheme.

```
<transactional-scheme>
   <scheme-name>example-transactional</scheme-name>
   <service-name>TransactionalCache</service-name>
   <serializer>
      <instance>
         <class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>
      </instance>
   </serializer>
   <autostart>true</autostart>
</transactional-scheme>
```

The Transaction Framework API also includes its own POF types which are defined in the `txn-pof-config.xml` POF configuration file which is included in `coherence.jar`. The POF types are required and must be found at run-time.

To load the transaction POF types at run time, modify an application's POF configuration file and include the `txn-pof-config.xml` POF configuration file using the `<include>` element. For example:

```
<pof-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-pof-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-pof-config
   coherence-pof-config.xsd">
   <user-type-list>
      <include>coherence-pof-config.xml</include>
      <include>txn-pof-config.xml</include>
   </user-type-list>
   ...
</pof-config>
```

See "Combining Multiple POF Configuration Files" on page 3-10 for more information on using the `<include>` element to combine POF configuration files.

Configuring Transactional Storage Capacity

The Transaction Framework API stores transactional data in internal distributed caches that use backing maps. The data includes versions of all keys and their values for a transactional cache. The framework uses the stored data in roll-back scenarios and also during recovery.

Due to the internal storage requirements, transactional caches have a constant overhead associated with every entry written to the cache. Moreover, transactional caches use multi-version concurrency control, which means that every write operation produces a new row into the cache even if it is an update. Therefore, the Transaction Framework API uses a custom eviction policy to help manage the growth of its internal storage caches. The eviction policy works by determining which versions of an entry can be kept and which versions are eligible for eviction. The latest version for a given key (the most recent) is never evicted. The eviction policy is enforced whenever a configured high-water mark is reached. After the threshold is reached, 25% of the eligible versions are removed.

Note:

- The eviction policy does not take the entire transactional storage into account when comparing the high-water mark. Therefore, transactional storage slightly exceeds the high-water mark before the storage eviction policy is notified.
- It is possible that storage for a transactional cache exceeds the maximum heap size if the cache is sufficiently broad (large number of distinct keys) since the current entry for a key is never evicted.

Because the storage eviction policy is notified on every write where the measured storage size exceeds the high-water mark, the default high-water mark may have to be increased so that it is larger than the size of the current data set. Otherwise, the eviction policy is notified on every write after the size of the current data set exceeds the high water mark resulting in decreased performance. If consistent reads are not used, the value can be set so that it slightly exceeds the projected size of the current data set since no historical versions is ever read. When using consistent reads, the high-water mark should be high enough to provide for enough historical versions. Use the below formulas to approximate the transactional storage size.

The high-water mark is configured using the `<high-units>` element within a transactional scheme definition. The following example demonstrates configuring a high-water mark of 20 MB.

```
<transactional-scheme>
   ...
   <high-units>20M</high-units>
   ...
</trnsactional-scheme>
```

The following formulas provide a rough estimate of the memory usage for a row in a transactional cache.

For insert operations:

- Primary – key(serialized) + key (on-heap size) + value(serialized) + 1095 bytes constant overhead
- Backup – key(serialized) + value(serialized) + 530 bytes constant overhead

For updated operations:

- Primary – value(serialized) + 685 bytes constant overhead
- Backup – value(serialized) + 420 bytes constant overhead

Performing Transactions from Java Extend Clients

The Transaction Framework API provides Java extend clients with the ability to perform cache operations within a transaction. In this case, the transaction API is used within an entry processor that is located on the cluster. At run time, the entry processor is executed on behalf of the Java client.

The instructions in this section do not include detailed instructions on how to setup and use Coherence*Extend. For those new to Coherence*Extend, see "Setting Up Coherence*Extend" in *Oracle Coherence Client Guide*. For details on performing transactions from C++ or .NET clients, see "Performing Transactions for C++ Clients" and "Performing Transactions for .NET Clients" in the *Oracle Coherence Client Guide*.

The following topics are included in this section and are required to perform transactions from Java extend clients:

- Create an Entry Processor for Transactions
- Configure the Cluster-Side Transaction Caches
- Configure the Client-Side Remote Cache
- Use the Transactional Entry Processor from a Java Client

Create an Entry Processor for Transactions

Transactions are performed using the transaction API within an entry processor that resides on the cluster. The entry processor is executed on behalf of a Java extend client.

Example 27–8 demonstrates an entry processor that performs a simple `update` operation within a transaction. At run time, the entry processor must be located on both the client and cluster.

Example 27–8 Entry Processor for Extend Client Transaction

```
public class MyTxProcessor extends AbstractProcessor
   {
   public Object process(InvocableMap.Entry entry)
   {
      // obtain a connection and transaction cache
      ConnectionFactory connFactory = new DefaultConnectionFactory();
      Connection conn = connFactory.createConnection("TransactionalCache");
      OptimisticNamedCache cache = conn.getNamedCache("MyTxCache");

      conn.setAutoCommit(false);

      // get a value for an existing entry
      String sValue = (String) cache.get("existingEntry");

      // create predicate filter
      Filter predicate = new EqualsFilter(IdentityExtractor.INSTANCE, sValue);

      try
         {
            // update the previously obtained value
            cache.update("existingEntry", "newValue", predicate);
         }
      catch (PredicateFailedException e)
         {
            // value was updated after it was read
            conn.rollback();
            return false;
```

```
            }
         catch (UnableToAcquireLockException e)
            {
               // row is being updated by another tranaction
               conn.rollback();
               return false;
            }
         try
            {
               conn.commit();
            }
         catch (RollbackException e)
            {
               // transaction was rolled back
               return false;
            }
         return true;
      }
   }
```

Configure the Cluster-Side Transaction Caches

Transactions require a transactional cache to be defined in the cluster-side cache configuration file. For details on defining a transactional cache, see "Defining Transactional Caches" on page 27-6.

The following example defines a transactional cache that is named `MyTxCache`, which is the cache name that was used by the entry processor in Example 27–8. The example also includes a proxy scheme and a distributed cache scheme that are required to execute the entry processor from a remote client. The proxy is configured to accept client TCP/IP connections on `localhost` at port `9099`.

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <caching-scheme-mapping>
      <cache-mapping>
         <cache-name>MyTxCache</cache-name>
         <scheme-name>example-transactional</scheme-name>
      </cache-mapping>
      <cache-mapping>
         <cache-name>dist-example</cache-name>
         <scheme-name>example-distributed</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <transactional-scheme>
         <scheme-name>example-transactional</scheme-name>
         <thread-count>7</thread-count>
         <high-units>15M</high-units>
         <task-timeout>0</task-timeout>
         <autostart>true</autostart>
      </transactional-scheme>
      <distributed-scheme>
         <scheme-name>example-distributed</scheme-name>
         <service-name>DistributedCache</service-name>
```

```
         <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>
```

Configure the Client-Side Remote Cache

Remote clients require a remote cache to connect to the cluster's proxy and run a transactional entry processor. The remote cache is defined in the client-side cache configuration file.

The following example configures a remote cache to connect to a proxy that is located on `localhost` at port `9099`. In addition, the name of the remote cache (`dist-example`) must match the name of a cluster-side cache that is used when initiating the transactional entry processor.

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <caching-scheme-mapping>
      <cache-mapping>
         <cache-name>dist-example</cache-name>
         <scheme-name>extend</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <remote-cache-scheme>
         <scheme-name>extend</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>30s</connect-timeout>
            </tcp-initiator>
            <outgoing-message-handler>
               <request-timeout>30s</request-timeout>
```

```
            </outgoing-message-handler>
          </initiator-config>
        </remote-cache-scheme>
    </caching-schemes>
</cache-config>
```

Use the Transactional Entry Processor from a Java Client

A Java extend client invokes an entry processor as normal. However, at run time, the cluster-side entry processor is invoked. The client is unaware that the invocation has been delegated. The following example demonstrates how a Java client calls the entry processor shown in Example 27–8.

```
NamedCache  cache    =  CacheFactory.getCache("dist-example");
Object      oReturn  = cache.invoke("AnyKey",  new MyTxProcessor());

System.out.println("Result of extend tx execution: " + oReturn);
```

Viewing Transaction Management Information

The transaction framework leverages the existing Coherence JMX management framework. See *Oracle Coherence Management Guide* for detailed information on enabling and using JMX in Coherence.

This section describes two MBeans that provide transaction information: `CacheMBean` and `TransactionManagerMBean`.

CacheMBeans for Transactional Caches

The `CacheMBean` managed resource provides attributes and operations for all caches, including transactional caches. Many of the MBeans attributes are not applicable to transactional cache; invoking such attributes simply returns a -1 value. A cluster node may have zero or more instances of cache managed beans for transactional caches. The object name uses the form:

`type=Cache, service=`*`service name`*`, name=`*`cache name`*`,`
`nodeId=`*`cluster node's id`*

Table 27–2 describes the `CacheMBean` attributes that are supported for transactional caches.

Table 27–2 Transactional Cache Supported Attributes

Attribute	Type	Description
`AverageGetMillis`	Double	The average number of milliseconds per `get()` invocation
`AveragePutMillis`	Double	The average number of milliseconds per `put()` invocation since the cache statistics were last reset.
`Description`	String	The cache description.
`HighUnits`	Integer	The limit of the cache size measured in units. The cache prunes itself automatically after it reaches its maximum unit level. This is often referred to as the high water mark of the cache.
`Size`	Integer	The number of entries in the current data set
`TotalGets`	Long	The total number of `get()` operations since the cache statistics were last reset.

Table 27–2 (Cont.) Transactional Cache Supported Attributes

Attribute	Type	Description
`TotalGetsMillis`	Long	The total number of milliseconds spent on `get()` operations since the cache statistics were last reset.
`TotalPuts`	Long	The total number of `put()` operations since the cache statistics were last reset.
`TotalPutsMillis`	Long	The total number of milliseconds spent on `put()` operations since the cache statistics were last reset.

For transactional caches, the `resetStatistics` operation is supported and resets all transaction manager statistics.

TransactionManagerBean

The `TransactionManagerMBean` managed resource is specific to the transactional framework. It provides global transaction manager statics by aggregating service-level statistics from all transaction service instances. Each cluster node has an instance of the transaction manager managed bean per service. The object name uses the form:

`type=TransactionManager, service=`*`service name`*`, nodeId=`*`cluster node's id`*

Note: For certain transaction manager attributes, the count is maintained at the coordinator node for the transaction, even though multiple nodes may have participated in the transaction. For example, a transaction may include modifications to entries stored on multiple nodes but the `TotalCommitted` attribute is only incremented on the MBean on the node that coordinated the commit of that transaction.

Table 27–3 describes TransactionManager attributes.

Table 27–3 TransactionManagerMBean Attributes

Attribute	Type	Description
`TotalActive`	Long	The total number of currently active transactions. An active transaction is counted as any transaction that contains at least one modified entry and has yet to be committed or rolled back. Note that the count is maintained at the coordinator node for the transaction, even though multiple nodes may have participated in the transaction.
`TotalCommitted`	Long	The total number of transactions that have been committed by the Transaction Manager since the last time the statistics were reset. Note that the count is maintained at the coordinator node for the transaction being committed, even though multiple nodes may have participated in the transaction.
`TotalRecovered`	Long	The total number of transactions that have been recovered by the Transaction Manager since the last time the statistics were reset. Note that the count is maintained at the coordinator node for the transaction being recovered, even though multiple nodes may have participated in the transaction.

Table 27–3 (Cont.) TransactionManagerMBean Attributes

Attribute	Type	Description
`TotalRolledback`	Long	The total number of transactions that have been rolled back by the Transaction Manager since the last time the statistics were reset. Note that the count is maintained at the coordinator node for the transaction being rolled back, even though multiple nodes may have participated in the transaction.
`TotalTransactionMillis`	Long	The cumulative time (in milliseconds) spent on active transactions.
`TimeoutMillis`	Long	The transaction timeout value in milliseconds. Note that this value only applies to transactional connections obtained after the value is set. This attribute is currently not supported.

The `TransactionManagerMBean` includes a single operation called `resetStatistics`, which resets all transaction manager statistics.

Using the Coherence Resource Adapter

Coherence includes a J2EE Connector Architecture (J2CA) 1.5 compliant resource adaptor that is used to get connections to a Coherence cache. The resource adapter leverages the connection API of the Coherence Transaction Framework and therefore provides default transaction guarantees. In addition, the resource adapter provides full XA support which allows Coherence to participate in global transactions. A global transaction is unit of work that is managed by one or more resource managers and is controlled and coordinated by an external transaction manager, such as the transaction manager that is included with WebLogic server or OC4J.

The resource adapter is packaged as a standard Resource Adaptor Archive (RAR) and is named `coherence-transaction.rar`. The resource adapter is located in *`COHERENCE_HOME`*`/lib` and can be deployed to any Java EE container compatible with J2CA 1.5. The resource adapter includes proprietary resource adapter deployment descriptors for WebLogic (`weblogic-ra.xml`) and OC4J (`oc4j-ra.xml`) and can be deployed to these platforms without modification. Check your application server vendor's documentation for details on defining a proprietary resource adapter descriptor that can be included within the RAR.

> **Note:** Coherence continues to include the `coherence-tx.rar` resource adapter for backward compatibility. However, it is strongly recommended that applications use the `coherence-transaction.rar` resource adapter which provides full XA support. Those accustomed to using the Coherence `CacheAdapter` class can continue to do so with either resource adapter. See "Using the Coherence Cache Adapter for Transactions" on page 27-27.

The following topics are included in this section:

- Performing Cache Operations within a Transaction
- Packaging the Application
- Using the Coherence Cache Adapter for Transactions

Performing Cache Operations within a Transaction

Java EE application components (Servlets, JSPs, and EJBs) use the Coherence resource adapter to perform cache operations within a transaction. The resource adapters supports both local transactions and global transactions. Local transactions are used to perform cache operations within a transaction that is only scoped to a Coherence cache and cannot participate in a global transaction. Global transactions are used to perform cache operations that automatically commit or roll back based on the outcome of multiple resources that are enlisted in the transaction.

Like all JavaEE application components, the Java Naming and Directory Interface (JNDI) API is used to lookup the resource adapter's connection factory. The connection factory is then used to get logical connections to a Coherence cache.

The following examples demonstrate how to use the Coherence resource adapter to perform cache operations within a global transaction. Example 27–9 is an example of using Container Managed Transactions (CMT); where, the container ensures that all methods execute within the scope of a global transaction. Example 27–10 is an example of user-controlled transactions; where, the application component uses the Java Transaction API (JTA) to manually demarcate transaction boundaries.

Transactions require a transactional cache scheme to be defined within a cache configuration file. These examples use the transactional cache that was defined in Example 27–4.

Example 27–9 Performing a Transaction When Using CMT

```
Context initCtx = new InitialContext();
ConnectionFactory cf = (ConnectionFactory)
 initCtx.lookup("java:comp/env/eis/CoherenceTxCF");

Connection con = cf.createConnection("TransactionalCache");

try
   {
   OptimisticNamedCache cache = con.getNamedCache("MyTxCache");

   cache.delete("key1", null);
   cache.insert("key1", "value1");
   }
finally
   {
   con.close();
   }
```

Example 27–10 Performing a User-Controlled Transaction

```
Context initCtx = new InitialContext();
ConnectionFactory cf = (ConnectionFactory)
 initCtx.lookup("java:comp/env/eis/CoherenceTxCF");

UserTransaction ut = (UserTransaction) new
 InitialContext().lookup("java:comp/UserTransaction");
ut.begin();

Connection con = cf.createConnection("TransactionalCache");

try
   {
   OptimisticNamedCache cache = con.getNamedCache("MyTxCache");
```

```
   cache.delete("key1", null);
   cache.insert("key1", "value1");
   ut.commit();
   }

catch (Exception e)
   {
   ut.rollback();
   throw e;
   }

finally
   {
   con.close();
   }
```

Creating a Coherence Connection

Applications use the `com.tangosol.coherence.ConnectionFactory` interface to create connections to a Coherence cluster. An instance of this interface is obtained using a JNDI lookup. The following code sample from Example 27–10 performs a JNDI lookup for a connection factory that is bound to the `java:comp/env/eis/CoherenceTxCF` namespace:

```
Context initCtx = new InitialContext();
ConnectionFactory cf = (ConnectionFactory)
   initCtx.lookup("java:comp/env/eis/CoherenceTxCF");
```

The `ConnectionFactory` is then used to create a `com.tangosol.coherence.transaction.Connection` instance. The `Connection` instance represents a logical connection to a Coherence service:

```
Connection con = cf.createConnection("TransactionalCache");
```

The `creatConnection(ServiceName)` method creates a connection that is a member of a transactional service. The service name is a `String` that indicates which transactional service this connection belongs to and must map to a service name that is defined in a `<transactional-scheme>` within a cache configuration file. For details on defining transactional cache schemes and specifying the service name, see "Defining Transactional Caches" on page 27-6.

A `Connection` instance always has an associated transaction which is scoped within the connection. A new transaction is started when a transaction is completed. The following default behaviors are associated with a connection. For more information on the `Connection` interface and changing the default settings, see "Using Transactional Connections" on page 27-11.

- Connections are in auto-commit mode by default which means that each statement is executed in a distinct transaction and when the statement completes the transaction is committed and the connection is associated with a new transaction.

> **Note:** When the connection is used for a global transaction, auto-commit mode is disabled and cannot be enabled. Cache operations are performed in a single transaction and either commit or roll back as a unit. In addition, the `Connection` interface's commit an rollback methods cannot be used if the connection is enlisted in a global transaction.

- The connection's isolation level is set to `READ_COMMITTED`. The transaction can only view committed data from other transactions.
- Eager mode is enabled by default which means every operation is immediately flushed to the cluster and are not queued to be flushed in batches.
- The default transaction timeout is 300 seconds.

> **Note:** When the connection is used for a global transaction, the transaction timeout that is associated with a connection is overridden by the transaction timeout value that is set by a container's JTA configuration. If an application attempts to set the transaction timeout value directly on the connection while it is enlisted in a global transaction, the attempt is ignored and a warning message is emitted indicating that the transaction timeout cannot be set. The original timeout value that is set on the connection is restored after the global transaction completes.

Getting a Named Cache

The `com.tangosol.coherence.transaction.OptimisticNamedCache` interface extends the `NamedCache` interface. It supports all the customary named cache operations and adds its own operations for updating, deleting, and inserting objects into a cache. When performing transactions, all cache instances must be derived from this type. The following code sample from Example 27–10 demonstrates getting a named cache called `MyTxCache` and performing operations on the cache. The cache must be defined in the cache configuration file.

```
try
   {
   OptimisticNamedCache cache = con.getNamedCache("MyTxCache");

   cache.delete("key1", null);
   cache.insert("key1", "value1");
```

> **Note:** `OptimisticNamedCache` does not extend any operations from the `ConcurrentMap` interface since it uses its own locking strategy.

Demarcating Transaction Boundaries

Application components that perform user-controlled transactions use a JNDI lookup to get a JTA `UserTransaction` interface instance. The interface provide methods for demarcating the transaction. The following code sample from Example 27–10 demonstrates getting a `UserTransaction` instance and demarcating the transaction boundaries:

```
UserTransaction ut = (UserTransaction) new
 InitialContext().lookup("java:comp/UserTransaction");

ut.begin();
Connection con = cf.createConnection("TransactionalCache");

try
   {
   OptimisticNamedCache cache = con.getNamedCache("MyTxCache");

   cache.delete("key1", null);
   cache.insert("key1", "value1");
   ut.commit();
```

The above code demonstrates a typical scenario where the connection and the named cache exist within the transaction boundaries. However, the resource adapter also supports scenarios where connections are used across transaction boundaries and are obtained before the start of a global transaction. For example:

```
Connection con = cf.createConnection("TransactionalCache");

try
   {
   OptimisticNamedCache cache = con.getNamedCache("MyTxCache");

   cache.delete("key1", null);

   UserTransaction ut = (UserTransaction) new
    InitialContext().lookup("java:comp/UserTransaction");

   ut.begin();
   cache.insert("key1", "value1");
   ut.commit();
```

Packaging the Application

This section provides instructions for packaging JavaEE applications that use the Coherence resource adapter so that they can be deployed to an application server. The following topics are included in this section:

- Configure the Connection Factory Resource Reference
- Configure the Resource Adapter Module Reference
- Include the Required Libraries

Configure the Connection Factory Resource Reference

Application components must provide a resource reference for the resource adapter's connection factory. For EJBs, the resource references are defined in the `ejb-jar.xml` deployment descriptor. For Servlets and JSPs, the resource references are defined in the `web.xml` deployment descriptor. The following sample demonstrates defining a resource reference for the resource adapter's connection factory and is applicable to the code in Example 27–10:

```
<resource-ref>
   <res-ref-name>eis/CoherenceTxCF</res-ref-name>
   <res-type>
      com.tangosol.coherence.transaction.ConnectionFactory
   </res-type>
```

```
   <res-auth>Container</res-auth>
</resource-ref>
```

In addition to the standard Java EE application component deployment descriptors, many application servers require a proprietary deployment descriptor as well. For example, WebLogic server resource references are defined in the `weblogic.xml` or `weblogic-ejb-jar.xml` files respectively:

```
<reference-descriptor>
   <resource-description>
      <res-ref-name>eis/CoherenceTxCF</res-ref-name>
      <jndi-name>tangosol.coherenceTx</jndi-name>
   </resource-description>
</reference-descriptor>
```

Consult your application server vendor's documentation for detailed information on using their proprietary application component deployment descriptors and information on alternate methods for defining resource reference using dependency injection or annotations.

Configure the Resource Adapter Module Reference

JavaEE applications must provide a module reference for the Coherence resource adapter. The module reference is defined in the EAR's `application.xml` file. The module reference points to the location of the Coherence RAR file (`coherence-transaction.rar`) within the EAR file. For example, the following definition points to the Coherence resource adapter RAR file that is located in the root of the EAR file:

```
<application>
...
<module>
   <connector>coherence-transaction.rar</connector>
</module>
...
</application>
```

In addition to the standard Java EE application deployment descriptors, many application servers require a proprietary application deployment descriptor as well. For example, the Coherence resource adapter is defined in the WebLogic server `weblogic-application.xml` file as follows:

```
<weblogic-application>
   <classloader-structure>
      ...
      <module-ref>
         <module-uri>coherence-transaction.rar</module-uri>
      </module-ref>
      ...
   </classloader-structure>
</weblogic-application>
```

Consult your application server vendor's documentation for detailed information on using their proprietary application deployment descriptors

Include the Required Libraries

JavaEE applications that use the Coherence resource adapter must include the `coherence-transaction.rar` file and the `coherence.jar` file within the EAR file. The following example places the libraries at the root of the EAR file:

```
/
/coherence-transaction.rar
/coherence.jar
```

When deploying to WebLogic server, the `coherence.jar` file must be placed in the `/APP-INF/lib` directory of the EAR file. For example:

```
/
/coherence-transaction.rar
/APP-INF/lib/coherence.jar
```

This deployment scenario results in a single Coherence cluster node that is shared by all application components in the EAR. See *Oracle Coherence Administrator's Guide* for different Coherence deployment options.

Using the Coherence Cache Adapter for Transactions

The Coherence `CacheAdapter` class provides an alternate client approach for creating transactions and is required when using the `coherence-tx.rar` resource adapter. The new `coherence-transaction.rar` resource adapter also supports the `CacheAdapter` class (with some modifications) and allows those accustomed to using the class to leverage the benefits of the new resource adapter. However, it is recommended that applications use the Coherence resource adapter natively which offers stronger transactional support. Examples for both resource adapters is provided in this section.

Example 27–11 demonstrates performing cache operations within a transaction when using the `CacheAdapter` class with the new `coherence-transaction.rar` resource adapter. For this example a transactional cache named `MyTxCache` must be configured in the cache configuration file. The cache must map to a transactional cache scheme with the service name `TransactionalCache`. See "Defining Transactional Caches" on page 27-6 for more information on defining a transactional cache scheme.

Example 27–11 Using the CacheAdapter Class When Using coherence-transaction.rar

```
Context initCtx = new InitialContext();

CacheAdapter adapter = new CacheAdapter(initCtx,
   "java:comp/env/eis/CoherenceTxCCICF", 0, 0, 0);

adapter.connect("TransactionalCache", "scott", "tiger");

try
   {
   UserTransaction ut = (UserTransaction) new
      InitialContext().lookup("java:comp/UserTransaction");

   ut.begin();
   OptimisticNamedCache cache =
   (OptimisticNamedCache) adapter.getNamedCache("MyTxCache",
      getClass().getClassLoader());
   cache.delete("key", null);
   cache.insert("key", "value");
   ut.commit();

   }
finally
   {
   adapter.close();
```

```
   }
```

Example 27–12 demonstrates performing cache operations within a transaction when using the `CacheAdapter` class with the `coherence-tx.rar` resource adapter.

Example 27–12 Using the CacheAdapter Class When Using coherence-tx.rar

```
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);
```

```
        }
    }
```

28

Working with Partitions

This chapter provides instructions for using data affinity to ensure data is located in specific partitions and also includes instructions for changing the default partition distribution strategy.

The following sections are included in this chapter:

Specifying 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 generally is 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 are two benefits for using data affinity. First, only a single cache node is required to manage queries and transactions against a set of related items. Second, all concurrency operations are managed locally and avoids 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.

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 have 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 do not contain the parent key information (as it is derived data, the size of the data may be decided explicitly, and it also does 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 are all assigned to a single partition, and are not 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` is 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 28–1 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 28–2 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 28–3 Configuring a Key Associator

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

Example of Using Affinity

Example 28–4 illustrates how to use affinity to create a more efficient query (`NamedCache.entrySet(Filter)`) and cache access (`NamedCache.getAll(Collection)`).

Example 28–4 Using Affinity for a More Efficient Query

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

// this Filter is 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 directs 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);
```

Changing the Partition Distribution Strategy

Partition distribution defines how partitions are assigned to storage-enabled cluster members. There are two methods of distribution available:

- Autonomous distribution – This is the default distribution strategy. Each storage-enabled service member acts autonomously to calculate its own balanced

share of the available partitions and to request distribution in order to balance the service.

- Centralized distribution – This method of distribution uses a centralized partition assignment strategy to make a global distribution decision to be carried out by each storage-enabled member. The centralized distribution allows for more expressive distribution algorithms to be utilized and uses a more complete and global view of the service.

The autonomous distribution strategy is enabled by default. To use a centralized distribution strategy, an assignment strategy implementation must be provided. The assignment strategy must implement the `com.tangosol.net.partition.PartitionAssignmentStrategy` interface. Coherence includes a default assignment strategy implementation and custom assignment strategies can be written and configured as required.

Specifying the Simple Partition Assignment Strategy

The simple partition assignment strategy is a centralized distribution method that uses a similar partition-balancing algorithm as the default autonomous distribution. The simple strategy attempts to distribute a fair-share number of partitions to each storage-enabled server while maintaining (as much as the service membership topology allows) machine, rack, or site safety. The assignment strategy is implemented in the `com.tangosol.net.partition.SimpleAssignmentStrategy` class, which can be extended as required.

To configure the simple partition assignment strategy for a specific partitioned cache service, add a `<partition-assignment-strategy>` element within a distributed cache definition and include the fully qualified class name within the `<instance>` element. For example:

```
<distributed-scheme>
   ...
   <partition-assignment-strategy>
      <instance>
         <class-name>com.tangosol.net.partition.SimpleAssignmentStrategy
         </class-name>
      </instance>
   </partition-assignment-strategy>
   ...
</distributed-scheme>
```

To configure the partition assignment strategy for all instances of the distributed cache service type, override the partitioned cache service's `partition-assignment-strategy` initialization parameter in an operational override file. For example:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <services>
         <service id="3">
            <init-params>
               <init-param id="21">
                  <param-name>partition-assignment-strategy</param-name>
                  <param-value>com.tangosol.net.partition.SimpleAssignmentStrategy
```

```
                  </param-value>
               </init-param>
            </init-params>
         </service>
      </services>
   </cluster-config>
</coherence>
```

Enabling a Custom Partition Assignment Strategy

To specify a Custom partition assignment strategy, include an `<instance>` subelement within the `<partition-assignment-strategy>` element and provide a fully qualified class name that implements the `com.tangosol.net.partition.PartitionAssignmentStrategy` interface. See "instance" on page B-49 for detailed instructions on using the `<instance>` element. The following example enables a partition assignment strategy that is implemented in the `MyPAStrategy` class.

```
<distributed-scheme>
   ...
   <partition-assignment-strategy>
      <instance>
         <class-name>package.MyPAStrategy</class-name>
      </instance>
   </partition-assignment-strategy>
   ...
</distributed-scheme>
```

As an alternative, the `<instance>` element supports the use of a `<class-factory-name>` element to use a factory class that is responsible for creating `PartitionAssignmentStrategy` instances, and a `<method-name>` element to specify the static factory method on the factory class that performs object instantiation. The following example gets a strategy instance using the `getStrategy` method on the `MyPAStrategyFactory` class.

```
<distributed-scheme>
   ...
   <partition-assignment-strategy>
      <instance>
         <class-factory-name>package.MyPAStrategyFactory</class-factory-name>
         <method-name>getStrategy</method-name>
      </instance>
   </partition-assignment-strategy>
   ...
</distributed-scheme>
```

Any initialization parameters that are required for an implementation can be specified using the `<init-params>` element. The following example sets the `iMaxTime` parameter to `2000`.

```
<distributed-scheme>
   ...
   <partition-assignment-strategy>
      <instance>
         <class-name>package.MyPAStrategy</class-name>
         <init-params>
            <init-param>
               <param-name>iMaxTime</param-name>
               <param-value>2000</param-value>
```

```
            </init-param>
          </init-params>
        </instance>
      </partition-assignment-strategy>
      ...
    </distributed-scheme>
```

29

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. Use these features with extreme caution because they can dramatically affect performance and throughput of the data grid.

The following sections are included in this chapter:

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. For example, if a write-through in a `CacheStore` is blocked and exceeds the configured timeout value, the Coherence Task Manager attempts to interrupt the execution of the thread and an exception is thrown. In a similar fashion, queries or aggregations that exceed configured timeouts are interrupted and an exception is thrown. Applications that use this feature should ensure 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

The `<request-timeout>`, `<task-timeout>`, and the `<task-hung-threshold>` elements are used to configuring execution timeouts for a service's worker threads. These timeout settings are configured in a cache configuration file and can also be set using command line parameters. See Chapter 30, "Using the Service Guardian," for information on setting timeouts for service threads.

Table 29–1 describes the execution timeout elements.

Table 29–1 Execution Timeout Elements

Element Name	Description
`<request-timeout>`	Specifies the default timeout value for requests that can time out (for example, implement the `PriorityTask` interface), but do not 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. If the value does not contain a unit, a unit of milliseconds is assumed. Legal values are positive integers or zero (indicating no default timeout). Default value is an infinite timeout (`0s`) for clustered client requests and 30 seconds (`30s`) for non-clustered client requests.
`<task-timeout>`	Specifies the default timeout value for tasks that can be timed-out (for example, implement the `PriorityTask` interface), but do not 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). If `zero` is specified, the default service-guardian `<timeout-milliseconds>` value is used.
`<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).

The following example sets a distributed cache's thread count to `7` with a task time out of `5000` milliseconds and a task hung threshold of `10000` milliseconds:

Example 29–1 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-hung-threshold>10000</task-hung-threshold>
      <task-timeout>5000</task-timeout>
    </distributed-scheme>
</caching-schemes>
```

Setting the client request timeout to `15` milliseconds

Example 29–2 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

Use the command line options to set the service type default (such as distributed cache, invocation, proxy, and so on) for the node. Table 29–2 describes the options.

Table 29–2 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 is scheduled for execution in a natural (based on the request arrival time) order
- `SCHEDULE_FIRST`—a task is 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 is immediately executed by any idle worker thread; if all of them are active, a new thread is 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 must 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 29–3.

Table 29–3 Methods to Support Task Timeout

Method	Description
public long getRequestTimeoutMillis()	Obtains the maximum amount of time a calling thread is can 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 can 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 attempts 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 attempts 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 is called.
public int getSchedulingPriority()	Obtains this task's scheduling priority. Valid values are SCHEDULE_STANDARD, SCHEDULE_FIRST, SCHEDULE_IMMEDIATE

Table 29–3 (Cont.) Methods to Support Task Timeout

Method	Description
`public void runCanceled(boolean fAbandoned)`	This method is 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 causes a delay of the corresponding service.

Errors Thrown by Task Timeouts

When a task timeout occurs the node gets a `RequestTimeoutException`. Example 29–3 illustrates an exception that may be thrown.

Example 29–3 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)
```

30

Using the Service Guardian

The following sections are included in this chapter:

Overview

The service guardian is a mechanism that detects and attempts to resolve deadlocks in Coherence threads. Deadlocked threads on a member may result in many undesirable behaviors that are visible to the rest of the cluster, such as the inability to add new nodes to the cluster and the inability to service requests by nodes currently in the cluster.

The service guardian receives periodic heartbeats that are issued by Coherence-owned and created threads. Should a thread fail to issue a heartbeat before the configured timeout, the service guardian takes corrective action. Both the timeout and corrective action (recovery) can be configured as required.

> **Note:** The term deadlock does not necessarily indicate a true deadlock; a thread that does not issue a timely heartbeat may be executing a long running process or waiting on a slow resource. The service guardian does not have the ability to distinguish a deadlocked thread from a slow one.

Interfaces That Are Executed By Coherence

Implementations of the following interfaces are executed by Coherence-owned threads. Any processing in an implementation that exceeds the configured guardian timeout results in the service guardian attempting to recover the thread. The list is not exhaustive and only provides the most common interfaces that are implemented by end users.

```
com.tangosol.net.Invocable
com.tangosol.net.cache.CacheStore
com.tangosol.util.Filter
com.tangosol.util.InvocableMap.EntryAggregator
com.tangosol.util.InvocableMap.EntryProcessor
com.tangosol.util.MapListener
com.tangosol.util.MapTrigger
```

Understanding Recovery

The service guardian's recovery mechanism uses a series of steps to determine if a thread is deadlocked. Corrective action is taken if the service guardian concludes that the thread is deadlocked. The action to take can be configured and custom actions can be created if required. The recovery mechanism is outlined below:

- Soft Timeout – The recovery mechanism first attempts to interrupt the thread just before the configured timeout is reached. The following example log message demonstrates a soft timeout message:

  ```
  <Error> (thread=DistributedCache, member=1): Attempting recovery (due to soft
  timeout) of Daemon{Thread="Thread[WriteBehindThread:CacheStoreWrapper(com.
  tangosol.examples.rwbm.TimeoutTest),5,WriteBehindThread:CacheStoreWrapper(com.
  tangosol.examples.rwbm.TimeoutTest)]", State=Running}
  ```

  If the thread can be interrupted and it results in a heartbeat, normal processing resumes.

- Hard Timeout – The recovery mechanism attempts to stop a thread after the configured timeout is reached. The following example log message demonstrates a hard timeout message:

  ```
  <Error> (thread=DistributedCache, member=1): Terminating guarded execution (due
  to hard timeout) of Daemon{Thread="Thread[WriteBehindThread:CacheStoreWrapper
  (com.tangosol.examples.rwbm.TimeoutTest),5,WriteBehindThread:CacheStoreWrapper
  (com.tangosol.examples.rwbm.TimeoutTest)]", State=Running}
  ```

- Lastly, if the thread cannot be stopped, the recovery mechanism performs an action based on the configured failure policy. Actions that can be performed include: shutting down the cluster service, shutting down the JVM, and performing a custom action. The following example log message demonstrates an action taken by the recovery mechanism:

  ```
  <Error> (thread=Termination Thread, member=1): Write-behind thread timed out;
  stopping the cache service
  ```

Configuring the Service Guardian

The service guardian is enabled out-of-the box and has two configured items: the timeout value and the failure policy. The timeout value is the length of time the service guardian waits to receive a heartbeat from a thread before starting recovery. The failure policy is the corrective action that the service guardian takes after it concludes that the thread is deadlocked.

Setting the Guardian Timeout

The service guardian timeout can be set in three different ways based on the level of granularity that is required:

- All threads – This option allows a single timeout value to be applied to all Coherence-owned threads on a cluster node. This is the out-of-box configuration and is set at `305000` milliseconds by default.
- Threads per service type – This option allows different timeout values to be set for specific service types. The timeout value is applied to the threads of all service instances. If a timeout is not specified for a particular service type, then the timeout defaults to the timeout that is set for all threads.

- Threads per service instance – This option allows different timeout values to be set for specific service instances. If a timeout is not set for a specific service instance, then the service's timeout value, if specified, is used; otherwise, the timeout that is set for all threads is used.

Setting the timeout value to `0` stops threads from being guarded. In general, the service guardian timeout value should be set equal to or greater than the timeout value for packet delivery.

> **Note:** The guardian timeout can also be used for cache store implementations that are configured with a read-write-backing-map scheme. In this case, the `<cachestore-timeout>` element is set to `0`, which defaults the timeout to the guardian timeout. See "read-write-backing-map-scheme" on page B-88.

Setting the Guardian Timeout for All Threads

To set the guardian timeout for all threads in a cluster node, add a `<timeout-milliseconds>` element to an operational override file within the `<service-guardian>` element. The following example sets the timeout value to `120000` milliseconds:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <service-guardian>
         <timeout-milliseconds>120000</timeout-milliseconds>
      </service-guardian>
   </cluster-config>
</coherence>
```

The `<timeout-milliseconds>` value can also be set using the `tangosol.coherence.guard.timeout` system property.

Setting the Guardian Timeout Per Service Type

To set the guardian timeout per service type, override the service's `guardian-timeout` initialization parameter in an operational override file. The following example sets the guardian timeout for the `DistributedCache` service to `120000` milliseconds:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <services>
         <service id="3">
            <init-params>
               <init-param id="17">
                  <param-name>guardian-timeout</param-name>
                  <param-value>120000</param-value>
               </init-param>
```

```
            </init-params>
         </service>
      </services>
   </cluster-config>
</coherence>
```

The `guardian-timeout` initialization parameter can be set for the `DistributedCache`, `ReplicatedCache`, `OptimisticCache`, `Invocation`, and `Proxy` services. Refer to the `tangosol-coherence.xml` file that is located in the `coherence.jar` file for the correct service ID and initialization parameter ID to use when overriding the `guardian-timeout` parameter for a service.

Each service also has a system property that sets the guardian timeout, respectively:

```
tangosol.coherence.distributed.guard.timeout
tangosol.coherence.replicated.guard.timeout
tangosol.coherence.optimistic.guard.timeout
tangosol.coherence.invocation.guard.timeout
tangosol.coherence.proxy.guard.timeout
```

Setting the Guardian Timeout Per Service Instance

To set the guardian timeout per service instance, add a `<guardian-timeout>` element to a cache scheme definition in the cache configuration file. The following example sets the guardian timeout for a distributed cache scheme to `120000` milliseconds.

```
<distributed-scheme>
   <scheme-name>example-distributed</scheme-name>
   <service-name>DistributedCache</service-name>
   <guardian-timeout>120000</guardian-timeout>
   <backing-map-scheme>
      <local-scheme>
         <scheme-ref>example-binary-backing-map</scheme-ref>
      </local-scheme>
   </backing-map-scheme>
   <autostart>true</autostart>
</distributed-scheme>
```

The `<guardian-timeout>` element can be used in the following schemes: `<distributed-scheme>`, `<replicated-scheme>`, `<optimistic-scheme>`, `<transaction-scheme>`, `<invocation-scheme>`, and `<proxy-scheme>`.

Using the Timeout Value From the PriorityTask API

Custom implementations of the `Invocable`, `EntryProcessor`, and `EntryAggregator` interface can implement the `com.tangosol.net.PriorityTask` interface. In this case, the service guardian attempts recovery after the task has been executing for longer than the value returned by `getExecutionTimeoutMillis()`. See Chapter 29, "Priority Tasks," for more information on using the API.

The execution timeout can be set using the `<task-timeout>` element within an `<invocation-scheme>` element defined in the cache configuration file. For the `Invocation` service, the `<task-timeout>` element specifies the timeout value for `Invocable` tasks that implement the `PriorityTask` interface, but do not explicitly specify the execution timeout value; that is, the `getExecutionTimeoutMillis()` method returns `0`.

If the <task-timeout> element is set to 0, the default guardian timeout is used. See Appendix B, "Cache Configuration Elements" for more information on the different cache schemes that support the use of the <task-timeout> element.

Setting the Guardian Service Failure Policy

The service failure policy determines the corrective action that the service guardian takes after it concludes that a thread is deadlocked. The following policies are available:

- exit-cluster – This policy attempts to recover threads that appear to be unresponsive. If the attempt fails, an attempt is made to stop the associated service. If the associated service cannot be stopped, this policy causes the local node to stop the cluster services. This is the default policy if no policy is specified.
- exit-process – This policy attempts to recover threads that appear to be unresponsive. If the attempt fails, an attempt is made to stop the associated service. If the associated service cannot be stopped, this policy cause the local node to exit the JVM and terminate abruptly.
- logging – This policy logs any detected problems but takes no corrective action.
- custom – the name of a Java class that provides an implementation for the com.tangosol.net.ServiceFailurePolicy interface. See "Enabling a Custom Guardian Failure Policy" on page 30-6.

The service guardian failure policy can be set three different ways based on the level of granularity that is required:

- All threads – This option allows a single failure policy to be applied to all Coherence-owned threads on a cluster node. This is the out-of-box configuration.
- Threads per service type – This option allows different failure policies to be set for specific service types. The policy is applied to the threads of all service instances. If a policy is not specified for a particular service type, then the timeout defaults to the timeout that is set for all threads.
- Threads per service instance – This option allows different failure policies to be set for specific service instances. If a policy is not set for a specific service instance, then the service's policy, if specified, is used; otherwise, the policy that is set for all threads is used.

Setting the Guardian Failure Policy for All Threads

To set a guardian failure policy, add a <service-failure-policy> element to an operational override file within the <service-guardian> element. The following example sets the failure policy to exit-process:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <service-guardian>
         <service-failure-policy>exit-process</service-failure-policy>
      </service-guardian>
   </cluster-config>
</coherence>
```

Setting the Guardian Failure Policy Per Service Type

To set the failure policy per service type, override the service's `service-failure-policy` initialization parameter in an operational override file. The following example sets the failure policy for the `DistributedCache` service to the `logging` policy:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <services>
         <service id="3">
            <init-params>
               <init-param id="18">
                  <param-name>service-failure-policy</param-name>
                  <param-value>logging</param-value>
              </init-param>
            </init-params>
         </service>
      </services>
   </cluster-config>
</coherence>
```

The `service-failure-policy` initialization parameter can be set for the `DistributedCache`, `ReplicatedCache`, `OptimisticCache`, `Invocation`, and `Proxy` services. Refer to the `tangosol-coherence.xml` file that is located in the `coherence.jar` file for the correct service ID and initialization parameter ID to use when overriding the `service-failure-policy` parameter for a service.

Setting the Guardian Failure Policy Per Service Instance

To set the failure policy per service instance, add a `<service-failure-policy>` element to a cache scheme definition in the cache configuration file. The following example sets the failure policy to `logging` for a distributed cache scheme:

```
<distributed-scheme>
   <scheme-name>example-distributed</scheme-name>
   <service-name>DistributedCache</service-name>
   <guardian-timeout>120000</guardian-timeout>
   <service-failure-policy>logging</service-failure-policy>
   <backing-map-scheme>
      <local-scheme>
         <scheme-ref>example-binary-backing-map</scheme-ref>
      </local-scheme>
   </backing-map-scheme>
   <autostart>true</autostart>
</distributed-scheme>
```

The `<service-failure-policy>` element can be used in the following schemes: `<distributed-scheme>`, `<replicated-scheme>`, `<optimistic-scheme>`, `<transaction-scheme>`, `<invocation-scheme>`, and `<proxy-scheme>`.

Enabling a Custom Guardian Failure Policy

To use a custom failure policy, include an `<instance>` subelement and provide a fully qualified class name that implements the `ServiceFailurePolicy` interface. See "instance" on page A-25 for detailed instructions on using the `<instance>`

element. The following example enables a custom failure policy that is implemented in the `MyFailurePolicy` class. Custom failure policies can be enabled for all threads (as shown below) or can be enabled per service instance within a cache scheme definition.

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <service-guardian>
         <service-failure-policy>
            <instance>
               <class-name>package.MyFailurePolicy</class-name>
            </instance>
         </service-failure-policy>
      </service-guardian>
   </cluster-config>
</coherence>
```

As an alternative, the `<instance>` element supports the use of a `<class-factory-name>` element to use a factory class that is responsible for creating `ServiceFailurePolicy` instances, and a `<method-name>` element to specify the static factory method on the factory class that performs object instantiation. The following example gets a custom failure policy instance using the `getPolicy` method on the `MyPolicyFactory` class.

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <service-guardian>
         <service-failure-policy>
            <instance>
               <class-factory-name>package.MyPolicyFactory</class-factory-name>
               <method-name>getPolicy</method-name>
            </instance>
         </service-failure-policy>
      </service-guardian>
   </cluster-config>
</coherence>
```

Any initialization parameters that are required for an implementation can be specified using the `<init-params>` element. The following example sets the `iMaxTime` parameter to `2000`.

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
   <cluster-config>
      <service-guardian>
         <service-failure-policy>
```

```
            <instance>
               <class-name>package.MyFailurePolicy</class-name>
               <init-params>
                  <init-param>
                     <param-name>iMaxTime</param-name>
                     <param-value>2000</param-value>
                  </init-param>
               </init-params>
            </instance>
         </service-failure-policy>
      </service-guardian>
   </cluster-config>
</coherence>
```

Issuing Manual Guardian Heartbeats

The `com.tangosol.net.GuardSupport` class provides heartbeat methods that applications can use to manually issue heartbeats to the guardian:

```
GuardSupport.heartbeat();
```

For known long running operations, the heartbeat can be issued with the number of milliseconds that should pass before the operation is considered "stuck:"

```
GuardSupport.heartbeat(long cMillis);
```

31

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 do not have 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 31–1 shows the implementation of a simple custom eviction policy:

Example 31–1 Implementing a Custom Eviction Policy

```
package com.tangosol.examples.eviction;

import com.tangosol.net.cache.AbstractEvictionPolicy;
import com.tangosol.net.cache.ConfigurableCacheMap;
import com.tangosol.net.cache.LocalCache;
import com.tangosol.net.BackingMapManagerContext;
import com.tangosol.util.ConverterCollections;
import java.util.Iterator;
import java.util.Map;

/**
 * Custom eviction policy that evicts items randomly (or more specifically,
 * based on the natural order provided by the map's iterator.)
 * This example may be used in cases where fast eviction is required
 * with as little processing as possible.
 */
public class SimpleEvictionPolicy
        extends AbstractEvictionPolicy
    {
    /**
     * Default constructor; typically used with local caches or the front
     * parts of near caches.
     */
    public SimpleEvictionPolicy()
        {
        }

    /**
     * Constructor that accepts {@link BackingMapManagerContext}; should
     * be used with partitioned cache backing maps.
```

```
 *
 * @param ctx  backing map context
 */
public SimpleEvictionPolicy(BackingMapManagerContext ctx)
    {
    m_ctx = ctx;
    }

/**
 * {@inheritDoc}
 */
public void entryUpdated(ConfigurableCacheMap.Entry entry)
    {
    }

/**
 * {@inheritDoc}
 */
public void entryTouched(ConfigurableCacheMap.Entry entry)
    {
    }

/**
 * {@inheritDoc}
 */
public void requestEviction(int cMaximum)
    {
    ConfigurableCacheMap cache = getCache();
    Iterator iter  = cache.entrySet().iterator();

    for (int i = 0, c = cache.getUnits() - cMaximum; i < c && iter.hasNext();
       i++)
        {
        ConfigurableCacheMap.Entry entry  = (ConfigurableCacheMap.Entry)
           iter.next();
        StringBuffer               buffer = new StringBuffer();

        // If the contents of the entry (for example the key/value) need
        // to be examined, invoke convertEntry(entry) in case
        // the entry must be deserialized
        Map.Entry convertedEntry = convertEntry(entry);
        buffer.append("Entry: ").append(convertedEntry);

        // Here's how to get metadata about creation/last touched
        // timestamps for entries.  This information might be used
        // in determining what gets evicted.
        if (entry instanceof LocalCache.Entry)
            {
            buffer.append(", create millis=");
            buffer.append(((LocalCache.Entry) entry).getCreatedMillis());
            }
        buffer.append(", last touch millis=");
        buffer.append(entry.getLastTouchMillis());

        // This output is for illustrative purposes; this may generate
        // excessive output in a production system
        System.out.println(buffer);

        // iterate and remove items
        // from the cache until below the maximum.  Note that
```

```
            // the non converted entry key is passed to the evict method
            cache.evict(entry.getKey());
            }
        }

    /**
     * If a {@link BackingMapManagerContext} is configured, wrap the
     * Entry with {@link ConverterCollections.ConverterEntry} in order
     * to deserialize the entry.
     *
     * @see ConverterCollections.ConverterEntry
     * @see BackingMapManagerContext
     *
     * @param entry  entry to convert if necessary
     *
     * @return an entry that deserializes its key and value if necessary
     */
    protected Map.Entry convertEntry(Map.Entry entry)
        {
        BackingMapManagerContext ctx = m_ctx;
        return ctx == null ? entry :
                new ConverterCollections.ConverterEntry(entry,
                        ctx.getKeyFromInternalConverter(),
                        ctx.getValueFromInternalConverter(),
                        ctx.getValueToInternalConverter());
        }

    private BackingMapManagerContext m_ctx;
    }
```

Example 31–2 illustrates a Coherence cache configuration file (`coherence-cache-config.xml`) with an eviction policy:

Example 31–2 Custom Eviction Policy in a coherence-cache-config.xml File

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
   <caching-scheme-mapping>
      <cache-mapping>
         <cache-name>*</cache-name>
         <scheme-name>example-near</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <near-scheme>
         <scheme-name>example-near</scheme-name>
         <front-scheme>
            <local-scheme>
               <eviction-policy>
                  <class-scheme>
                     <class-name>
                        com.tangosol.examples.eviction.SimpleEvictionPolicy
                     </class-name>
                  </class-scheme>
               </eviction-policy>
```

```
                  <high-units>1000</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>

         <distributed-scheme>
            <scheme-name>example-distributed</scheme-name>
            <service-name>DistributedCache</service-name>
            <backing-map-scheme>
               <local-scheme>
                  <eviction-policy>
                     <class-scheme>
                        <class-name>
                           com.tangosol.examples.eviction.SimpleEvictionPolicy
                        </class-name>
                        <init-params>
                     <!--
                      Passing the BackingMapManagerContext to the eviction policy;
                      this is required for deserializing entries
                      -->
                              <init-param>
                                 <param-type>
                                 com.tangosol.net.BackingMapManagerContext</param-type>
                                 <param-value>{manager-context}</param-value>
                              </init-param>
                        </init-params>
                     </class-scheme>
                  </eviction-policy>
                  <high-units>20m</high-units>
                  <unit-calculator>binary</unit-calculator>
               </local-scheme>
            </backing-map-scheme>
            <autostart>true</autostart>
         </distributed-scheme>
      </caching-schemes>
   </cache-config>
```

32

Constraints on Re-entrant Calls

The Coherence architecture is based on 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, 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.

The following sections are included in this chapter:

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 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 requirement 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 enough concurrent calls could fill a thread pool, 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
- 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 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.

A

Operational Configuration Elements

This appendix provides a detailed reference of the operational deployment descriptor elements and briefly describes the deployment descriptor files in which these elements can appear.

The following sections are included in this appendix:

Operational Deployment Descriptor

The `tangosol-coherence.xml` operational deployment descriptor specifies the operational and run-time settings that control clustering, communication, and data management services. The operational deployment descriptor is located in the root of the `coherence.jar` library. A custom `tangosol-coherence.xml` file can be created; however, the preferred approach to changing the operational settings is to use a `tangosol-coherence-override.xml` operational override file as described in "Operational Override File" below.

The operational deployment descriptor schema is defined in the `coherence-operational-config.xsd` file, which imports the `coherence-operational-config-base.xsd` file, which, in turn, implicitly imports the `coherence-config-base.xsd` file. The operational deployment descriptor schema file is located in the root of the `coherence.jar` library and at the following Web URL:

`http://xmlns.oracle.com/coherence/coherence-operational-config/1.0/coherence-operational-config.xsd`

The `<cohrence>` element is the root element of the operational descriptor and includes an XSD and Coherence namespace reference and the location of the `coherence-operational-config.xsd` file. For example:

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

<coherence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/
   coherence-operational-config coherence-operational-config.xsd">
```

Notes:

- The schema located in the `coherence.jar` library is always used at run time even if the `xsi:schemaLocation` attribute references the Web URL.
- The `xsi:schemaLocation` attribute can be omitted to disable schema validation.
- When deploying Coherence into environments where the default character set is EBCDIC rather than ASCII, ensure that the deployment descriptor file is in ASCII format and is deployed into its run-time environment in the binary format.

Operational Override File

The preferred approach for configuring operational settings is to use an operational override file. The schema for the override file and the operational deployment descriptor are the same except that all elements are optional. Any missing elements are loaded from the `tangosol-coherence.xml` operational deployment descriptor. The default name for the override file is `tangosol-coherence-override.xml`. At run time, this file must be found in the classpath before the `coherence.jar` library.

Additional override files can be configured using the `xml-override` attribute within the `<coherence>` element. This allows for additional fine tuning between similar deployment environments such as staging and production. For an example of this feature, see the `tangosol-coherence-override-eval.xml`, `tangosol-coherence-override-dev.xml`, and `tangosol-coherence-override-prod.xml` files within `coherence.jar`. See "Attribute Reference" on page A-85 for details on using the `xml-override` attribute.

Element Reference

Table A–1 lists all non-terminal operational configuration elements.

Table A–1 Non-Terminal Operational Configuration Elements

Element	Used in
access-controller	security-config
address-provider	well-known-addresses
authorized-hosts	cluster-config
cache-factory-builder-config	coherence
callback-handler	security-config
cluster-config	coherence
cluster-quorum-policy	cluster-config
coherence	*root element*
configurable-cache-factory-config	coherence
flashjournal-manager	journaling-config
flow-control	packet-delivery
host-range	authorized-hosts
identity-asserter	security-config
identity-manager	ssl
identity-transformer	security-config
incoming-message-handler	cluster-config
init-param	init-params
init-params	access-controller, address-provider, callback-handler, configurable-cache-factory-config, service
instance	socket-provider, service-failure-policy
journaling-config	cluster-config
key-store	identity-manager, trust-manager
license-config	coherence
logging-config	coherence
management-config	coherence
mbean	mbeans
mbeans	management-config
mbean-filter	management-config
member-identity	cluster-config
multicast-listener	cluster-config
notification-queueing	packet-publisher
outgoing-message-handler	cluster-config
outstanding-packets	flow-control

Table A–1 (Cont.) Non-Terminal Operational Configuration Elements

Element	Used in
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
provider	ssl, identity-manager, trust-manager
ramjournal-manager	journaling-config
reporter	management-config
security-config	coherence
serializer	serializers
serializers	cluster-config
service-guardian	cluster-config
service	services
services	cluster-config
shutdown-listener	cluster-config
socket-address	well-known-addresses
socket-provider	socket-providers, unicast-listener
socket-providers	cluster-config
ssl	socket-provider
tcp-ring-listener	cluster-config
traffic-jam	packet-publisher
trust-manager	ssl
unicast-listener	cluster-config
volume-threshold	packet-speaker
well-known-addresses	unicast-listener

access-controller

Used in: `security-config`.

Description

The `access-controller` element contains the configuration information for the class that implements the `com.tangosol.net.security.AccessController` interface, which is used by the Coherence Security Framework to check access right and encrypt/decrypt node-to-node communications.

Elements

Table A–2 describes the subelements of the `access-controller` element.

Table A–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 is used by the security framework to check access rights for clustered resources and encrypt/decrypt node-to-node communications regarding those rights. See *Oracle Coherence Security Guide* for more information on using an access controller. The 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>`
		
		The The preconfigured system property overrides 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 the subelements of the `init-param` element, see "init-param" on page A-22.

address-provider

Used in: well-known-addresses

Description

Contains the configuration information for an address factory that implements the com.tangosol.net.AddressProvider interface.

Elements

Table A–3 describes the subelements of the address-provider element.

Table A–3 address-provider Subelements

Element	Required/ Optional	Description
<class-name>	Optional	Specifies the fully qualified name of a class that implements the com.tangosol.net.AddressProvider interface. This element cannot be used with the <class-factory-name> element.
<class-factory-name>	Optional	Specifies the fully qualified name of a factory class for creating address provider instances. The instances must implement the com.tangosol.net.AddressProvider interface. This element cannot be used with the <class-name> element and is used with the <method-name> element.
<method-name>	Optional	Specifies the name of a static factory method on the factory class which performs 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. Initialization parameters can be specified for both the <class-name> element and the <class-factory-name> element.

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 A–4 describes the subelements of the `authorized-hosts` element.

Table A–4 authorized-hosts Subelements

Element	Required/ Optional	Description
`<host-address>`	Optional	Specifies an IP address or host name. If any are specified, only hosts with specified host-addresses or within the specified host-ranges is 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 is 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-filter>`	Optional	Specifies class configuration information for a `com.tangosol.util.Filter` implementation that is used by the cluster to determine whether to accept a new cluster member. The `evaluate()` method is passed the `java.net.InetAddress` of the client. Implementations should return `true` to allow the new member to join the cluster.

cache-factory-builder-config

Used in: coherence

Description

The `cache-factory-builder-config` element contains the configuration information for constructing an instance of the `com.tangosol.net.CacheFactoryBuilder` interface. The default implementation is the `com.tangosol.net.DefaultCacheFactoryBuilder` class, which can be extended in advanced use-cases to provide further domain-specific logic for creating and managing `ConfigurableCacheFactory` instances.

A custom `CacheFactoryBuilder` implementation is used to build and manage multiple cache factory configurations across multiple class loaders. This is an advanced use case that allows applications that are scoped by different class loaders to use separate cache configuration files (as is the case with JavaEE and OSGI). For example, the following code uses a custom `ConfigurableCacheFactory` implementation from two classloaders.

```
CacheFactoryBuilder cfb = CacheFactory.getCacheFactoryBuilder();

//load the first configuration
cfb.getConfigurableCacheFactory("example-config.xml", loader0);
CacheFactory.ensureCluster();
NamedCache cache = CacheFactory.getCache("dist-example");

//load the second configuration
cfb.getConfigurableCacheFactory("example-config1.xml", loader1);
CacheFactory.ensureCluster();
NamedCache cache1 = CacheFactory.getCache("dist-example1");
```

Elements

Table A–5 describes the subelements of the `cache-factory-builder-config` element.

Table A–5 cache-factory-builder-config Subelements

Element	Required/ Optional	Description
`<class-name>`	Optional	Specifies the name of a Java class that implements the `com.tagosol.net.CacheFactoryBuilder` interface. The default value is `com.tangosol.net.DefaultCacheFactoryBuilder`.
`<init-params>`	Optional	Contains initialization parameters for the cache factory builder implementation.
`<scope-resolver>`	Optional	Specifies the configuration information for a class that implements the `com.tangosol.net.ScopeResolver` interface. A scope resolver implementation provides the ability to modify the scope name for a given `ConfigurableCacheFactory` at run time to enforce (or disable) isolation between applications running in the same cluster. The custom scope resolver implementation is specified within an `<class-name>` subelement. See *Oracle Coherence Java API Reference* for details on the `ScopeResolver` interface. See the `<scope-name>` subelement of the `<cache-config>` element on page B-20 for details on specifying a scope name within a cache configuration file.

callback-handler

Used in: security-config.

Table A–6 describes the subelements of the `callback-handler` element.

Table A–6 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.

cluster-config

Used in: <coherence>

Description

Contains the cluster configuration information, including communication and service parameters.

Elements

Table A–7 describes the subelements of the `cluster-config` element.

Table A–7 cluster-config Subelements

Element	Required/ Optional	Description
<member-identity>	Optional	Specifies detailed identity information that is useful for defining the location and role of the cluster member.
<unicast-listener>	Required	Specifies the configuration information for the unicast listener, used for receiving point-to-point network communications.
<multicast-listener>	Required	Specifies the configuration information for the multicast listener, used for receiving point-to-multipoint network communications.
<tcp-ring-listener>	Required	Specifies configuration information for the TCP ring listener, used to death detection.
<shutdown-listener>	Required	Specifies the action to take upon receiving an external shutdown request.
<service-guardian>	Required	Specifies the configuration information for the service guardians, used for detecting and resolving service deadlock.
<packet-speaker>	Required	Specifies configuration information for the packet speaker, used for network data transmission.
<packet-publisher>	Required	Specifies configuration information for the packet publisher, used for managing network data transmission.
<incoming-message-handler>	Required	Specifies configuration information for the incoming message handler, used for dispatching incoming cluster communications.
<outgoing-message-handler>	Required	Specifies configuration information for the outgoing message handler, used for dispatching outgoing cluster communications.
<authorized-hosts>	Optional	Specifies the hosts which are allowed to join the cluster.
<services>	Required	Specifies the declarative data for all available Coherence services.
<serializers>	Optional	Specifies any number of serializer class configurations that implement `com.tangosol.io.Serializer`.
<socket-providers>	Required	Contains socket provider definitions.
<cluster-quorum-policy>	Optional	Contains the configuration information for the quorum-based action policy for the Cluster service.
<journaling-config>	Optional	Specifies configuration for the journaling subsystem.

cluster-quorum-policy

Used in: <cluster-config>

Description

The `cluster-quorum-policy` element contains quorum policy settings for the Cluster service.

Element

Table A–8 describes the subelements of the `cluster-quorum-policy` element.

Table A–8 cluster-quorum-policy-scheme Subelements

Element	Required/ Optional	Description
`<timeout-survivor-quorum>`	Optional	Specifies the minimum number of cluster members that must remain to terminate one or more cluster members due to a detected network timeout, irrespective of the root cause. The value must be a nonnegative integer. Use the `role` attribute to specify this value for cluster members of a given role (as defined in the `<role-name>` element). For example: `<timeout-survivor-quorum role="Server">50 </timeout-survivor-quorum>`
`<class-name>`	Optional	Specifies a class that provides custom quorum policies. This element cannot be used with the `<timeout-survivor-quorum>` or the `<class-factory-name>` element. The class must implement the `com.tangosol.net.ActionPolicy` interface. Initialization parameters can be specified using the `<init-params>` element.
`<class-factory-name>`	Optional	Specifies a factory class for creating custom action policy instances. This element cannot be used with the `<timeout-survivor-quorum>` or `<class-name>` elements. This element is used with the `<method-name>` element. The action policy instances must implement the `com.tangosol.net.ActionPolicy` interface. In addition, initialization parameters can be specified using the `<init-params>` element.

coherence

root element

Description

The `coherence` element is the root element of the operational deployment descriptor `tangosol-coherence.xml`.

Elements

Table A–9 describes the subelements of the `coherence` element.

Table A–9 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, which controls from where and how the cache configuration settings are loaded.
<cache-factory-builder-config>	Required	Contains the configuration information for a cache factory builder, which allows building and managing multiple cache factory configurations across multiple class loaders.
<management-config>	Required	Contains the configuration information for the coherence Management Framework. See *Oracle Coherence Management Guide* 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.

configurable-cache-factory-config

Used in: coherence

Description

The configurable-cache-factory-config element contains the configuration information for constructing an instance of the com.tangosol.net.ConfigurableCacheFactory interface. The default implementation is the com.tangosol.net.DefaultConfigurableCacheFactory class.

Using a custom ConfigurableCacheFactory implementation is an advanced use case and is typically used to allow applications that are scoped by different class loaders to use separate cache configuration files (as is the case with JavaEE and OSGI). Typically, the DefaultConfigurableCacheFactory class is extended for such use cases.

The following example loads two configuration files which contain different cache definitions and use different ClassLoaders.

```
//load the first configuration and use a cache

ConfigurableCacheFactory  dccf= new
   DefaultConfigurableCacheFactory("example-config.xml", loader0);
NamedCache cache = dccf.ensureCache("dist-example", loader0);
cache.put(key, value);

//load the second cache configuration and use a cache

ConfigurableCacheFactory  dccf1= new
   DefaultConfigurableCacheFactory("example-config1.xml", loader1);
NamedCache cache1 = dccf1.ensureCache("dist-example1", loader1);
cache1.put(key, value);
```

Note: This example requires each cache definition to use a different service name; otherwise, an exception is thrown indicating that the service was started by a factory with a different configuration descriptor.

Elements

Table A–10 describes the subelements of the configurable-cache-factory-config element.

Table A–10 configurable-cache-factory-config Subelements

Element	Required/ Optional	Description
`<class-name>`	Required	Specifies the name of a Java class that implements the `com.tangosol.net.ConfigurableCacheFactory` interface. The default value is `com.tangosol.net.DefaultConfigurableCacheFactory`.

`<init-params>`	Optional	Contains initialization parameters for the cache configuration factory implementation. For the default cache configuration factory class, a single parameter is used as follows:
		`<init-param>`
		`  <param-type>java.lang.String</param-type>`
		`  <param-value>coherence-cache-config.xml</param-value>`
		`</init-param>`
		
		Unless an absolute or relative path is specified, such as with `./path/to/config.xml`, the application's classpath is used to find the specified descriptor.
		The preconfigured system property override is `tangosol.coherence.cacheconfig`.

flashjournal-manager

Used in: journaling-config

Description

The <flashjournal-manager> element contains the configuration for a flash journal resources manager, which manages I/O for temporary journal-based files to a solid state device.

Elements

Table A–11 describes the subelements of the flashjournal-manager element.

Table A–11 flashjournal-manager Subelements

Element	Required/ Optional	Description
<maximum-value-size>	Optional	Specifies the maximum size, in bytes, of binary values that are to be stored in the flash journal. The value cannot exceed 64MB. The default value is 64MB.
<block-size>	Optional	Specifies the size of the write buffers in which writes to an underlying disk file occur. The size should match or be a multiple of the physical device's optimal block size and must be a power of two. The value must be between 4KB and 1MB. The default value is 256KB.
<maximum-file-size>	Optional	Specifies the maximum file size of the underlying journal files. The value must be a power of two and a multiple of the block size. The value must be between 1MB and 4GB. The default value is 2GB.
<maximum-pool-size>	Optional	Specifies the size, in bytes, for the buffer pool. The size does not limit the number of buffers that can be allocated or that can exist at any point in time. The size only determines the amount of buffers that are recycled. The pools size cannot exceed 1GB. The default value is 16MB.
<directory>	Optional	Specifies the directory where the journal files should be placed. The directory must exist and is not created at run time. If the directory does not exist or is not specified, the JVM/operating system default temporary directory is used. The suggested location is a local flash (SSD) drive. Specifying a directory that is located on a drive which is shared by other applications or system operations increases the potential for unplanned space usage. Use a directory location on a non-shared disk partition to ensure a more predictable environment.
<async-limit>	Optional	Specifies the maximum size, in bytes, of the backlog. The backlog is the amount of data that has yet to be persisted. Client threads are blocked if the configured limit is exceeded and remain blocked until the backlog recedes below the limit. This helps prevent out-of-memory conditions. **Note:** The maximum amount of memory used by the backlog is at least twice the configured amount, since the data is in binary form and rendered to the write-behind buffers. The value must be between 4KB and 1GB. The default value is 16MB.

flow-control

Used in: packet-delivery.

Description

The flow-control element contains configuration information related to packet throttling and remote GC detection.

Elements

Table A–12 describes the subelements of the `flow-control` element.

Table A–12 flow-control Subelements

Element	Required/ Optional	Description
<enabled>	Optional	Specifies if flow control is enabled. The default value is `true`
<pause-detection>	Optional	Defines the number of packets that are resent to an unresponsive cluster node after which the node is assumed to be paused.
<outstanding-packets>	Optional	Defines the number of unconfirmed packets that are sent to a cluster node before packets addressed to that node are 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 A–13 describes the subelements of each `host-range` element.

Table A–13 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.

identity-asserter

Used in: security-config

Description

The <identity-asserter> element contains the configuration information for a class that implements the com.tangosol.net.security.IdentityAsserter interface. The class is called to validate an identity token to establish a user's identity and is used on a Coherence*Extend proxy server. The identity asserter is used with an identity transformer (used on a Coherence*Extend client) to ensure that only valid clients are allowed to connect to an extend proxy.

Elements

Table A–14 describes the subelements of the <identity-asserter> element.

Table A–14 identity-asserter Subelements

Element	Required/ Optional	Description
<class-name>	Optional	Specifies a class that implements com.tangosol.net.security.IdentityAsserter. This element cannot be used with the <class-factory-name> element.
<class-factory-name>	Optional	Specifies a factory class for creating asserter instances. The instances must implement com.tangosol.net.security.IdentityAsserter. This element cannot be used with the <class-name> element. This element can be used with the <method-name> element.
<method-name>	Optional	Specifies the name of a static factory method on the factory class which performs object instantiation.
<init-params>	Optional	Contains class initialization parameters for the asserter implementation.

identity-manager

Used in: ssl.

Description

The <identity-manager> element contains the configuration information for initializing a javax.net.ssl.KeyManager instance.

The identity manager is responsible for managing the key material which is used to authenticate the local connection to its peer. If no key material is available, the connection cannot present authentication credentials.

Elements

Table A–15 describes the subelements of the identity-manager element.

Table A–15 identity-manager Subelements

Element	Required/ Optional	Description
<algorithm>	Optional	Specifies the algorithm used by the identity manager. The default value is SunX509.
<provider>	Optional	Specifies the configuration for a security provider instance.
<key-store>	Optional	Specifies the configuration for a key store implementation.
<password>	Required	Specifies the private key password.

identity-transformer

Used in: security-config

Description

The <identity-transformer> element contains the configuration information for a class that implements the com.tangosol.net.security.IdentityTransformer interface. The class is called to transform a Subject (Principal in .NET) to a token that asserts identity and is used on a Coherence*Extend client. The identity transformer is used with an identity asserter (used on a Coherence*Extend proxy server) to ensure that only valid clients are allowed to connect to an extend proxy.

Elements

Table A–16 describes the subelements of the <identity-transformer> element.

Table A–16 identity-transformer Subelements

Element	Required/ Optional	Description
<class-name>	Optional	Specifies a class that implements com.tangosol.net.security.IdentityTransformer. This element cannot be used with the <class-factory-name> element.
<class-factory-name>	Optional	Specifies a factory class for creating asserter instances. The instances must implement com.tangosol.net.security.IdentityTransformer. This element cannot be used with the <class-name> element. This element can be used with the <method-name> element.
<method-name>	Optional	Specifies the name of a static factory method on the factory class which performs object instantiation.
<init-params>	Optional	Contains class initialization parameters for the transformer implementation.

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 A–17 describes the subelements of the incoming-message-handler element.

Table A–17 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 is closer between multiple systems running in the cluster; however, the process of joining the cluster is 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 a larger value may be necessary. The default value is 16.
<packet-pool>	Required	Specifies how many incoming packets Coherence buffers before blocking.
<use-nack-packets>	Required	Specifies whether the packet receiver uses negative acknowledgments (packet requests) to pro-actively respond to known missing packets. See "notification-queueing" on page A-41 for additional details and configuration. Legal values are true or false. The default value is true.
<priority>	Required	Specifies a priority of the incoming message handler execution thread. Legal values are from 1 to 10. The default value is 7.

init-param

Used in: `init-params`.

Description

Defines an individual initialization parameter.

Elements

Table A–18 describes the subelements of the `init-param` element.

Table A–18 init-param Subelement

Element	Required/ Optional	Description
<param-name>	Optional	Specifies the name of the initialization parameter. For example: <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> The <param-name> element cannot be specified if the <param-type> element is specified. See "Initialization Parameter Settings" on page A-59 for information on the pre-defined parameters that are available for specific services.
<param-type>	Optional	Specifies the Java type of the initialization parameter.The following standard types are supported: ■ java.lang.String (string) ■ java.lang.Boolean (boolean) ■ java.lang.Integer (int) ■ java.lang.Long (long) ■ java.lang.Double (double) ■ java.math.BigDecimal ■ java.io.File ■ java.sql.Date ■ java.sql.Time ■ java.sql.Timestamp For example: <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> The <param-type> element cannot be specified if the <param-name> element is specified.
<param-value>	Required	Specifies the value of the initialization parameter. The value is in the format specific to the Java type of the parameter.
<description>	Optional	Specifies a description for the initialization parameter.

init-params

Used in: address-provider, service, configurable-cache-factory-config, access-controller, and callback-handler.

Description

Defines a series of initialization parameters.

Elements

Table A–19 describes the subelements of the init-params element.

Table A–19 init-params Subelement

Element	Required/ Optional	Description
<init-param>	Optional	Defines an individual initialization parameter.

instance

Used in: socket-provider, service-failure-policy, scope-resolver, and partition-assignment-strategy

Description

The `<instance>` element contains the configuration of an implementation class or class factory that is used to plug in custom functionality.

Elements

Table A–20 describes the subelements of the `instance` element.

Table A–20 instance Subelements

Element	Required/ Optional	Description
`<class-name>`	Optional	Specifies the fully qualified name of an implementation class. This element cannot be used with the `<class-factory-name>` element.
`<class-factory-name>`	Optional	Specifies the fully qualified name of a factory class for creating implementation class instances. This element cannot be used with the `<class-name>` element and is used with the `<method-name>` element.
`<method-name>`	Optional	Specifies the name of a static factory method on the factory class which performs object instantiation.
`<init-params>`	Optional	Contains class initialization parameters for the implementation class.

journaling-config

Used in: cluster-config

Description

The <journaling-config> element contains the configuration for the resource managers that are responsible for storing data in a binary format to flash and RAM memory.

Elements

Table A–20 describes the subelements of the journaling-config element.

Table A–21 journaling-config Subelements

Element	Required/ Optional	Description
<ramjournal-manager>	Required	Specifies the RAM Journal Resource Manager's configuration.
<flashjournal-manager>	Required	Specifies the Flash Journal Resource Manager's configuration.

key-store

Used in: identity-manager, trust-manager.

Description

The `key-store` element specifies the configuration for a key store implementation to use when implementing SSL. The key store implementation is an instance of the `java.security.KeyStore` class.

Elements

Table A–22 describes the subelements of the `key-store` element.

Table A–22 key-store Subelements

Element	Required/ Optional	Description
`<url>`	Required	Specifies the Uniform Resource Locator (URL) to a key store.
`<password>`	Optional	Specifies the password for the key store.
`<type>`	Optional	Specifies the type of a `java.security.KeyStore` instance. The default value is `JKS`.

license-config

Used in: coherence.

Table A–23 describes the subelements of the `license-config` element.

Table A–23 license-config Subelements

Element	Required/ Optional	Description
`<edition-name>`	Optional	Specifies the product edition that the member uses. This allows multiple product editions to be used within the same cluster, with each member specifying the edition that it is using. Valid values are: `GE` (Grid Edition), `EE` (Enterprise Edition), `SE` (Standard Edition), `RTC` (Real-Time Client), `DC` (Data Client). The default value is `GE`.
`<license-mode>`	Optional	Specifies whether the product is being used in a 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. The default value is `dev`.

logging-config

Used in: coherence.

Elements

The following table describes the subelements of the logging-config element.

Table A–24 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 ■ *file name* 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. The preconfigured system property override is tangosol.coherence.log.
<logger-name>	Optional	Specifies a logger name within chosen logging system that logs Coherence related messages. This value is only used by the JDK and log4j logging systems. The default value is Coherence. The preconfigured system property override is tangosol.coherence.log.logger.
<severity-level>	Required	Specifies which logged messages are emitted to the log destination. The legal values are -1 to 9. No messages are emitted if -1 is specified. More log messages are emitted as the log level is increased. The preconfigured system property override is tangosol.coherence.log.level.
<message-format>	Required	Specifies how messages that have a logging level specified are formatted before passing them to the log destination. The format can include static text and any of the following replaceable parameters: {date}, {uptime}, {product}, {version}, {level}, {thread}, {member}, {location}, {role}, and {text}. The default value is: {date}/{uptime} {product} {version} <{level}> (thread={thread}, member={member}): {text}
<character-limit>	Required	Specifies the maximum number of characters that the logger daemon processes from the message queue before discarding all remaining messages in the queue. All messages that are discarded are summarized by the logging system with a single log entry that details the number of messages that were discarded and their total size. Legal values are positive integers or 0. Zero implies no limit. The default value in production mode is 1048576 and 2147483647 in development mode. The preconfigured system property override is tangosol.coherence.log.limit.

management-config

Used in: coherence.

Elements

Table A–25 describes the subelements of the management-config element.

Table A–25 management-config Subelements

Element	Optional/ Required	Description
<managed-nodes>	Required	Specifies whether a cluster node's JVM has an [in-process] MBean server and if so, whether this node allows management of other nodes' managed objects. Legal values are: ■ none – (default) No MBean server is instantiated on this cluster node. ■ local-only – Manage only MBeans which are local to this cluster node (that is, within the same JVM). ■ remote-only – Manage MBeans on other remotely manageable cluster nodes. See <allowed-remote-management> subelement. ■ all – Manage both local and remotely manageable cluster nodes. See <allowed-remote-management> subelement. The preconfigured system property override is tangosol.coherence.management.
<allow-remote-management>	Required	Specifies whether this cluster node exposes its managed objects to remote MBean server(s). Legal values are: true or false. The default value is true. The preconfigured system property override is tangosol.coherence.management.remote.
<refresh-policy>	Optional	Specifies the method which is used to refresh remote management information. Legal values are: refresh-ahead, refresh-behind or refresh-expired. The default value is refresh-ahead. The preconfigured system property override is tangosol.coherence.management.refresh.policy
<refresh-expiry>	Optional	Specifies the time interval (in milliseconds) after which a remote MBean information is invalidated on the management node. Legal values are strings representing time intervals. The default value is 1s. The preconfigured system property override is tangosol.coherence.management.refresh.expiry
<refresh-timeout>	Optional	Specifies the duration which the management node waits for a response from a remote node when refreshing MBean information. This value must be less than the refresh-expiry interval. Legal values are strings representing time intervals. The default value is 250ms. The preconfigured system property override is tangosol.coherence.management.refresh.timeout

Table A–25 (Cont.) management-config Subelements

Element	Optional/ Required	Description
<read-only>	Optional	Specifies whether the managed objects exposed by this cluster node allow operations that modify run-time attributes. Legal values are: true or false. The default value is false. The preconfigured system property override is tangosol.coherence.management.readonly.
<default-domain-name>	Optional	Specifies the default domain name for the MBean server that is used to register MBeans exposed by the Coherence management framework. This value is only used by the cluster nodes that have an in-process MBean server and allow management of local or other node's managed objects. If this value is not specified, the first existing MBean server is used. The element should only be used to identify an existing MBean server which Coherence should use to register MBeans. This element is also used when implementing the MBeanServerFinder interface. See the <server-factory> element below.
<service-name>	Optional	Specifies the name of the Invocation Service used for remote management. This element is used only if allow-remote-management is set to true.
<server-factory>	Optional	Contains the configuration information for the MBeanServer factory that implements the com.tangosol.net.management.MBeanServerFinder interface, which is used to find an MBean server that is used by the Coherence JMX framework to register new or locate existing MBeans. The class name is entered using the <class-name> subelement and supports initialization parameters using the <init-params> element.
<mbeans>	Optional	Contains a list of MBeans to be registered when a node joins the cluster.
<mbean-filter>	Optional	Contains the configuration information of a filter class that is used to filter MBeans before they are registered.
<reporter>	Optional	Contains the Reporter's configuration.

mbean

Used in: mbeans

Description

The `mbean` element contains a list of elements to be instantiated and registered with the Coherence management framework.

Elements

Table A–26 describes the subelements of the `mbean` element.

Table A–26 Subelements of mbean

Element	Required/ Optional	Description
`<mbean-class>`	Optional	Specifies the full class name of the standard MBean to instantiate and register with the Coherence management framework. The MBean class must be in the classpath to correctly instantiate. This element cannot be used with the `<mbean-factory>` element or the `<mbean-query>` element.
`<mbean-factory>`	Optional	Specifies the name of a class factory used to obtain MBeans to register with the Coherence management framework. The factory class must be in the classpath to correctly instantiate. This element is used with the `<mbean-accessor>` element. This element cannot be used with the `<mbean-class>` element or the `<mbean-query>` element.
`<mbean-query>`	Optional	Specifies 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 framework. This allows for a single point of consolidation of MBeans for the grid. For example, the following query includes all the MBeans under the `java.lang` domain in the Coherence management infrastructure. `<mbean-query>java.lang:*</mbean-query>` This element cannot be used with the `<mbean-class>` element or the `<mbean-factory>` element.
`<mbean-server-domain>`	Optional	Specifies the name of a default domain for the source MBean server. This is used to locate the MBean server where the mbean-query should be executed.
`<mbean-accessor>`	Optional	Specifies the method name on the factory class (specified by the `<mbean-factory>` element) that is used to instantiate the MBean.

Table A–26 (Cont.) Subelements of mbean

Element	Required/ Optional	Description
`<mbean-name>`	Required	Specifies the JMX `ObjectName` prefix for the MBean that is registered with the Coherence management framework. The prefix should be a comma-delimited *`Key=Value`* pair. The Coherence MBean naming convention stipulates that the name should begin with a type/value pair (for example, `type=Platform`).
`<local-only>`	Optional	Specifies whether the MBean is visible across the cluster. Valid values are `true` or `false`. If set to `true`, the MBean is registered only with a local MBean server and is not accessible by other cluster nodes. If set to `false`, the `nodeId=...` key attribute is added to its name and the MBean is visible from any of the managing nodes (nodes that set the `<managed-nodes>` element to values of `all` or `remote-only`). The default value is `false`.
`<enabled>`	Optional	Specifies whether the MBean should be instantiated and registered on this instance. Valid values are `true` or `false`. The default value is `false`.
`<extend-lifecycle>`	Optional	Specifies whether the MBean should extend beyond the node connection life cycle. Valid values are `true` or `false`. If `true`, the MBean maintains the statistics and values across connections (coincides with the JVM life cycle). If `false`, the MBean is destroyed and re-created when a node is disconnected from the grid. The default value is `false`.

mbeans

Used in: management-config

Description

The `mbeans` element is the root element for defining custom mbeans and is the root element of a custom mbean configuration file. It contains a list of mbean elements to be instantiated and registered with the Coherence management framework.

Elements

Table A–27 describes the subelements of the `mbeans` element.

Table A–27 Subelement of mbeans

Element	Required/ Optional	Description
<`mbean`>	Required	Specifies the MBean type, implementation, and `ObjectName` that are instantiated and registered with the Coherence management framework.

mbean-filter

Used in management-config.

Description

The `mbean-filter` element is used to specify a filter that evaluates MBean names before they are registered in the MBean server. The `com.tangosol.net.management.ObjectNameExcludeFilter` class is the default filter and is used to exclude MBeans from being registered based on their JMX object name using standard regex patterns. The list is entered as a list of names separated by any white space characters. The following MBeans are excluded by the out-of-box configuration:

```
<management-config>
   <mbean-filter>
      <class-name>com.tangosol.net.management.ObjectNameExcludeFilter</class-name>
      <init-params>
         <init-param>
            <param-type>string</param-type>
            <param-value system-property="tangosol.coherence.management.exclude">
                 .*type=Service,name=Management,.*
                 .*type=Platform,Domain=java.lang,subType=ClassLoading,.*
                 .*type=Platform,Domain=java.lang,subType=Compilation,.*
                 .*type=Platform,Domain=java.lang,subType=MemoryManager,.*
                 .*type=Platform,Domain=java.lang,subType=Threading,.*
            </param-value>
         </init-param>
      </init-params>
   </mbean-filter>
</management-config>
```

Elements

Table A–43 describes the subelements of the `mbean-filter` element.

Table A–28 mbean-filter Subelements

Element	Required/ Optional	Description
`<class-name>`	Optional	Specifies the name of a filter class for filtering mbeans. This element cannot be used with the `<class-factory-name>` element.
`<class-factory-name>`	Optional	Specifies a factory class for creating filter instances. This element cannot be used with the `<name>` element or the `<class-name>` element. This element can be used with the `<method-name>` element.
`<method-name>`	Optional	Specifies the name of a static factory method on the factory class which performs object instantiation.
`<init-params>`	Optional	Contains class initialization parameters for the filter implementation.

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 A–29 describes the subelements of the `member-identity` element.

Table A–29 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. A cluster name should always be specified for production systems, thus preventing accidental cluster discovery among applications. The preconfigured system property override is `tangosol.coherence.cluster`.
`<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. This element is not currently used to make decisions about data backup location. The preconfigured system property override is `tangosol.coherence.site`.
`<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. This element is not currently used to make decisions about data backup location. The preconfigured system property override is `tangosol.coherence.rack`.
`<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 name is used as the basis for creating a ID, which in turn is used to guarantee that data are backed up on different computers 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. The preconfigured system property override is `tangosol.coherence.machine`.

Table A–29 (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 computer. 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 exists per JVM, and in that situation this name would be redundant. The preconfigured system property override is tangosol.coherence.process.
<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 computer (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. The preconfigured system property override is tangosol.coherence.member.
<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. The preconfigured system property override is tangosol.coherence.role.
<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 are 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 is ejected. Valid values are from 1 to 10. The preconfigured system property override is tangosol.coherence.priority.

message-pool

Used in: outgoing-message-handler

Description

The `<message-pool>` element is used to control how many message buffers are pooled for message transmission. Pooling message buffers relieves the pressure on the JVM garbage collector by pooling the memory resources needed for messaging.

Elements

Table A–30 describes the subelements of the `message-pool` element.

Table A–30 message-pool Subelements

Element	Required/ Optional	Description
`<segments>`	Optional	Specifies the number of segments used by the message pool to store buffers. Each segment stores buffers of a specific size. The buffer size difference between segments is calculated using the `<growth-factor>` element value. The default value is `4`.
`<segment-size>`	Optional	Specifies the maximum size of a single pool segment. The maximum size of the entire pool is the total number of segments times the maximum size of a segment. The default value is `16MB`.
`<min-buffer-size>`	Optional	Specifies the smallest available buffer size to be stored in a segment. This value must be a multiple of 1024. Therefore, the smallest possible buffer is 1024 bytes. The default value is `1KB`.
`<growth-factor>`	Optional	Specifies the rate of growth (as bitwise left shift) between successive segments. The default value is `2`.

multicast-listener

Used in: `cluster-config`.

Description

Specifies the configuration information for the Multicast listener. This element is used to specify the address and port that a cluster uses 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. If you are having difficulties establishing a cluster when using multicast, see *Oracle Coherence Administrator's Guide* for instructions on performing a multicast connectivity test.

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.

Elements

Table A–31 describes the subelements of the `multicast-listener` element.

Table A–31 multicast-listener Subelements

Element	Required /Optional	Description
`<interface>`	Optional	Specifies the IP address that a multicast socket is bound to. By default, the interface (NIC) of the unicast-listener IP address is used for the multicast socket; this option allows a different interface to be specified for multicast. Setting this address to `0.0.0.0` allows the operating system to use the unicast routing table to select the interface automatically.
`<address>`	Required	Specifies the multicast IP address that a Socket listens or publishes on. Legal values are from 224.0.0.0 to 239.255.255.255. The default value depends on the release and build level and typically follows the convention of `{build}.{major version}.{minor version}.{patch}`. The preconfigured system property override is `tangosol.coherence.clusteraddress`.
`<port>`	Required	Specifies the port that the Socket listens or publishes on. Legal values are from 1 to 65535. The default value depends on the release and build level and typically follows the convention of `{version}+{{{build}`. The preconfigured system property override is `tangosol.coherence.clusterport`.
`<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. The default value is `4`. The preconfigured system property override is `tangosol.coherence.ttl`.
`<packet-buffer>`	Required	Specifies how many incoming packets the operating system is requested to buffer. The value may be expressed either in terms of packets or bytes.

Table A–31 (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. The default value is 8.
`<join-timeout-milliseconds>`	Required	Specifies the number of milliseconds that a new member waits without finding any evidence of a cluster before starting its own cluster and electing itself as the senior cluster member. Legal values are from 1000 to 1000000. The default value is `3000`. **Note:** For production use, the recommended value is `30000`.
`<multicast-threshold-percent>`	Required	Specifies the threshold percentage value used to determine whether a packet is 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 is sent multicast if and only if the following both hold true: **1.** The packet is being sent over the network to multiple nodes, 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 allows the implementation to use multicast for basically all multi-point traffic. Setting it to 100 forces 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 is never exceeded. With the setting of 25 the implementation sends 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. Legal values are from 1 to 100. The default value is `25`. **Note:** This element is only used if the `well-known-addresses` element is empty.

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.

Elements

The following table describes the subelements of the notification-queuing element.

Table A–32 notification-queuing Subelements

Element	Required/ Optional	Description
<ack-delay-milliseconds>	Required	Specifies the maximum number of milliseconds that the packet publisher delays 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. The default value is 16.
<nack-delay-milliseconds>	Required	Specifies the number of milliseconds that the packet publisher delays before sending a NACK packet. The default value is 1.

outgoing-message-handler

Used in: cluster-config

Description

The `outgoing-message-handler` element contains the outgoing message handler (also known as a dispatcher) related configuration information.

Elements

Table A–33 describes the subelements of the `outgoing-message-handler` element.

Table A–33 outgoing-message-handler Subelement

Element	Required/ Optional	Description
`<message-pool>`	Optional	Specifies the size of the message buffer pool.

outstanding-packets

Used in: `flow-control`.

Description

Defines the number of unconfirmed packets that are sent to a cluster node before packets addressed to that node are deferred. This helps to prevent the sender from flooding the recipient's network buffers.

Elements

Table A–34 describes the subelements of the `outstanding-packets` element.

Table A–34 outstanding-packets Subelements

Element	Required/ Optional	Description
`<maximum-packets>`	Optional	The maximum number of unconfirmed packets that are sent to a cluster node before packets addressed to that node are deferred. It is recommended that this value not be set below 256. The default value is `4096`.
`<minimum-packets>`	Optional	The lower bound on the range for the number of unconfirmed packets that are sent to a cluster node before packets addressed to that node are deferred. It is recommended that this value not be set below 16. The default value is `64`.

packet-buffer

Used in: unicast-listener, multicast-listener, packet-publisher.

Description

Specifies the size (in packets or bytes) of the operating system buffer for datagram sockets.

Elements

Table A–35 describes the subelements of the packet-buffer element.

Table A–35 packet-buffer Subelements

Element	Required/ Optional	Description
<maximum-packets>	Optional	For unicast-listener, multicast-listener and packet-publisher: Specifies the number of packets of packet-size that the datagram socket are 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. The default values are 32 for publishing, 64 for multicast listening, and 1428 for unicast listening. The <maximum-packets> element cannot be specified if the <size> element is specified.
<size>	Optional	Specifies the requested size of the underlying socket buffer in bytes rather than the number of packets. The <size> element cannot be specified if the <maximum-packets> element is specified.

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.

Elements

Table A–36 describes the subelements of the packet-bundling element.

Table A–36 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 results in the algorithm not waiting, and only bundling the readily accessible packets. A value greater than zero causes 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. The 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 allows for high packet utilization while keeping latency to a minimum. The default value is 0.

packet-delivery

Used in: packet-publisher.

Description

Specifies timing and transmission rate parameters related to packet delivery.

Elements

Table A–37 describes the subelements of the packet-delivery element.

Table A–37 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. The default value is 200.
<timeout-milliseconds>	Required	For packets which require confirmation, specifies the maximum amount of time, in milliseconds, that a packet is resent. After this timeout expires Coherence makes a determination if the recipient is to be considered terminated. This determination takes additional data into account, such as if other nodes are still able to communicate with the recipient. The default value is 300000. For production use, the recommended value is the greater of 300000 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. The default value is 1000.
<flow-control>	Optional	Configures per-node packet throttling and remote GC detection.
<packet-bundling>	Optional	Configures how aggressively Coherence attempts to maximize packet utilization.

packet-pool

Used in: `incoming-message-handler`, `packet-publisher`.

Description

The packet pool is a buffer for use in transmitting and receiving UDP packets. Unlike the packet buffers (see "packet-buffer" on page A-44), these buffers are internally managed by Coherence rather then the operating system and are allocated on the JVM's heap.

Elements

Table A–38 describes the subelements of the `packet-pool` element.

Table A–38 packet-pool Subelements

Element	Required/ Optional	Description
`<size>`	Required	Specifies the maximum size of the pool. The value is entered in bytes. By default, the `<packet-pool>` element is unspecified within the configuration and the size defaults to `0`. For the packet publisher, `0` indicates that the buffer is calculated by factoring the preferred MTU size with `2048`. For the incoming message handler, `0` indicates that the buffer is calculated by factoring the preferred MTU size with `8192`. If the `<packet-pool>` element is specified and a size is defined, then the number of packets is calculated as *`pool size/MTU size`*.

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 is retransmitted (see <packet-delivery> subelement). The recipient node delays the ACK, to batch a series of ACKs into a single response (see <notification-queuing> subelement).

Elements

Table A–39 describes the subelements of the packet-publisher element.

Table A–39 packet-publisher Subelements

Element	Required/ Optional	Description
<packet-size>	Optional	Specifies the UDP packet sizes to use.
<packet-pool>	Required	Specifies how many outgoing packets Coherence buffers before blocking.
<packet-delivery>	Required	Specifies timing parameters related to reliable packet delivery.
<notification-queueing>	Required	Contains the notification queue related configuration info.
<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 is requested to buffer. The value may be expressed either in terms of packets of bytes.
<priority>	Required	Specifies a priority of the packet publisher execution thread. Legal values are from 1 to 10. The default value is 6.
<enabled>	Optional	Specifies if TCMP clustering is enabled. When using 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. The default value is true. The preconfigured system property override is tangosol.coherence.tcmp.enabled.

packet-size

Used in: packet-publisher.

Description

The packet-size element specifies the maximum and preferred UDP packet sizes. All cluster nodes must use identical maximum packet sizes.

Elements

Table A–40 describes the subelements of the packet-size element.

Table A–40 packet-size Subelement

Element	Required/ Optional	Description
<maximum-length>	Required	Specifies the packet size, in bytes, which all cluster members can safely support. This value must be the same for all members in the cluster. A low value can artificially limit the maximum size of the cluster. This value should be at least 512, and defaults to 64KB.
<preferred-length>	Required	Specifies the preferred size, in bytes, of the DatagramPacket objects that are sent and received on the unicast and multicast sockets. This value can be larger or smaller then the <maximum-length> value, and need not be the same for all cluster members. The ideal value is one which fits within the network MTU, leaving enough space for either the UDP or TCP packet headers, which are 32, and 52 bytes respectively. This value should be at least 512, and defaults to a value based on the local nodes MTU. An MTU of 1500 is assumed if the MTU cannot be obtained and results in a value of 1448.

packet-speaker

Used in: `cluster-config`.

Description

Specifies configuration information for the packet speaker which is used for network data transmission.

Elements

Table A–41 describes the subelements of the `packet-speaker` element.

Table A–41 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. The default value is `8`.

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 are 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 A–42 describes the subelements of the `pause-detection` element.

Table A–42 pause-detection Subelements

Element	Required/ Optional	Description
`<maximum-packets>`	Optional	The maximum number of packets that are resent to an unresponsive cluster node after which the node is assumed to be paused. Specifying a value of 0 disables pause detection. The default value is `16`.

provider

Used in: ssl, identity-manager, trust-manager.

Description

The provider element contains the configuration information for a security provider that extends the `java.security.Provider` class.

Elements

Table A–43 describes the subelements of the `provider` element.

Table A–43 provider Subelements

Element	Required/ Optional	Description
`<name>`	Optional	Specifies the name of a security provider that extends the `java.security.Provider` class. The class name can be entered using either this element or by using the `<class-name>` element or by using the `<class-factory-name>` element.
`<class-name>`	Optional	Specifies the name of a security provider that extends the `java.security.Provider` class. This element cannot be used with the `<name>` element or the `<class-factory-name>` element.
`<class-factory-name>`	Optional	Specifies a factory class for creating `Provider` instances. The instances must implement the `java.security.Provider` class. This element cannot be used with the `<name>` element or the `<class-name>` element. This element can be used with the `<method-name>` element.
`<method-name>`	Optional	Specifies the name of a static factory method on the factory class which performs object instantiation.
`<init-params>`	Optional	Contains class initialization parameters for the provider implementation. This element cannot be used with the `<name>` element.

ramjournal-manager

Used in: journaling-config

Description

The <ramjournal-manager> element contains the configuration for a RAM journal resources manager, which manages memory buffers for journal-based storage in-memory. A RAM journal resource manager always uses a flash journal resource manager to store large objects and is also used as an overflow when the amount of total memory allocated to the RAM journal is reached. A RAM journal also uses a flash journal when the journal garbage collection is temporarily not able to keep up with demand. See "flashjournal-manager" on page A-15 for details on configuring a flash journal resource manager.

Elements

Table A–44 describes the subelements of the ramjournal-manager element.

Table A–44 ramjournal-manager Subelements

Element	Required/ Optional	Description
<maximum-value-size>	Optional	Specifies the maximum size, in bytes, of binary values that are to be stored in the RAM journal. The value cannot exceed 4MB. The default value is 64KB. When a flash journal is used to back up a RAM journal, binary values that exceed the maximum value size are automatically delegated to the flash journal.
<maximum-size>	Optional	Specifies the maximum amount of RAM that is used by the journal. The value can either be specified as a percentage of the maximum available heap or as a specific amount of memory. If the value contains a percentage sign (%), it is interpreted as a percentage of the maximum JVM heap (the JVM max heap is typically specified by the -Xmx argument on the java command line). If specified as a specific amount of memory, the value must be between 16MB and 64GB. The default value is 25%. That is, the RAM journal resource manager uses a maximum of 25% of the available JVM heap. A RAM journal is always backed up by a flash journal and all data in excess of the maximum RAM size is automatically delegated to the flash journal.

reporter

Used in: management-config.

Description

The Reporter provides JMX reporting capabilities. The Reporter provides out-of-the-box reports and also supports the creation of custom reports. The reports help administrators and developers manage capacity and trouble shoot problems.

Elements

Table A–45 describes the subelements of the `reporter` element.

Table A–45 reporter Subelements

Element	Required/ Optional	Description
`<configuration>`	Required	Specifies the location for the report group deployment descriptor. The default file is `reports/report-group.xml` and is located in the `coherence.jar` library.
`<autostart>`	Required	Specifies whether the Reporter automatically starts when the node starts. Valid values are `true` and `false`. The default value is `false`.
`<distributed>`	Required	Specifies whether the reporter runs on multiple management nodes. Valid values are `true` and `false`. The default value is `false`.
`<timezone>`	Optional	Specifies the time zone to be used for timestamps that are displayed within a report. See `java.util.TimeZone` for supported time zone formats. The default, if no time zone is specified, is the local time zone.
`<timeformat>`	Optional	Specifies the time and date format to be used for timestamps that are displayed within a report. The value must be a pattern supported by the `java.text.SimpleDateFormat` class. The default value is `EEE MMM dd HH:mm:ss zzz yyyy`.

security-config

Used in: coherence.

Elements

Table A–46 describes the subelements of the security-config element.

Table A–46 security-config Subelements

Element	Required/ Optional	Description
<enabled>	Required	Specifies whether the access controller security feature is enabled. Legal values are true or false. The default value is false. The preconfigured system property override is tangosol.coherence.security.
<login-module-name>	Required	Specifies the name of the JAAS LoginModule that is used to authenticate the caller. This name should match a module in a configuration file is used by the JAAS (for example specified by using the -Djava.security.auth.login.config Java command line attribute). For details, refer to the *Oracle Login Module Developer's Guide*.
<access-controller>	Required	Contains the configuration information for the class that implements com.tangosol.net.security.AccessController interface, which is used by the security framework to check access rights for clustered resources and encrypt/decrypt node-to-node communications regarding those rights.
<callback-handler>	Optional	Contains the configuration information for the class that implements javax.security.auth.callback.CallbackHandler interlace which is called if an attempt is made to access a protected clustered resource when there is no identity associated with the caller.
<identity-asserter>	Optional	Contains the configuration information for a class that implements the com.tangosol.net.security.IdentityAsserter interface which is called to validate an identity token to establish a user's identity. An identity asserter is used with an identity transformer to protect connections between Coherence*Extend clients and proxies.
<identity-transformer>	Optional	Contains the configuration information for the class that implements com.tangosol.net.security.IdentityTransformer interface which is called to transform a Subject (Principal for .NET) to a token that asserts identity. An identity transformer is used with an identity asserter to protect connections between Coherence*Extend clients and proxies.
<subject-scope>	Optional	Specifies whether the remote cache or service reference is shared by subject. Valid values are true or false. Setting the value to true means that remote references are not globally shared; each subject gets a different reference. The default value is false.

serializer

Used in: serializers

Description

The `serializer` element contains a serializer class configuration. Serializer classes must implement `com.tangosol.io.Serializer`. A Java serializer and POF serializer are predefined:

```
<cluster-config>
   <serializers>
      <serializer id="java">
         <class-name>com.tangosol.io.DefaultSerializer</class-name>
      </serializer>

      <serializer id="pof">
         <class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>
        <init-params>
            <init-param>
               <param-type>String</param-type>
               <param-value>pof-config.xml</param-value>
            </init-param>
         </init-params>
      </serializer>
   </serializers>
</cluster-config>
```

Serializer definitions are referenced by individual cache scheme definitions (see "serializer" on page B-100) and can be referenced by the default serializer for services that do not explicitly define a serializer (see "defaults" on page B-29).

Additional serializers can be defined in an operational override file as required.

Elements

Table A–47 describes the subelements of the `serializer` element.

Table A–47 serializer Subelements

Element	Required/ Optional	Description
`<class-name>`	Optional	Specifies a class that implements `com.tangosol.io.Serializer`. This element cannot be used with the `<class-factory-name>` element.
`<class-factory-name>`	Optional	Specifies a factory class for creating custom serializer instances. The instances must implement `com.tangosol.io.Serializer`. This element cannot be used with the `<class-name>` element. This element can be used with the `<method-name>` element.
`<method-name>`	Optional	Specifies the name of a static factory method on the factory class which performs object instantiation.
`<init-params>`	Optional	Contains class initialization parameters for the serializer implementation.

serializers

Used in: cluster-config

Description

The `serializers` element contains the declarative data for each serializer.

Elements

Table A–48 describes the subelements of the `serializers` element.

Table A–48 serializers Subelements

Element	Required/ Optional	Description
`<serializer>`	Optional	Specifies the declarative data of a particular serializer.

service

Used in: services.

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.
- PartitionedService.PartitionedCache—A cache service which evenly partitions cache entries across the cluster nodes which run the service. This service is often referred to as the distributed cache service
- SimpleCache —A version of the ReplicatedCache which lacks concurrency 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.
- ProxyService—A service that accepts connections from Coherence*Extend clients.

Elements

Table A–49 describes the subelements of the services element.

Table A–49 service 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 B-25 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 ■ ProxyService
<init-params>	Optional	Specifies the initialization parameters that are specific to each service. Each parameter is described below.

Initialization Parameter Settings

The <init-param> element in the Coherence operational configuration deployment descriptor defines initialization parameters for a service. The parameters that appear under init-param are different, depending on the service.

The following sections describe the parameters that can be configured for each of the services:

- DistributedCache Service Parameters
- ReplicatedCache Service Parameters
- InvocationService Parameters
- ProxyService Parameters

The tables in each section describes the specific <param-name>/<param-value> pairs that can be configured for each service. The **Parameter Name** column refers to the value of the <param-name> element and **Paramter Value Description** column refers to the possible values for the corresponding <param-value> element.

DistributedCache Service Parameters

DistributedCache <service> elements support the parameters described in Table A–50. Many of these settings may also be specified for each service instance as part of the <distributed-scheme> element in the coherence-cache-config.xml descriptor.

Table A–50 DistributedCache Service Parameters

Parameter Name	Parameter Value Description
`lease-granularity`	Specifies the lease ownership granularity. Legal values are: ■ `thread` (default) ■ `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.

`partition-count`	Specifies the number of partitions that a partitioned (distributed) cache is "chopped up" into. Each member running the partitioned cache service that has the local-storage (`<local-storage>` subelement) option set to `true` manages a "fair" (balanced) number of partitions.
	The number of partitions should be a prime number and sufficiently large such that a given partition is expected to be no larger than 50MB.
	The following are good defaults based on service storage sizes:
	`service storage` `partition-count`
	`100M` `257`
	`1G` `509`
	`10G` `2039`
	`50G` `4093`
	`100G` `8191`
	A list of first 1,000 primes can be found at:
	`http://primes.utm.edu/lists/`
	Valid values are positive integers. The default value is `257`.

`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`. The default value is `true`. The preconfigured system property override is `tangosol.coherence.distributed.localstorage`.
`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 makes the distribution process take longer, but reduces network bandwidth utilization during this activity. Legal values are integers greater then zero. The default value is 0.5MB. The preconfigured system property override is `tangosol.coherence.distributed.transfer`.
`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 for abnormal termination, some portion of the data in the cache is lost. Value of N means that if up to N cluster nodes terminate immediately, the cache data is 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 is in the order of M*(N+1). Recommended values are 0, 1 or 2. The default value is 1.

Table A–50 (Cont.) DistributedCache Service Parameters

Parameter Name	Parameter Value Description
`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. The default value is `0`. Set the value to `0` for scenarios with purely in-memory data (no read-through, write-through, or write-behind) and simple access (no entry processors, aggregators, and so on). For heavy compute scenarios (such as aggregators), the number of threads should be the number of available cores for that compute. For example, if you run 4 nodes on a 16 core box, then there should be roughly 4 threads in the pool. For I/O intensive scenarios (such as read through, write-through, and write-behind), the number of threads must be higher. In this case, increase the threads just to the point that the box is saturated. The preconfigured system property override is `tangosol.coherence.distributed.threads`.
`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.
`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.
`task-hung-threshold`	Specifies the amount of time in milliseconds that a task can execute before it is considered hung. Legal values are positive integers or zero (indicating no default timeout). **Note**: a posted task that has not yet started is never considered 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 do not 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).

Table A–50 (Cont.) DistributedCache Service Parameters

Parameter Name	Parameter Value Description
`request-timeout`	Specifies the maximum amount of time a client waits 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 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. Legal values are positive integers or zero (indicating no default timeout). The default value is an infinite timeout (`0s`) for clustered client requests and 30 seconds (`30s`) for extend client requests.
`serializer`	Specifies a serializer class for object serialization. Serializer classes must implement the `com.tangosol.io.Serializer` interface. The preferred method for specifying a serializer is to define it within the global serializer element and then configure it for a cache within the cache configuration file.
`backup-count-after-writebehind`	Specifies the number of members of the partitioned cache service that holds 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 is backed up on the number of members specified by the `backup-count` parameter. If the unit of storage is not marked as requiring write-behind, then it is backed up by the number of members specified by the `backup-count-after-writebehind` paramter. 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. The value of 0 means that when write-behind has occurred, the backup copies of that data is discarded. However, until write-behind occurs, the data is backed up in accordance with the `backup-count` parameter. Recommended value is 0.

Table A–50 (Cont.) DistributedCache Service Parameters

Parameter Name	Parameter Value Description
guardian-timeout	Specifies the guardian timeout value to use for guarding the service and any dependent threads. If the paramter is not specified, the default guardian timeout (as specified by the <timeout-milliseconds> operational configuration element) is used. See the service-guardian element to globally configure the service guardian for all services. 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.
service-failure-policy	Specifies the action to take when an abnormally behaving service thread cannot be terminated gracefully by the service guardian. See the service-guardian element to globally configure the service guardian for all services. Legal values are: ■ exit-cluster – attempts to recover threads that appear to be unresponsive. If the attempt fails, an attempt is made to stop the associated service. If the associated service cannot be stopped, this policy causes the local node to stop the cluster services. ■ exit-process – attempts to recover threads that appear to be unresponsive. If the attempt fails, an attempt is made to stop the associated service. If the associated service cannot be stopped, this policy cause the local node to exit the JVM and terminate abruptly. ■ logging – causes any detected problems to be logged, but no corrective action to be taken. ■ a custom class – the class configuration information for a com.tangosol.net.ServiceFailurePolicy implementation.
member-listener/ class-name	Specifies the configuration information for a class that implements the com.tangosol.net.MemberListener interface. The implementation must have a public default constructor. The MemberListener implementation receives cache service lifecycle events. The member-listener is used as an alternative to programmatically adding a MapListener on a service.
partitioned-quorum-policy-scheme	Specifies quorum policy settings for the partitioned cache service. See "partitioned-quorum-policy-scheme" on page B-80.
partition-assignment-strategy	Specifies the strategy that is used by a partitioned service to manage partition distribution. Valid values are legacy or a class that implements the com.tangosol.net.partition.PartitionAssignmentStrategy interface. The legacy assignment strategy indicates that partition distribution is managed individually on each cluster member. Whereas; a custom strategy allows for a shared strategy across the cluster. The default value is legacy.

Table A–50 (Cont.) DistributedCache Service Parameters

Parameter Name	Parameter Value Description
`compressor`	Specifies whether or not backup updates should be compressed in delta form or sent whole. A delta update represents the parts of a backup entry that must be changed in order to synchronize it with the primary version of the entry. Valid values are: ■ `none` (default) – Disables delta backup; no compressor is used. The whole backup binary entry is replaced when the primary entry changes. ■ `standard` – Automatically selects a delta compressor based on the serializer being used by the partitioned service. ■ the fully qualified name of a class that implements the `com.tangosol.io.DeltaCompressor` interface. The preconfigured system property override is `tangosol.coherence.distributed.compressor`.
`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/ 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). The default value is `1MB`.
`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`) uses as "root" to store files in. If not specified or specifies a non-existent directory, a temporary file in the default location is used. The default value is the default temporary directory designated by the Java run time.

Table A–50 (Cont.) DistributedCache Service Parameters

Parameter Name	Parameter 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). The 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` – (default) 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`. ■ `file-mapped` – The corresponding implementations class is `com.tangosol.util.nio.BinaryMap` using `com.tangosol.util.nio.MappedBufferManager`. ■ `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. The preconfigured system property override is `tangosol.coherence.distributed.backup`.

ReplicatedCache Service Parameters

ReplicatedCache `service` elements support the parameters described in Table A–51. These settings may also be specified as part of the `replicated-scheme` element in the `coherence-cache-config.xml` descriptor.

Table A–51 ReplicatedCache Service Parameters

Parameter Name	Value Description
`lease-granularity`	Specifies the lease ownership granularity. Available since release 2.3.Legal values are: ■ `thread` (default) ■ `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.
`request-timeout`	Specifies the maximum amount of time a client waits 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 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. Legal values are positive integers or zero (indicating no default timeout). The default value is an infinite timeout (`0s`) for clustered client requests and 30 seconds (`30s`) for extend client requests.
`mobile-issues`	Specifies whether lease issues should be transferred to the most recent lock holders. Legal values are `true` or `false`. The 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 is automatically 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. The default value is `0`.

InvocationService Parameters

`InvocationService` `service` elements support the following parameters listed in Table A–52. These settings may also be specified as part of the `invocation-scheme` element in the `coherence-cache-config.xml` descriptor.

Table A–52 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 do not 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 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. Legal values are positive integers or zero (indicating no default timeout). The default value is an infinite timeout (`0s`) for clustered client requests and 30 seconds (`30s`) for extend client requests.
`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 do not 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. The default value is 0. Set the value to `0` for scenarios with purely in-memory data (no read-through, write-through, or write-behind) and simple access (no entry processors, aggregators, and so on). For heavy compute scenarios (such as aggregators), the number of threads should be the number of available cores for that compute. For example, if you run 4 nodes on a 16 core box, then there should be roughly 4 threads in the pool. For I/O intensive scenarios (such as read through, write-through, and write-behind), the number of threads must be higher. In this case, increase the threads just to the point that the box is saturated. The preconfigured system property override is `tangosol.coherence.invocation.threads`.

ProxyService Parameters

ProxyService `service` elements support the parameters described in Table A–53. These settings may also be specified as part of the `proxy-scheme` element in the `coherence-cache-config.xml` descriptor.

Table A–53 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. The default value is `0`. Proxy service threads perform operations on behalf of the calling application. Therefore, set the value to as many threads as there are concurrent operations that are occurring.
`request-timeout`	Specifies the maximum amount of time a client waits 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 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. Legal values are positive integers or zero (indicating no default timeout). The default value is an infinite timeout (`0s`) for clustered client requests and 30 seconds (`30s`) for extend client requests.
`task-hung-threshold`	Specifies the amount of time in milliseconds that a task can execute before it is considered hung. Legal values are positive integers or zero (indicating no default timeout). **Note**: a posted task that has not yet started is never considered 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 do not 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).
`load-balancer`	Specifies the default load balancing strategy that is used by a proxy service if a strategy is not explicitly configured as part of the proxy scheme. Legal values are: ■ `proxy` – (default) This strategy attempts to distribute client connections equally across proxy service members based upon existing connection count, connection limit, incoming and outgoing message backlog, and daemon pool utilization. ■ `client` – This strategy relies upon the client address provider implementation to dictate the distribution of clients across proxy service members. If no client address provider implementation is provided, the extend client tries each proxy service in a random order until a connection is successful.

service-guardian

Used in: `cluster-config`

Description

Specifies the configuration of the service guardian, which detects and attempts to resolve service deadlocks.

Elements

Table A–54 describes the subelements of the `service-guardian` element.

Table A–54 service-guardian Subelements

Element	Required/ Optional	Description
`<service-failure-policy>`	Optional	Specifies the action to take when an abnormally behaving service thread cannot be terminated gracefully by the service guardian. Legal values are: ■ `exit-cluster` – (default) attempts to recover threads that appear to be unresponsive. If the attempt fails, an attempt is made to stop the associated service. If the associated service cannot be stopped, this policy causes the local node to stop the cluster services. ■ `exit-process` – attempts to recover threads that appear to be unresponsive. If the attempt fails, an attempt is made to stop the associated service. If the associated service cannot be stopped, this policy cause the local node to exit the JVM and terminate abruptly. ■ `logging` – causes any detected problems to be logged, but no corrective action to be taken. ■ a custom class – an `<instance>` subelement is used to provide the class configuration information for a `com.tangosol.net.ServiceFailurePolicy` implementation.
`<timeout-milliseconds>`	Optional	The timeout value used to guard against deadlocked or unresponsive services. It is recommended that `service-guardian/timeout-milliseconds` be set equal to or greater than the `packet-delivery/timeout-milliseconds` value. A timeout of `0` disables service guardians. The default value is `305000`. The preconfigured system property override is `tangosol.coherence.guard.timeout`

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 "Attribute Reference" on page A-85 for more information.

services

Used in: cluster-config

Description

The services element contains the declarative data for each service.

Elements

Table A–49 describes the subelements of the `services` element.

Table A–55 services Subelements

Element	Required/ Optional	Description
<service>	Optional	Specifies the declarative data of a particular service.

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 A–56 describes the subelements of the `shutdown-listener` element.

Table A–56 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` – (default) 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. The preconfigured system property override is `tangosol.coherence.shutdownhook`.

socket-address

Used in: `well-known-addresses`.

Elements

Table A–57 describes the subelements of the `socket-address` element.

Table A–57 socket-address Subelements

Element	Required/ Optional	Description
`<address>`	Required	Specifies the IP address that a Socket listens or publish on. **Note:** The localhost setting may not work on systems that define localhost as the loopback address; in that case, specify the computer name or the specific IP address.
`<port>`	Required	Specifies the port that the Socket listens or publish on. Legal values are from 1 to 65535. When setting up a WKA member, the port value must match the port value that is specified for the unicast listener port. See the `<unicast-listener>` element.

socket-provider

Used in: socket-providers, unicast-listener.

Description

The `<socket-provider>` element contains the configuration information for a socket and channel factory that implements the `com.tangosol.net.SocketProvider` interface. The following pre-defined socket providers are included out-of-box and are referenced using their defined `id` attribute name.

```
<socket-providers>
   <socket-provider id="system">
      <system/>
   </socket-provider>

   <socket-provider id="tcp">
      <tcp/>
   </socket-provider>

   <socket-provider id="ssl">
      <ssl>
         <identity-manager>
            <key-store>
               <url system-property="tangosol.coherence.security.keystore">
                  file:keystore.jks</url>
               <password system-property="tangosol.coherence.security.password"/>
            </key-store>
            <password system-property="tangosol.coherence.security.password"/>
         </identity-manager>
         <trust-manager>
            <algorithm>PeerX509</algorithm>
            <key-store>
               <url system-property="tangosol.coherence.security.keystore">
                  file:keystore.jks</url>
               <password system-property="tangosol.coherence.security.password"/>
            </key-store>
         </trust-manager>
      </ssl>
   </socket-provider>
</socket-providers>
```

Additional socket provider implementations can be created as required. Alternate SSL definitions can be created to support more elaborate SSL configurations.

Elements

Table A–58 describes the subelements of the `socket-provider` element.

Table A–58 socket-provider Subelements

Element	Required/ Optional	Description
`<system>`	Optional	Specifies a socket provider that produces instances of the JVM's default socket and channel implementations. This is the default socket provider.
`<ssl>`	Optional	Specifies a socket provider that produces socket and channel implementations which use SSL.
`<tcp>`	Optional	Specifies a socket provider that produces TCP-based sockets and channel implementations.
`<instance>`	Optional	Contains the class configuration information for a `com.tangosol.net.SocketProvider` implementation.

socket-providers

Used in cluster-config

Description

The `socket-providers` element contains the declarative data for each socket provider implementation. Coherence includes the following pre-defined socket providers: `system`, `tcp`, and `ssl`. Additional socket providers can be created as required.

Elements

Table A–59 describes the subelements of the `socket-providers` element.

Table A–59 socket-providers Subelements

Element	Required/ Optional	Description
`<socket-provider>`	Optional	Specifies the configuration information for a socket and channel factory that implements the `com.tangosol.net.SocketProvider` interface.

ssl

Used in: socket-provider.

Description

The <ssl> element contains the configuration information for a socket provider that produces socket and channel implementations which use SSL. If SSL is configured for the unicast listener, the listener must be configured to use well known addresses.

Elements

Table A–60 describes the subelements of the ssl element.

Table A–60 ssl Subelements

Element	Required/ Optional	Description
<protocol>	Optional	Specifies the name of the protocol used by the socket and channel implementations produced by the SSL socket provider. The default value is TLS.
<provider>	Optional	Specifies the configuration for a security provider instance.
<executor>	Optional	Specifies the configuration information for an implementation of the java.util.concurrent.Executor interface. A <class-name> subelement is used to provide the name of a class that implements the Executor interface. As an alternative, use a <class-factory-name> subelement to specify a factory class for creating Executor instances and a <method-name> subelement that specifies the name of a static factory method on the factory class which performs object instantiation. Either approach can specify initialization parameters using the <init-params> element.
<identity-manager>	Optional	Specifies the configuration information for initializing an identity manager instance.
<trust-manager>	Optional	Specifies the configuration information for initializing a trust manager instance.
<hostname-verifier>	Optional	Specifies the configuration information for an implementation of the javax.net.ssl.HostnameVerifier interface. During the SSL handshake, if the URL's host name and the server's identification host name mismatch, the verification mechanism calls back to this instance to determine if the connection should be allowed. A <class-name> subelement is used to provide the name of a class that implements the HostnameVerifier interface. As an alternative, use a <class-factory-name> subelement to specify a factory class for creating HostnameVerifier instances and a <method-name> subelement that specifies the name of a static factory method on the factory class which performs object instantiation. Either approach can specify initialization parameters using the <init-params> element.

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 use a single "ring" of TCP connections spanning the entire cluster. A cluster node can use the TCP connection to detect the death of another node within a heartbeat interval (the default value s 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 A–61 describes the subelements of the `tcp-ring-listener` element.

Table A–61 tcp-ring-listener Subelements

Element	Required/ Optional	Description
<enabled>	Optional	Specifies whether the tcp ring listener should be enabled to defect node failures faster. Legal values are `true` and `false`. The default value is `true`.
`<ip-timeout>`	Optional	Specifies the timeout to use for determining that a computer that is hosting cluster members has become unreachable. A number of connection attempts may be made before determining that the unreachable members should be removed. Legal values are strings representing time intervals. A timeout of `0` disables system-level monitoring and is not recommended. The default value is `5s`. The values of the `<ip-timeout>` and `<ip-attempts>` elements should be high enough to insulate against allowable temporary network outages. This feature relies upon the `java.net.InetAddress.isReachable` mechanism, refer to the API documentation see for a description of how it identifies reachability.
`<ip-attempts>`	Optional	specifies the number of connection attempts to make before determining that a computer that is hosting cluster members has become unreachable, and that those cluster members should be removed. The values of the `<ip-timeout>` and `<ip-attempts>` elements should be high enough to insulate against allowable temporary network outages. Legal values are positive integers. The default value is `3`.
`<listen-backlog>`	Optional	Specifies the size of the TCP/IP server socket backlog queue. Valid values are positive integers. The default value is O/S dependent.
`<priority>`	Required	Specifies a priority of the tcp ring listener execution thread. Legal values are from 1 to 10. The default value is 6.

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 is 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 element.

Elements

Table A–62 describes the subelements of the traffic-jam element.

Table A–62 traffic-jam Subelements

Element	Required/ Optional	Description
<maximum-packets>	Required	Specifies the maximum number of pending packets that the Publisher tolerates 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. The default value is 8192.
<pause-milliseconds>	Required	Number of milliseconds that the Publisher pauses a client thread that is trying to send a message when the Publisher is clogged. The Publisher does not allow the message to go through until the clog is gone, and repeatedly sleeps the thread for the duration specified by this property. The default value is 10.

trust-manager

Used in: ssl.

Description

The <trust-manager> element contains the configuration information for initializing a javax.net.ssl.TrustManager instance.

A trust manager is responsible for managing the trust material that is used when making trust decisions and for deciding whether credentials presented by a peer should be accepted.

A valid trust-manager configuration contains at least one child element.

Elements

Table A–63 describes the elements of the trust-manager element.

Table A–63 trust-manager Subelements

Element	Required/ Optional	Description
<algorithm>	Optional	Specifies the algorithm used by the trust manager. The default value is SunX509.
<provider>	Optional	Specifies the configuration for a security provider instance.
<key-store>	Optional	Specifies the configuration for a key store implementation.

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 binds to, to listen for point-to-point cluster communications.

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.

Elements

Table A–64 describes the subelements of the unicast-listener element.

Table A–64 unicast-listener Subelements

Element	Required/ Optional	Description
<socket-provider>	Optional	Specifies either: the class configuration information for a com.tangosol.net.SocketProvider implementation, or it references a socket provider configuration that is defined within the <socket-providers> element. Three pre-defined socket providers are available: system (default), ssl, and tcp and are referred to using their defined id attribute name. For example: <socket-provider>ssl</socket-provider> The preconfigured system property override is tangosol.coherence.cluster.socketprovider.
<reliable-transport>	Optional	Specifies the name of the transport used by TCMP for reliable point-to-point communications. Valid values are: datagram, tmb (TCP Message Bus), and sdmb (SDP Message Bus). Default value is datagram. This element is only valid for Exalogic systems. **Note:** Setting a value other than datagram results in the majority of cluster data being transported outside of the protection of SSL. Specifically, only cluster join operations are protected when enabling SSL for TCMP. The preconfigured system property override is tangosol.coherence.transport.reliable.
<well-known-addresses>	Optional	Contains a list of "well known" addresses (WKA) that are used by the cluster discovery protocol instead of using multicast broadcast to discover cluster members.
<address>	Required	Specifies the IP address that a Socket listens or publishes on. **Note:** The localhost setting may not work on systems that define localhost as the loopback address; in that case, specify the computer name or the specific IP address. Also, the multicast listener, by default, binds to the same interface as defined by this address. The default value is localhost. The preconfigured system property override is tangosol.coherence.localhost.

Table A–64 (Cont.) unicast-listener Subelements

Element	Required/ Optional	Description
`<port>`	Required	Specifies the ports that the Socket listens or publishes on. A second port is automatically opened and defaults to the next available port. Legal values are from 1 to 65535. The default value is `8088` for the first port and `8089` (if available) for the second port. The preconfigured system property override is `tangosol.coherence.localport`.
`<port-auto-adjust>`	Required	Specifies whether the unicast port is automatically incremented if the specified port cannot be bound to because it is in use. Legal values are `true` or `false`. The default value is `true`. The preconfigured system property override is `tangosol.coherence.localport.adjust`.
`<packet-buffer>`	Required	Specifies how many incoming packets the operating system is requested to buffer. The value may be expressed either in terms of packets of bytes.
`<priority>`	Required	Specifies a priority of the unicast listener execution thread. Legal values are from 1 to 10. The default value is `8`.

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

Elements

Table A–65 describes the subelements of the `packet-speaker` element.

Table A–65 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 forces the speaker to always be used, while a very high value causes it to never be used. If unspecified (the default), it matches the `packet-buffer`.

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.
>
> If you are having difficulties establishing a cluster when using multicast, see *Oracle Coherence Administrator's Guide* for instructions on 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 node that is configured as a well-known node must be running for other nodes to be able to join.

Example

Example A–1 illustrates a configuration for two well-known-addresses with the default port.

Example A–1 Configuration for Two Well-Known-Addresses

```
<cluster-config>
   <unicast-listener>
      <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>
   </unicast-listener>
</cluster-config>
```

Elements

Table A–66 describes the subelements of the `well-known-addresses` element.

Table A–66 well-known-addresses Subelements

Element	Required/ Optional	Description
<socket-address>	Optional	Specifies a list of WKA that are used by the cluster discovery protocol instead of using multicast broadcast. If one or more WKA is specified, for a member to join the cluster it either has to be a WKA or there has to be at least one WKA member running. Additionally, all cluster communication is performed using unicast. If empty or unspecified, multicast communications is used. The preconfigured system property overrides are `tangosol.coherence.wka` and `tangosol.coherence.wka.port`.
<address-provider>	Optional	Contains the configuration for a `com.tangosol.net.AddressProvider` implementation that supplies the WKAs. The calling component attempts to obtain the full list upon node startup, the provider must return a terminating `null` address to indicate that all available addresses have been returned.

Attribute Reference

Table A–67 describes the attributes available in the operational deployment descriptor.

Table A–67 Operational Deployment Descriptor Attributes

Attribute	Required/ Optional	Description
`xml-override`	Optional	The `xml-override` attribute allows the content of an element to be fully or partially overridden with an XML document that is external to the base document. Legal value of this attribute is the name of the XML document an should be accessible using the `ClassLoader.getResourceAsStream(String name)` by the classes contained in `coherence.jar` library. In general, the 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, <`service`>) the `id` attribute is used to identify the base element that is 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. The following elements can be overridden by its own XML override file: `authorized-hosts`, `cache-factory-builder-config`, `cluster-config`, `coherence`, `configurable-cache-factory-config`, `incoming-message-handler`, `logging-config`, `multicast-listener`, `outgoing-message-handler`, `security-config`, `serializer`, `service`, `service-failure-policy`, `shutdown-listener`, `tcp-ring-listener`, `unicast-listener`, `packet-speaker`, `packet-publisher`, `mbeans`
`id`	Optional	The `id` attribute differentiates elements that can have multiple occurrences (for example, <`service`>). See "Understanding the XML Override Feature" on page 3-15.
`system-property`	Optional	This attribute is used to specify a system property name for any element. The system property is used to override the element value from the Java command line. This feature enables the same operational descriptor (and override file) to be used across all cluster nodes and customize each node using the system properties. See Appendix C, "Command Line Overrides," for more information on this feature.

B

Cache Configuration Elements

This appendix provides a detailed reference of the cache configuration deployment descriptor elements and includes a brief overview of the descriptor.

The following sections are included in this appendix:

Cache Configuration Deployment Descriptor

The cache configuration deployment descriptor specifies the various types of caches that can be used within a cluster. The name and location of the descriptor is specified in the operational deployment descriptor and defaults to `coherence-cache-config.xml`. A sample configuration descriptor is packaged in the root of the `coherence.jar` library and is used unless a custom `coherence-cache-config.xml` file is found before the `coherence.jar` file within the application's classpath. All cluster members should use identical cache configuration descriptors if possible.

The cache configuration deployment descriptor schema is defined in the `coherence-cache-config.xsd` file, which imports the `coherence-cache-config-base.xsd` file, which, in turn, imports the `coherence-config-base.xsd` file. These XSD files are located in the root of the `coherence.jar` library and at the following Web URL:

`http://xmlns.oracle.com/coherence/coherence-cache-config/1.0/coherence-cache-config.xsd`

The `<cache-config>` element is the root element of the cache configuration descriptor and typically includes an XSD and Coherence namespace reference and the location of the `coherence-cache-config.xsd` file. For example:

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

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">
```

Notes:

- The schema located in the `coherence.jar` library is always used at run time even if the `xsi:schemaLocation` attribute references the Web URL.
- The `xsi:schemaLocation` attribute can be omitted to disable schema validation.
- When deploying Coherence into environments where the default character set is EBCDIC rather than ASCII, ensure that the deployment descriptor file is in ASCII format and is deployed into its run-time environment in the binary format.

Element Reference

Table B–1 lists all non-terminal cache configuration deployment descriptor elements.

Table B–1 Non-Terminal Cache Configuration Elements

Element	Used In
acceptor-config	proxy-scheme
address-provider	tcp-acceptor, remote-addresses
async-store-manager	external-scheme, paged-external-scheme
authorized-hosts	tcp-acceptor
back-scheme	near-scheme, overflow-scheme
backing-map-scheme	distributed-scheme, optimistic-scheme, replicated-scheme
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
defaults	cache-config
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
http-acceptor	acceptor-config
identity-manager	ssl
flashjournal-scheme	backing-map-scheme
front-scheme	near-scheme, overflow-scheme
init-param	init-params
init-params	class-scheme
initiator-config	remote-cache-scheme, remote-invocation-scheme
instance	serializer, socket-provider, service-failure-policy

Table B–1 (Cont.) Non-Terminal Cache Configuration Elements

Element	Used In
invocation-scheme	caching-schemes
key-associator	distributed-scheme
key-partitioning	distributed-scheme
key-store	identity-manager, trust-manager
lh-file-manager	external-scheme, paged-external-scheme, async-store-manager
listener	local-scheme, external-scheme, paged-external-scheme, distributed-scheme, replicated-scheme, optimistic-scheme, near-scheme, overflow-scheme, read-write-backing-map-scheme
local-address	http-acceptor, tcp-acceptor, tcp-initiator
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
partitioned-quorum-policy-scheme	distributed-scheme
provider	identity-manager, ssl, trust-manager
proxy-config	proxy-scheme
proxy-scheme	caching-schemes
proxy-quorum-policy-scheme	proxy-scheme
ramjournal-scheme	backing-map-scheme
read-write-backing-map-scheme	caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme
remote-addresses	tcp-initiator
remote-cache-scheme	cachestore-scheme, caching-schemes, near-scheme
remote-invocation-scheme	caching-schemes

Table B–1 (Cont.) Non-Terminal Cache Configuration Elements

Element	Used In
replicated-scheme	caching-schemes, near-scheme, overflow-scheme
serializer	acceptor-config, defaults, distributed-scheme, initiator-config, invocation-scheme, optimistic-scheme, replicated-scheme, transactional-scheme
socket-address	remote-addresses
socket-provider	tcp-acceptor, tcp-initiator
ssl	socket-provider
tcp-acceptor	acceptor-config
tcp-initiator	initiator-config
transactional-scheme	caching-schemes
trust-manager	ssl

acceptor-config

Used in: proxy-scheme

Description

The acceptor-config element specifies the configuration information for a TCP/IP or HTTP (for REST) connection acceptor. The connection acceptor is used by a proxy service to enable Coherence*Extend clients to connect to the cluster and use cluster services without having to join the cluster.

Elements

Table B–2 describes the subelements of the acceptor-config element.

Table B–2 acceptor-config Subelements

Element	Required/ Optional	Description
<http-acceptor>	Optional	Specifies the configuration information for a connection acceptor that accepts connections from remote REST clients over HTTP. This element cannot be used together with the <tcp-acceptor> element.
<tcp-acceptor>	Optional	Specifies the configuration information for a connection acceptor that enables Coherence*Extend clients to connect to the cluster over TCP/IP. This element cannot be used together with the <http-acceptor> element.
<outgoing-message-handler>	Optional	Specifies the configuration information used by the connection acceptor to detect dropped client-to-cluster connections.

<serializer>	Optional	Specifies the class configuration information 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>`

<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. The default value is zero.

address-provider

Used in: tcp-acceptor, remote-addresses

Description

Contains the configuration information for an address factory that implements the com.tangosol.net.AddressProvider interface.

Elements

Table B–3 describes the subelements of the address-provider element.

Table B–3 address-provider Subelements

Element	Required/ Optional	Description
<class-name>	Optional	Specifies the fully qualified name of a class that implements the com.tangosol.net.AddressProvider interface. This element cannot be used with the <class-factory-name> element.
<class-factory-name>	Optional	Specifies the fully qualified name of a factory class for creating address provider instances. The instances must implement the com.tangosol.net.AddressProvider interface. This element cannot be used with the <class-name> element and is used with the <method-name> element.
<method-name>	Optional	Specifies the name of a static factory method on the factory class which performs 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. Initialization parameters can be specified for both the <class-name> element and the <class-factory-name> element.

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 B–4 describes the subelements of the async-store-manager element.

Table B–4 async-store-manager Subelements

Element	Required/ Optional	Description
<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.
<init-params>	Optional	Specifies initialization parameters, for use in custom async-store-manager implementations which implement the com.tangosol.run.xml.XmlConfigurable interface.
<bdb-store-manager>	Optional	Configures the external cache to use Berkeley Database JE on disk databases for cache storage.
<custom-store-manager>	Optional	Configures the external cache to use a custom storage manager implementation.
<lh-file-manager>	Optional	Configures the external cache to use a Coherence LH on disk database for cache storage.

Table B–4 (Cont.) async-store-manager Subelements

Element	Required/ Optional	Description
<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.
<async-limit>	Optional	Specifies the maximum number of bytes that are 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 necessaries 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. The default value is 4MB.

authorized-hosts

Used in: `tcp-acceptor`.

Description

This element contains the collection of IP addresses of TCP/IP initiator hosts that are allowed to connect to the cluster using a TCP/IP acceptor. If this collection is empty no constraints are imposed. Any number of `host-address` and `host-range` elements may be specified.

Elements

Table B–5 describes the subelements of the `authorized-hosts` element.

Table B–5 authorized-hosts Subelements

Element	Required/ Optional	Description
`<host-address>`	Optional	Specifies an IP address or host name. If any are specified, only hosts with specified host-addresses or within the specified host-ranges are 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 are allowed to join the cluster.
`<host-filter>`	Optional	Specifies class configuration information for a `com.tangosol.util.Filter` implementation that is used by a TCP/IP acceptor to determine whether to accept a particular TCP/IP initiator. The `evaluate()` method is passed to the `java.net.InetAddress` of the client. Implementations should return `true` to allow the client to connect. Classes are specified using the `<class-name>` subelement. Any initialization parameters can be defined within an `<init-params>` subelement.

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 "Attribute Reference" on page A-85.

back-scheme

Used in: near-scheme, overflow-scheme

Description

The back-scheme element specifies the back-tier cache of a composite cache.

Elements

Table B–6 describes the subelements of the back-scheme element.

Table B–6 back-scheme Subelements

Element	Required/ Optional	Description
<distributed-scheme>	Optional	Defines a cache scheme where storage of cache entries is partitioned across the cluster nodes.
<optimistic-scheme>	Optional	Defines a replicated cache scheme which uses optimistic rather then pessimistic locking.
<replicated-scheme>	Optional	Defines a cache scheme where each cache entry is stored on all cluster nodes.
<transactional-scheme>	Optional	Defines a cache scheme where storage of cache entries is partitioned across the cluster nodes with transactional guarantees.
<local-scheme>	Optional	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.
<external-scheme>	Optional	External schemes define caches which are not JVM heap based, allowing for greater storage capacity.
<paged-external-scheme>	Optional	As with external-scheme, paged-external-schemes define caches which are not JVM heap based, allowing for greater storage capacity.
<class-scheme>	Optional	Class schemes provide a mechanism for instantiating an arbitrary Java object for use by other schemes. The scheme which contains this element dictates what class or interface(s) must be extended.
<flashjournal-scheme>	Optional	Specifies a scheme that uses journaling to store data to flash memory.
<ramjournal-scheme>	Optional	Specifies a scheme that uses journaling to store data to RAM memory.
<remote-cache-scheme>	Optional	Defines a cache scheme that enables caches to be accessed from outside a Coherence cluster by using Coherence*Extend.

backing-map-scheme

Used in: distributed-scheme, optimistic-scheme, replicated-scheme

Description

Specifies what type of cache is used within the cache server to store the entries.

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 17-6 for an example configuration.

Note: The partitioned subelement is used if and only if the parent element is the distributed-scheme.

Elements

Table B–7 describes the subelements of the backing-map-scheme element.

Table B–7 backing-map-scheme Subelements

Element	Required/ Optional	Description
<partitioned>	Optional	Specifies whether the enclosed backing map is a PartitionAwareBackingMap. (This element is respected only within a distributed-scheme.) If set to true, the scheme that is specified as the backing map is used to configure backing maps for each individual partition of the PartitionAwareBackingMap; otherwise, it is used for the entire backing map itself. The concrete implementations of the PartitionAwareBackingMap interface are: ▪ com.tangosol.net.partition.ObservableSplittingBackingCache ▪ com.tangosol.net.partition.PartitionSplittingBackingCache ▪ com.tangosol.net.partition.ReadWriteSplittingBackingMap Valid values are true or false. The default value is false. See Chapter 13, "Implementing Storage and Backing Maps."
<local-scheme>	Optional	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.
<external-scheme>	Optional	External schemes define caches which are not JVM heap based, allowing for greater storage capacity.
<paged-external-scheme>	Optional	As with external-scheme, paged-external-schemes define caches which are not JVM heap based, allowing for greater storage capacity.
<overflow-scheme>	Optional	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.

Table B–7 (Cont.) backing-map-scheme Subelements

Element	Required/ Optional	Description
<class-scheme>	Optional	Class schemes provide a mechanism for instantiating an arbitrary Java object for use by other schemes. The scheme which contains this element dictates what class or interface(s) must be extended.
<flashjournal-scheme>	Optional	Specifies a scheme that uses journaling to store data to flash memory.
<ramjournal-scheme>	Optional	Specifies a scheme that uses journaling to store data to RAM memory.
<read-write-backing-map-scheme>	Optional	The read-write-backing-map-scheme defines a backing map which provides a size limited cache of a persistent store.

backup-storage

Used in: `distributed-scheme`.

Description

The `backup-storage` element specifies the type and configuration of backup storage for a partitioned cache.

Elements

Table B–8 describes the subelements of the `backup-storage` element.

Table B–8 backup-storage Subelements

Element	Required/ Optional	Description
`<type>`	Optional	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. The 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 A-59.
`<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). The default value is the `backup-storage/initial-size` value specified in the `tangosol-coherence.xml` descriptor. See "DistributedCache Service Parameters" on page A-59 for more information.
`<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). The default value is the `backup-storage/maximum-size` value specified in the `tangosol-coherence.xml` descriptor. See "DistributedCache Service Parameters" on page A-59 for more information.

Table B–8 (Cont.) backup-storage Subelements

Element	Required/ Optional	Description
`<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`) uses as "root" to store files in. If not specified or specifies a non-existent directory, a temporary file in the default location is used. The default value is the `backup-storage/directory` value specified in the `tangosol-coherence.xml` descriptor. See "DistributedCache Service Parameters" on page A-59 for more information.
`<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. The default value is the `backup-storage/class-name` value specified in the `tangosol-coherence.xml` descriptor. See "DistributedCache Service Parameters" on page A-59 for more information.
`<scheme-name>`	Optional	Only applicable with the scheme type. Specifies a scheme name for the `ConfigurableCacheFactory`. The default value is the `backup-storage/scheme-name` value specified in the `tangosol-coherence.xml` descriptor. See "DistributedCache Service Parameters" on page A-59 for more information.

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.

http://www.oracle.com/technology/documentation/berkeley-db/je/index.html

Description

The BDB store manager is used to define external caches which uses Berkeley Database JE on disk embedded databases for storage. See the examples of Berkeley-based store configurations in "Persistent Cache on Disk" on page 17-3 and "In-memory Cache with Disk Based Overflow" on page 17-4.

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 B–9 describes the subelements of the bdb-store-manager element.

Table B–9 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.

Table B–9 (Cont.) bdb-store-manager Subelements

Element	Required/ Optional	Description
<init-params>	Optional	Specifies additional Berkeley DB configuration settings. See the Berkeley DB Configuration instructions: http://www.oracle.com/technology/documentation/berkeley-db/je/GettingStartedGuide/administration.html#propertyfile Also used to specify initialization parameters, for use in custom implementations which implement the `com.tangosol.run.xml.XmlConfigurable` interface.
<directory>	Optional	Specifies the path name to the root directory where the Berkeley Database JE store manager stores files. If not specified or specified with a non-existent directory, a temporary directory in the default location is used.
<store-name>	Optional	Specifies the name for a database table that the Berkeley Database JE store manager uses to store data in. Specifying this parameter causes 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. This setting should **not** be enabled with replicated or distributed caches. Normally, the <store-name> element should be left unspecified, indicating that temporary storage is to be used. When specifying this property, it is recommended to use the `{cache-name}` macro. See "Using Parameter Macros" on page 12-12 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 B–10 describes the subelements of the bundle-config element.

Table B–10 bundle-config Subelements

Element	Required/ Optional	Description
<operation-name>	Optional	Specifies the operation name for which calls performed concurrently on multiple threads are "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. The default value is all.
<preferred-size>	Optional	Specifies the bundle size threshold. When a bundle size reaches this value, the corresponding "bulk" operation is invoked immediately. This value is measured in context-specific units. Valid values are zero (disabled bundling) or positive values. The default value is zero.
<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 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. The default value is 1.
<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. The default value is 4.
<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. The default value is false.

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 B-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 B–11 describes the subelements of the `cache-config` element.

Table B–11 cache-config Subelements

Element	Required/ Optional	Description
`<scope-name>`	Optional	Specifies the scope name for this configuration. The scope name is typically used (as a prefix) for all services generated by a cache factory to isolate services indicated in this cache configuration from services created by cache factories with other configurations, thus avoiding unintended joining of services with similar names from different configurations.
`<defaults>`	Optional	Defines factory wide default settings.
`<caching-scheme-mapping>`	Required	Specifies the caching-scheme that is 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 B–12 describes the subelements of the `cache-mapping` element.

Table B–12 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 is 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 is replaced with the string `com.acme.MyCacheLoader` and any occurrence of the literal `size-limit` is replaced with the value of `1000`.

cache-service-proxy

Used in: proxy-config

Description

The `cache-service-proxy` element contains the configuration information for a cache service proxy that is managed by a proxy service.

Elements

Table B–13 describes the subelements of the `cache-service-proxy` element.

Table B–13 cache-service-proxy Subelements

Element	Required/ Optional	Description
`<class-name>`	Optional	Specifies the fully qualified name of a class that implements the `com.tangosol.net.CacheService` interface. The class acts as an interceptor between a client and a proxied cache service to implement custom processing as required. For example, the class could be used to perform authorization checks before allowing the use of the proxied cache service.
`<init-params>`	Optional	Contains initialization parameters for the `CacheService` implementation.
`<enabled>`	Optional	Specifies whether the cache service proxy is enabled. If disabled, clients are not able to access any proxied caches. Legal values are `true` or `false`. The 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`. The 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`. The default value is `false`.

cachestore-scheme

Used in: local-scheme, read-write-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 17-4 for an example of using a `cachestore-scheme`.

Elements

Table B–14 describes the subelements of the `cachestore-scheme` element.

Table B–14 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 "Using Scheme Inheritance" on page 12-9.
<class-scheme>	Optional	Specifies the implementation of the cache store. The specified class must implement either 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 information 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-` uses a distributed (distributed-scheme) caching scheme, while caches starting with the name `rates-` uses a `replicated-scheme` caching scheme.

Elements

Table B–15 describes the subelement you can define within the `caching-scheme-mapping` element.

Table B–15 caching-scheme-mapping Subelement

Element	Required/ Optional	Description
<cache-mapping>	Required	Contains a single binding between a cache name and the caching scheme this cache uses.

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 mappings (see caching-scheme-mapping).

Elements

Table B–16 describes the different types of schemes you can define within the `caching-schemes` element.

Table B–16 caching-schemes Subelements

Element	Required/ Optional	Description
`<distributed-scheme>`	Optional	Defines a cache scheme where storage of cache entries is partitioned across the cluster nodes.
`<optimistic-scheme>`	Optional	Defines a replicated cache scheme which uses optimistic rather then pessimistic locking.
`<replicated-scheme>`	Optional	Defines a cache scheme where each cache entry is stored on all cluster nodes.
`<transactional-scheme>`	Optional	Defines a cache scheme where storage of cache entries is partitioned across the cluster nodes with transactional guarantees.
`<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.
`<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.
`<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.
`<flashjournal-scheme>`	Optional	Specifies a scheme that stores data to flash memory.
`<ramjournal-scheme>`	Optional	Specifies a scheme that stores data to RAM memory.
`<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.

Table B–16 (Cont.) caching-schemes Subelements

Element	Required/ Optional	Description
<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.
<remote-cache-scheme>	Optional	Defines a cache scheme that enables caches to be accessed from outside a Coherence cluster by using Coherence*Extend.
<remote-invocation-scheme>	Optional	Defines an invocation scheme that enables invocations from outside a Coherence cluster by using Coherence*Extend.
<proxy-scheme>	Optional	Defines a proxy service scheme that enables remote connections to a cluster using Coherence*Extend.

class-scheme

Used in: caching-schemes, local-scheme, distributed-scheme, replicated-scheme, optimistic-scheme, near-scheme, overflow-scheme, read-write-backing-map-scheme, cachestore-scheme, listener, eviction-policy, unit-calculator.

Description

Class schemes provide a mechanism for instantiating an arbitrary Java object for use by other schemes. The scheme which contains this element dictates what class or interface(s) must be extended. See "Cache of a Database" on page 17-4 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 B–17 describes the subelements of the class-scheme element.

Table B–17 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 "Using Scheme Inheritance" on page 12-9 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. This element cannot be used with the <class-factory-name> element.
<class-factory-name>	Optional	Specifies a fully specified name of a Java class that is used as a factory for object instantiation. This element cannot be used with the <class-name> element and is used with the <method-name> element.
<method-name>	Optional	Specifies the name of a static factory method on the factory class which performs 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 B–18 describes the subelements of the custom-store-manager element.

Table B–18 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.

defaults

Used in: cache-config

Description

The defaults element defines factory wide default settings. This feature enables global configuration of serializers and socket providers used by all services which have not explicitly defined these settings.

Elements

Table B–19 describes the subelements of the defaults element.

Table B–19 defaults Subelements

Element	Required/ Optional	Description
<serializer>	Optional	Specifies either: the class configuration information for a com.tangosol.io.Serializer implementation, or it references a serializer class configuration that is defined within the <serializers> element in the operational configuration file. Two pre-defined serializers are available: java (default) and pof and are referred to using their defined id attribute name. For example: `<serializer>pof</serializer>`
<socket-provider>	Optional	Specifies either: the class configuration information for a com.tangosol.net.SocketProvider implementation, or it references a socket provider configuration that is defined within the <socket-providers> element of the operational deployment descriptor. Two pre-defined socket providers are available: system (default) and ssl and are referred to using their defined id attribute name. For example: `<socket-provider>ssl</socket-provider>` This setting only specifies the socket provider for Coherence*Extend services. The TCMP socket provider is specified within the <unicast-listener> element in the operational configuration.

distributed-scheme

Used in: caching-schemes, near-scheme, overflow-scheme

Description

The distributed-scheme defines caches where the storage for entries is partitioned across cluster nodes. See "Distributed Cache" on page 11-1 for a more detailed description of partitioned caches. See "Partitioned Cache" on page 17-6 for 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) is 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 ensures 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> subelement. For instance a partitioned cache which uses a local-scheme for its backing map results in cache entries being stored in-memory on the storage-enabled cluster nodes.

Failover

For the purposes of failover, a configured 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 does not reside on the same computer as the primary partition. If a cluster node abruptly leaves the cluster,

responsibility for its partitions are automatically reassigned to the existing backups, and new backups of those partitions are 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 B–20 describes the subelements of the `distributed-scheme` element.

Table B–20 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 "Using Scheme Inheritance" on page 12-9 for more information.
`<service-name>`	Optional	Specifies the name of the service which manages caches created from this scheme. Services are configured in the `<services>` element in the `tangosol-coherence.xml` descriptor. See Appendix A, "Operational Configuration Elements" for more information.
`<serializer>`	Optional	Specifies either: the class configuration information for a `com.tangosol.io.Serializer` implementation used to serialize and deserialize user types, or it references a serializer class configuration that is defined in the operational configuration file (see "serializer" on page A-56).
`<compressor>`	Optional	Specifies whether or not backup updates should be compressed in delta form or sent whole. A delta update represents the parts of a backup entry that must be changed in order to synchronize it with the primary version of the entry. Deltas are created and applied using a compressor. The default value is the `compressor` value specified in the `tangosol-coherence.xml` descriptor. See the `compressor` parameter in "DistributedCache Service Parameters" on page A-59 for more information. Valid values are: ▪ `none` – Disables delta backup; no compressor is used. The whole backup binary entry is replaced when the primary entry changes. ▪ `standard` – Automatically selects a delta compressor based on the serializer being used by the partitioned service. ▪ `<instance>` – The configuration for a class that implements the `com.tangosol.io.DeltaCompressor` interface.
`<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. The default value is the `thread-count` value specified in the `tangosol-coherence.xml` descriptor. See the `thread-count` parameter in "DistributedCache Service Parameters" on page A-59 for more information.

Table B–20 (Cont.) distributed-scheme Subelements

Element	Required/ Optional	Description
`<lease-granularity>`	Optional	Specifies the lease ownership granularity. 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. The 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 A-59 for more information.
`<local-storage>`	Optional	Specifies whether a cluster node contributes storage to the cluster, that is, maintain partitions. When disabled the node is considered a cache client. Legal values are `true` or `false`. The 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 A-59 for more information.

`<partition-count>`	Optional	Specifies the number of partitions that a partitioned (distributed) cache is organized into. Each member running the partitioned cache service that has the local-storage (`<local-storage>` subelement) option set to `true` manages a balanced number of partitions.
		The number of partitions should be a prime number and sufficiently large such that a given partition is expected to be no larger than 50MB.
		The following are good defaults based on service storage sizes:
		`service storage partition-count`
		`100M 257`
		`1G 509`
		`10G 2039`
		`50G 4093`
		`100G 8191`
		A list of first 1,000 primes can be found at
		`http://primes.utm.edu/lists/`
		Valid values are positive integers. The default value is `257` as specified in the `tangosol-coherence.xml` descriptor. See the `partition-count` parameter in "DistributedCache Service Parameters" on page A-59.

Table B–20 (Cont.) distributed-scheme Subelements

Element	Required/ Optional	Description
<transfer-threshold>	Optional	Specifies the threshold for the primary buckets distribution in kilobytes. 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 makes the distribution process take longer, but reduces network bandwidth utilization during this activity. Legal values are integers greater then zero. The 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 A-59 for more information.
<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. A value of 0 means that for abnormal termination, some portion of the data in the cache is lost. Value of N means that if up to N cluster nodes terminate immediately, the cache data is 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 is in the order of M*(N+1). Recommended values are 0 or 1. The 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 A-59 for more information.
<backup-count-after-writebehind>	Optional	Specifies the number of members of the partitioned cache service that holds 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 is backed up on the number of members specified by the <backup-count> subelement. If the unit of storage is not marked as requiring write-behind, then it is 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. The value of 0 means that when write-behind has occurred, the backup copies of that data is discarded. However, until write-behind occurs, the data is backed up in accordance with the <backup-count> setting. Recommended value is 0 or this element should be omitted.
<backup-storage>	Optional	Specifies the type and configuration for the partitioned cache backup storage.
<key-associator>	Optional	Specifies a class that is 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.

Table B–20 (Cont.) distributed-scheme Subelements

Element	Required/ Optional	Description
<key-partitioning>	Optional	Specifies a class that implements the com.tangosol.net.partition.KeyPartitioningStrategy interface, which is responsible for assigning keys to partitions. This implementation must have a zero-parameter public constructor. If unspecified, the default key partitioning algorithm is used, which ensures that keys are evenly segmented across partitions.
<partition-assignment-strategy>	Optional	Specifies the strategy that is used by a partitioned service to manage partition distribution. Valid values are legacy or a class that implements the com.tangosol.net.partition.PartitionAssignmentStrategy interface. The legacy assignment strategy indicates that partition distribution is managed individually on each cluster member. Whereas; a custom strategy allows for a shared strategy across the cluster. Enter a custom strategy using the <instance> element. The default value is legacy.
<partition-listener>	Optional	Specifies a class that implements the com.tangosol.net.partition.PartitionListener interface.
<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). The 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 A-59 for more information.
<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). If zero is specified, the default service-guardian <timeout-milliseconds> value is used. Legal values are nonnegative integers. The default value is the value specified in the tangosol-coherence.xml descriptor. See the task-timeout parameter in "DistributedCache Service Parameters" on page A-59.
<request-timeout>	Optional	Specifies the maximum amount of time a client waits 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). The default value is the value specified in the tangosol-coherence.xml descriptor. See the request-timeout parameter in "DistributedCache Service Parameters" on page A-59 for more information.

Table B–20 (Cont.) distributed-scheme Subelements

Element	Required/ Optional	Description
<guardian-timeout>	Optional	Specifies the guardian timeout value to use for guarding the service and any dependent threads. If the element is not specified for a given service, the default guardian timeout (as specified by the <timeout-milliseconds> operational configuration element) is used. See <service-guardian>. 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.
<service-failure-policy>	Optional	Specifies the action to take when an abnormally behaving service thread cannot be terminated gracefully by the service guardian. Legal values are: ■ exit-cluster (default) – attempts to recover threads that appear to be unresponsive. If the attempt fails, an attempt is made to stop the associated service. If the associated service cannot be stopped, this policy causes the local node to stop the cluster services. ■ exit-process – attempts to recover threads that appear to be unresponsive. If the attempt fails, an attempt is made to stop the associated service. If the associated service cannot be stopped, this policy cause the local node to exit the JVM and terminate abruptly. ■ logging – causes any detected problems to be logged, but no corrective action to be taken. ■ a custom class – an <instance> subelement is used to provide the class configuration information for a com.tangosol.net.ServiceFailurePolicy implementation.
<member-listener>	Optional	Specifies the configuration information for a class that implements the com.tangosol.net.MemberListener interface. The implementation must have a public default constructor. See the subelements for "instance" on page B-49 for the elements used to define the class. The MemberListener implementation receives cache service lifecycle events. The <member-listener> element is used as an alternative to programmatically adding a MapListener on a service.
<operation-bundling>	Optional	Specifies the configuration information for a bundling strategy.

Table B–20 (Cont.) distributed-scheme Subelements

Element	Required/ Optional	Description
<backing-map-scheme>	Optional	Specifies what type of cache is used within the cache server to store the entries. Legal schemes are: ■ local-scheme ■ external-scheme ■ paged-external-scheme ■ class-scheme ■ flashjournal-scheme ■ ramjournal-scheme ■ overflow-scheme ■ read-write-backing-map-scheme Note that when using an off-heap backing map it is important that the corresponding <backup-storage> be configured for off-heap (potentially using the same scheme as the backing-map). Here off-heap refers to any storage where some or all entries are stored outside of the JVMs garbage collected heap space. Examples include: <overflow-scheme> and <external-scheme>. See "Partitioned Cache with Overflow" on page 17-6 for an example configuration.
<partitioned-quorum-policy-scheme>	Optional	Specifies quorum policy settings for the partitioned cache service.
<listener>	Optional	Specifies an implementation of a `MapListener` which is notified of events occurring on the cache.
<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`. The default value is `false`.

external-scheme

Used in: caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme, near-scheme, overflow-scheme, read-write-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 17-1 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 B–21 describes the subelements of the `external-scheme` element.

Table B–21 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 "Using Scheme Inheritance" on page 12-9 for more information
`<class-name>`	Optional	Specifies a custom implementation of the external cache. Any custom implementation must extend either 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.
`<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 B-37
`<bdb-store-manager>`	Optional	Configures the external cache to use Berkeley Database JE on disk databases for cache storage.
`<custom-store-manager>`	Optional	Configures the external cache to use a custom storage manager implementation.
`<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.

Table B–21 (Cont.) external-scheme Subelements

Element	Required/ Optional	Description
<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 begins 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. The default value is zero.
<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 that are required to cache the object. This calculator is used for Partitioned Caches that cache data in a binary serialized form. 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/ConfigurableCacheMap.UnitCalculator interface. This element is used only if the high-units element is set to a positive number. The default value is FIXED.

Table B–21 (Cont.) external-scheme Subelements

Element	Required/ Optional	Description
`<unit-factor>`	Optional	The unit-factor element specifies the factor by which the units, low-units and high-units properties are adjusted. Using a `BINARY` unit calculator, for example, the factor of 1048576 could be used to count megabytes instead of bytes. Using a BINARY unit calculator, for example, the factor of 1048576 could be used to count megabytes instead of bytes. **Note:** This element was introduced only to avoid changing the type of the units, low units and high units properties from 32-bit values to 64-bit values and is used only if the high-units element is set to a positive number. Valid values are positive integer numbers. The default value is `1`.
`<expiry-delay>`	Optional	Specifies the amount of time since the last update that entries are kept by the cache before being expired. Entries that have expired are not be accessible and are evicted the next time a client accesses the cache. 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. The default value is `0`. **Note:** The expiry delay parameter (`cExpiryMillis`) is defined as an integer and is expressed in milliseconds. Therefore, the maximum amount of time can never exceed `Integer.MAX_VALUE` (2147483647) milliseconds or approximately 24 days.
`<listener>`	Optional	Specifies an implementation of a `com.tangosol.util.MapListener` which is notified of events occurring on the cache.

flashjournal-scheme

Used in: back-scheme, backing-map-scheme, caching-schemes, front-scheme, internal-cache-scheme

Description

The flashjournal-scheme element contains the configuration information for a scheme that stores data to external block-based file stores (flash). A flash journal resource manager controls flash journal behavior. See "flashjournal-manager" on page A-15 for additional details on configuring flash journal behavior.

This scheme uses the com.tangosol.net.cache.SimpleSerializationMap class as the backing map implementation and the com.tangosol.io.journal.JournalBinaryStore to store and retrieve binary key value pairs to a journal.

Elements

Table B–53 describes the subelements of the flashjournal-scheme element.

Table B–22 flashjournal-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 "Using Scheme Inheritance" on page 12-9 for more information.
<class-name>	Optional	Specifies a custom implementation of the simple serialization map cache. Any custom implementation must extend the com.tangosol.net.cache.SimpleSerializationMap class and declare the exact same set of public constructors as the superclass.
<init-params>	Optional	Specifies the initialization parameters for a custom serialization map cache.
<listener>	Optional	Specifies an implementation of a com.tangosol.util.MapListener which is notified of events occurring on the cache.

front-scheme

Used in: near-scheme, overflow-scheme

Description

The front-scheme element specifies the front-tier cache of a composite cache.

Elements

Table B–23 describes the subelements of the front-scheme element.

Table B–23 front-scheme Subelements

Element	Required/ Optional	Description
<local-scheme>	Optional	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.
<external-scheme>	Optional	External schemes define caches which are not JVM heap based, allowing for greater storage capacity.
<paged-external-scheme>	Optional	As with external-scheme, paged-external-schemes define caches which are not JVM heap based, allowing for greater storage capacity.
<class-scheme>	Optional	Class schemes provide a mechanism for instantiating an arbitrary Java object for use by other schemes. The scheme which contains this element dictates what class or interface(s) must be extended.
<flashjournal-scheme>	Optional	Specifies a scheme that uses journaling to store data to flash memory.
<ramjournal-scheme>	Optional	Specifies a scheme that uses journaling to store data to RAM memory.

http-acceptor

Used in `acceptor-config`

Description

The http-acceptor element specifies a connection acceptor that accepts connections from remote REST clients over HTTP.

Elements

Table B–23 describes the subelements of the `http-acceptor` element.

Table B–24 http-acceptor subelements

Elements	Required/ Optional	Description
`<class-name>`	Optional	Specifies an HTTP server class that implements the `com.tangosol.coherence.rest.server.HttpServer` interface. The HTTP server class handles inbound HTTP requests. Coherence REST provides two implementations out of the box: `com.tangosol.coherence.rest.server.DefaultHttpServer` (backed by Oracle's lightweight HTTP server) and `com.tangosol.coherence.rest.server.GrizzlyHttpServer` (backed by Grizzly). The default value if no value is specified is `com.tangosol.coherence.rest.server.DefaultHttpServer`.
`<init-params>`	Optional	Contains class initialization parameters for the HTTP server class.
`<local-address>`	Required	Specifies the local address (IP or DNS name) and port on which the HTTP server socket is bound.
`<resource-config>`	Optional	Specifies a Jersey resource configuration class that extends the `com.sun.jersey.api.core.ResourceConfig` class. The resource configuration class is responsible for configuring Coherence RESTful Web Services. The instance is used by the HTTP acceptor to load resource and provider classes. The class must be specified within an `<instance>` subelement. The default value is the `com.tangosol.coherence.rest.server.DefaultResource` class.

identity-manager

Used in: ssl.

Description

The <identity-manager> element contains the configuration information for initializing a javax.net.ssl.KeyManager instance.

The identity manager is responsible for managing the key material which is used to authenticate the local connection to its peer. If no key material is available, the connection cannot present authentication credentials.

Elements

Table B–25 describes the elements you can define within the identity-manager element.

Table B–25 identity-manager Subelements

Element	Required/ Optional	Description
<algorithm>	Optional	Specifies the algorithm used by the identity manager. The default value is SunX509.
<provider>	Optional	Specifies the configuration for a security provider instance.
<key-store>	Optional	Specifies the configuration for a key store implementation.
<password>	Required	Specifies the private key password.

initiator-config

Used in: remote-cache-scheme, remote-invocation-scheme.

Description

The `initiator-config` element specifies the configuration information for a TCP/IP 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.

Elements

Table B–26 describes the subelements of the `initiator-config` element.

Table B–26 initiator-config Subelements

Element	Required/ Optional	Description
`<tcp-initiator>`	Optional	Specifies the configuration information for a connection initiator that connects to the cluster over TCP/IP.
`<outgoing-message-handler>`	Optional	Specifies the configuration information used by the connection initiator to detect dropped client-to-cluster connections.
`<serializer>`	Optional	Specifies either: the class configuration information for a `com.tangosol.io.Serializer` implementation used to serialize and deserialize user types, or it references a serializer class configuration that is defined in the operational configuration file (see "serializer" on page A-56).

init-param

Used in: init-params.

Defines an individual initialization parameter.

Elements

Table B–27 describes the subelements of the `init-param` element.

Table B–27 init-param Subelements

Element	Required/ Optional	Description
<param-name>	Optional	Specifies the name of the initialization parameter. For example: <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> The <param-name> element cannot be specified if the <param-type> element is specified.
<param-type>	Optional	Specifies the Java type of the initialization parameter.The following standard types are supported: ■ java.lang.String (string) ■ java.lang.Boolean (boolean) ■ java.lang.Integer (int) ■ java.lang.Long (long) ■ java.lang.Double (double) ■ java.math.BigDecimal ■ java.io.File ■ java.sql.Date ■ java.sql.Time ■ java.sql.Timestamp For example: <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> The <param-type> element cannot be specified if the <param-name> element is specified.
<param-value>	Required	Specifies the value of the initialization parameter. The value is in the format specific to the Java type of the parameter.
<description>	Optional	Specifies a description for the initialization parameter.

init-params

Used in: class-scheme, cache-mapping.

Description

Defines a series of initialization parameters as name-value pairs. See "Partitioned Cache of a Database" on page 17-7 for an example of using `init-params`.

Elements

Table B–28 describes the subelements of the `init-params` element.

Table B–28 init-params Subelements

Element	Required/ Optional	Description
<init-param>	Optional	Defines an individual initialization parameter.

instance

Used in: serializer, socket-provider, service-failure-policy, load-balancer, and partition-assignment-strategy

Description

The <instance> element contains the configuration of an implementation class or class factory that is used to plug in custom functionality.

Elements

Table B–29 describes the subelements of the instance element.

Table B–29 instance Subelements

Element	Required/ Optional	Description
<class-name>	Optional	Specifies the fully qualified name of an implementation class. This element cannot be used with the <class-factory-name> element.
<class-factory-name>	Optional	Specifies the fully qualified name of a factory class for creating implementation class instances. This element cannot be used with the <class-name> element and is used with the <method-name> element.
<method-name>	Optional	Specifies the name of a static factory method on the factory class which performs object instantiation.
<init-params>	Optional	Contains class initialization parameters for the implementation class.

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

Table B–30 describes the subelements of the `invocation-scheme` element.

Table B–30 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 "Using Scheme Inheritance" on page 12-9 for more information.
`<service-name>`	Optional	Specifies the name of the service which manages invocations from this scheme.
`<serializer>`	Optional	Specifies either: the class configuration information for a `com.tangosol.io.Serializer` implementation used to serialize and deserialize user types, or it references a serializer class configuration that is defined in the operational configuration file (see "serializer" on page A-56).
`<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. The default value is the `thread-count` value specified in the `tangosol-coherence.xml` descriptor. See the `thread-count` parameter in "InvocationService Parameters" on page A-66.
`<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). The 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 A-66.
<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 do not 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). If `zero` is specified, the default service-guardian `<timeout-milliseconds>` value is used. Legal values are nonnegative integers. The default value is the `task-timeout` value specified in the `tangosol-coherence.xml` descriptor. See the `task-timeout` parameter in "InvocationService Parameters" on page A-66.

Table B–30 (Cont.) invocation-scheme Subelements

Element	Required/ Optional	Description
<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 do not 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). The default value is the `request-timeout` value specified in the `tangosol-coherence.xml` descriptor. See the `request-timeout` parameter in "InvocationService Parameters" on page A-66.
`<guardian-timeout>`	Optional	Specifies the guardian timeout value to use for guarding the service and any dependent threads. If the element is not specified for a given service, the default guardian timeout (as specified by the `<timeout-milliseconds>` operational configuration element) is used. See <service-guardian>. 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.

Table B–30 (Cont.) invocation-scheme Subelements

Element	Required/ Optional	Description
`<service-failure-policy>`	Optional	Specifies the action to take when an abnormally behaving service thread cannot be terminated gracefully by the service guardian. Legal values are: ■ `exit-cluster` (default) – attempts to recover threads that appear to be unresponsive. If the attempt fails, an attempt is made to stop the associated service. If the associated service cannot be stopped, this policy causes the local node to stop the cluster services. ■ `exit-process` – attempts to recover threads that appear to be unresponsive. If the attempt fails, an attempt is made to stop the associated service. If the associated service cannot be stopped, this policy cause the local node to exit the JVM and terminate abruptly. ■ `logging` – causes any detected problems to be logged, but no corrective action to be taken. ■ a custom class – an `<instance>` subelement is used to provide the class configuration information for a `com.tangosol.net.ServiceFailurePolicy` implementation.
`<member-listener>`	Optional	Specifies the configuration information for a class that implements the `com.tangosol.net.MemberListener` interface. The implementation must have a public default constructor. The `MemberListener` implementation receives service lifecycle events. The `<member-listener>` element is used as an alternative to programmatically adding a `MapListener` on a 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`. The default value is `false`.

invocation-service-proxy

Used in: proxy-config

Description

The `invocation-service-proxy` element contains the configuration information for an invocation service proxy managed by a proxy service.

Elements

Table B–31 describes the subelements of the `invocation-service-proxy` element.

Table B–31 invocation-service-proxy Subelement

Element	Required/ Optional	Description
`<class-name>`	Optional	Specifies the fully qualified name of a class that implements the `com.tangosol.net.InvocationService` interface. The class acts as an interceptor between a client and a proxied invocation service to implement custom processing as required. For example, the class could be used to perform authorization checks before allowing the use of the proxied invocation service.
`<init-params>`	Optional	Contains initialization parameters for the `InvocationService` implementation.
`<enabled>`	Optional	Specifies whether the invocation service proxy is enabled. If disabled, clients are not able to execute `Invocable` objects on the proxy service JVM. Legal values are `true` or `false`. The default value is `true`.

key-associator

Used in: distributed-scheme

Description

Specifies an implementation of a `com.tangosol.net.partition.KeyAssociator` which is 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 B–32 describes the subelements of the `key-associator` element.

Table B–32 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. The 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 A-59 for more information.
`<class-factory-name>`	Optional	Specifies the fully qualified name of a factory class for creating implementation class instances. This element cannot be used with the `<class-name>` element and is used with the `<method-name>` element.
`<method-name>`	Optional	Specifies the name of a static factory method on the factory class which performs object instantiation.
`<init-params>`	Optional	Contains class initialization parameters for the implementation class.

key-partitioning

Used in: distributed-scheme

Description

Specifies an implementation of a `com.tangosol.net.partition.KeyPartitioningStrategy` which is used to determine the partition in which a key resides.

Elements

Table B–33 describes the subelements of the `key-partitioning` element.

Table B–33 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. The 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 A-59 for more information.
`<class-factory-name>`	Optional	Specifies the fully qualified name of a factory class for creating implementation class instances. This element cannot be used with the `<class-name>` element and is used with the `<method-name>` element.
`<method-name>`	Optional	Specifies the name of a static factory method on the factory class which performs object instantiation.
`<init-params>`	Optional	Contains class initialization parameters for the implementation class.

key-store

Used in: identity-manager, trust-manager.

Description

The `key-store` element specifies the configuration for a key store implementation to use when implementing SSL. The key store implementation is an instance of the `java.security.KeyStore` class.

Elements

Table B–34 describes the elements you can define within the `key-store` element.

Table B–34 key-store Subelements

Element	Required/ Optional	Description
`<url>`	Required	Specifies the Uniform Resource Locator (URL) to a key store.
`<password>`	Optional	Specifies the password for the key store.
`<type>`	Optional	Specifies the type of a `java.security.KeyStore` instance. The default value is `JKS`.

lh-file-manager

Used in: external-scheme, paged-external-scheme, async-store-manager.

Description

Configures a store manager which uses a Coherence LH on disk, embedded database for storage. See "Persistent Cache on Disk" on page 17-3 and "In-memory Cache with Disk Based Overflow" on page 17-4 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 B–35 describes the subelements of the `lh-file-manager` element.

Table B–35 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 uses to store files in. If not specified or specifies a non-existent directory, a temporary file in the default location is used.
`<file-name>`	Optional	Specifies the name for a non-temporary (persistent) file that the LH file manager uses to store data in. Specifying this parameter causes 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 "Using Parameter Macros" on page 12-12. 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, overflow-scheme, read-write-backing-map-scheme

Description

The Listener element specifies an implementation of a com.tangosol.util.MapListener which is notified of events occurring on a cache.

Elements

Table B–36 describes the subelements of the listener element.

Table B–36 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-address

Used in: tcp-acceptor, tcp-initiator

Description

The `local-address` element specifies the local address (IP or DNS name) and port to which a TCP/IP socket is bound.

The `local-address` element is used within a TCP/IP acceptor definition to specify the address and port on which the TCP/IP server socket (opened by the connection acceptor) is bound. The socket is used by the proxy service to accept connections from Coherence*Extend clients. The following example binds the server socket to `192.168.0.2:9099`.

```
<local-address>
  <address>192.168.0.2</address>
  <port>9099</port>
</local-address>
```

The `local-address` element is used within a TCP/IP initiator definition to specify the local address and port on which the TCP/IP client socket (opened by the connection initiator) is bound. The socket is used by remote services to connect to a proxy service on the cluster. The following example binds the client socket to `192.168.0.1` on port `9099`:

```
<local-address>
  <address>192.168.0.1</address>
  <port>9099</port>
</local-address>
```

Elements

Table B–60 describes the subelements of the `local-address` element.

Table B–37 local-address Subelements

Element	Required/ Optional	Description
`<address>`	Optional	Specifies the address (IP or DNS name) on which a TCP/IP socket listens and publishes.
`<port>`	Optional	Specifies the port on which a TCP/IP socket listens and publishes. Legal values are from 1 to 65535. When used for a TCP/IP server (that is, for a TCP acceptor), the port child element is required.

local-scheme

Used in: caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme, near-scheme, overflow-scheme, read-write-backing-map-scheme, 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 "Near Cache" on page 17-8 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 B-23). Cache misses are read-through to the back end store to retrieve the data. If a writable store is provided, cache writes are also propagate to the cache store. For optimizing read/write access against a cache store, see the "read-write-backing-map-scheme" on page B-88.

Size Limited Cache

The cache may be configured as size-limited, which means that when it reaches its maximum allowable size (see the `<high-units>` subelement) it prunes itself back to a specified smaller size (see the `<low-units>` subelement), choosing which entries to evict according to its eviction-policy (see the `<eviction-policy>` subelement). The entries and size limitations are measured in terms of units as calculated by the scheme's unit-calculator (see the `<unit-calculator>` subelement).

Entry Expiration

The local cache supports automatic expiration of entries based on the age of the value (see the `<expiry-delay>` subelement).

Elements

Table B–38 describes the subelements of the `local-scheme` element.

Table B–38 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 "Using Scheme Inheritance" on page 12-9 for more information.
`<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.

Table B–38 (Cont.) local-scheme Subelements

Element	Required/ Optional	Description
<service-name>	Optional	Specifies the name of the service which manages caches created from this scheme. Services are configured from within the <services> element in the tangosol-coherence.xml descriptor. See Appendix A, "Operational Configuration Elements" for more information.
<init-params>	Optional	Specifies initialization parameters, for use in custom local cache implementations which implement the com.tangosol.run.xml.XmlConfigurable interface.
<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 (default) – Hybrid eviction policy chooses which entries to evict based on 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.LocalCache.EvictionPolicy interface.
<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 begins the pruning process, evicting entries according to the eviction policy. Legal values are positive integers or zero. Zero implies no limit. The default value is 0.
<low-units>	Optional	Contains the lowest number of units that a cache is pruned down to when pruning takes place. A pruning does not necessarily result in a cache containing this number of units, however a pruning never results in a cache containing less than this number of units. An entry is the unit of measurement, unless it is overridden by an alternate unit-calculator (see <unit-calculator> subelement). When pruning occurs entries continue to be evicted according to the eviction policy until this size. Legal values are positive integers or zero. Zero implies the default. The default value is 75% of the high-units setting (that is, for a high-units setting of 1000 the default low-units is 750).
<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. This element is used only if the high-units element is set to a positive number. Legal values are: ■ FIXED (default) – 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 that are required to cache the object. This calculator is used for Partitioned Caches that cache data in a binary serialized form. 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/ConfigurableCacheMap.UnitCalculator interface.

Table B–38 (Cont.) local-scheme Subelements

Element	Required/ Optional	Description
<unit-factor>	Optional	The unit-factor element specifies the factor by which the units, low-units and high-units properties are adjusted. Using a BINARY unit calculator, for example, the factor of 1048576 could be used to count megabytes instead of bytes. Using a BINARY unit calculator, for example, the factor of 1048576 could be used to count megabytes instead of bytes. **Note:** This element was introduced only to avoid changing the type of the units, low units and high units properties from 32-bit values to 64-bit values and is used only if the high-units element is set to a positive number. Valid values are positive integer numbers. The default value is 1.
<expiry-delay>	Optional	Specifies the amount of time since the last update that entries are kept by the cache before being expired. Entries that have expired are not accessible and are evicted the next time a client accesses the cache. Any attempt to read an expired entry results in a reloading of the entry from the configured cache store (see <cachestore-scheme>. 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. The default value is 0. **Note:** The expiry delay parameter (cExpiryMillis) is defined as an integer and is expressed in milliseconds. Therefore, the maximum amount of time can never exceed Integer.MAX_VALUE (2147483647) milliseconds or approximately 24 days.
<cachestore-scheme>	Optional	Specifies the store which is being cached. If unspecified the cached data only resides in memory, and only reflects operations performed on the cache itself.
<pre-load>	Optional	Specifies whether a cache pre-loads data from its CacheLoader (or CacheStore) object. Valid values are true and false. The default value is false.
<listener>	Optional	Specifies an implementation of a com.tangosol.util.MapListener which is notified of events occurring on the cache.

near-scheme

Used in: caching-schemes.

Description

The `near-scheme` defines a two-tier cache consisting of a front-tier which caches a subset of a back-tier cache. The front-tier is generally a fast, size limited cache, while the back-tier is slower, but much higher capacity cache. A typical deployment might use a local cahce for the front-tier, and a distributed cache for the back-tier. The result is that a portion of a large partitioned cache is cached locally in-memory allowing for very fast read access. See "Near Cache" on page 11-7 for a more detailed description of near caches, and "Near Cache" on page 17-8 for an example of near cache configurations.

Implementation

The near scheme is implemented by the `com.tangosol.net.cache.NearCache` class.

Front-tier Invalidation

The `<invalidation-strategy>` subelement defines a strategy that is used to keep the front tier of the near cache synchronized 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 B–39 describes the subelements of the `near-scheme` element.

Table B–39 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 "Using Scheme Inheritance" on page 12-9 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.
`<front-scheme>`	Required	Specifies the cache to use as the front-tier cache.
`<back-scheme>`	Required	Specifies the cache to use as the front-tier cache.

Table B–39 (Cont.) near-scheme Subelements

Element	Required/ Optional	Description
<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 (default) – instructs the near cache to switch between present and all strategies automatically based on the cache statistics.
<listener>	Optional	Specifies an implementation of a com.tangosol.util.MapListener which is notified of events occurring on the cache.
<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. The 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 B–40 describes the subelements of the nio-file-manager element.

Table B–40 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). The 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). The default value is 1024MB.
<directory>	Optional	Specifies the path name for the root directory that the manager uses to store files in. If not specified or specifies a non-existent directory, a temporary file in the default location is 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 17-2 for an example of an NIO cache configuration.

Note: JVMs require the use of a command line parameter if the total NIO buffers is 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 B–41 describes the subelements of the nio-memory-manager element.

Table B–41 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 B–41 (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). The 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). The default value is 1024MB.

operation-bundling

Used in: cachestore-scheme, distributed-scheme, remote-cache-scheme.

Description

The operation-bundling element specifies the configuration information 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.
- Operation bundling affects cache store operations. If operation bundling is configured, the CacheStore.storeAll() method is always called even if there is only one ripe entry.

See com.tangosol.net.cache.AbstractBundler for additional implementation details.

Elements

Table B–42 describes the subelement for the operation-bundling element.

Table B–42 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, 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 "Optimistic Cache" on page 11-7 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>` subelement). For instance, an optimistic cache which uses a `local-scheme` for its backing map results in cache entries being stored in-memory.

Elements

Table B–43 describes the subelements of the `optimistic-scheme` element.

Table B–43 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 "Using Scheme Inheritance" on page 12-9 for more information.
`<service-name>`	Optional	Specifies the name of the service which manages caches created from this scheme. Services are configured from within the `<services>` parameter in `tangosol-coherence.xml`. See Appendix A, "Operational Configuration Elements" for more information.
`<serializer>`	Optional	Specifies either: the class configuration information for a `com.tangosol.io.Serializer` implementation used to serialize and deserialize user types, or it references a serializer class configuration that is defined in the operational configuration file (see "serializer" on page A-56).

Table B–43 (Cont.) optimistic-scheme Subelements

Element	Required/ Optional	Description
`<request-timeout>`	Optional	Specifies the maximum amount of time a client waits 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). The default value is the value specified in the `tangosol-coherence.xml` descriptor. See the `request-timeout` parameter in "ReplicatedCache Service Parameters" on page A-65 for more information.
`<guardian-timeout>`	Optional	Specifies the guardian timeout value to use for guarding the service and any dependent threads. If the element is not specified for a given service, the default guardian timeout (as specified by the `<timeout-milliseconds>` operational configuration element) is used. See <service-guardian>. 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.
`<service-failure-policy>`	Optional	Specifies the action to take when an abnormally behaving service thread cannot be terminated gracefully by the service guardian. Legal values are: ■ `exit-cluster` (default) – attempts to recover threads that appear to be unresponsive. If the attempt fails, an attempt is made to stop the associated service. If the associated service cannot be stopped, this policy causes the local node to stop the cluster services. ■ `exit-process` – attempts to recover threads that appear to be unresponsive. If the attempt fails, an attempt is made to stop the associated service. If the associated service cannot be stopped, this policy cause the local node to exit the JVM and terminate abruptly. ■ `logging` – causes any detected problems to be logged, but no corrective action to be taken. ■ a custom class – an `<instance>` subelement is used to provide the class configuration information for a `com.tangosol.net.ServiceFailurePolicy` implementation.

Table B–43 (Cont.) optimistic-scheme Subelements

Element	Required/ Optional	Description
<member-listener>	Optional	Specifies the configuration information for a class that implements the com.tangosol.net.MemberListener interface. The implementation must have a public default constructor. The MemberListener implementation receives cache service lifecycle events. The <member-listener> element is used as an alternative to programmatically adding a MapListener on a service.
<backing-map-scheme>	Optional	Specifies what type of cache is used within the cache server to store the entries. Legal values are: ■ local-scheme ■ external-scheme ■ paged-external-scheme ■ overflow-scheme ■ class-scheme ■ flashjournal-scheme ■ ramjournal-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.
<listener>	Optional	Specifies an implementation of a com.tangosol.util.MapListener which is notified of events occurring on the cache.
<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. The default value is false.

outgoing-message-handler

Used in: `acceptor-config`, `initiator-config`.

Description

The `outgoing-message-handler` specifies the configuration information used to detect dropped client-to-cluster connections. For connection initiators and acceptors that use connectionless protocols, this information is necessary to 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 B–44 describes the subelements of the outgoing-message-handler element.

Table B–44 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 (`0s`) for clustered client requests and 30 seconds (`30s`) for non-clustered client requests.

overflow-scheme

Used in: caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme, read-write-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 than the JVM heap allows. In such a deployment, the local-scheme element's high-units and eviction-policy controls 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> subelement.

Elements

Table B–45 describes the subelements of the overflow-scheme element.

Table B–45 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 "Using Scheme Inheritance" on page 12-9 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.
<front-scheme>	Required	Specifies the cache to use as the front-tier cache.
<back-scheme>	Required	Specifies the cache-scheme to use in creating the back-tier cache.

Table B–45 (Cont.) overflow-scheme Subelements

Element	Required/ Optional	Description
<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 is maintained. Legal values are: ■ local-scheme
<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 is implemented using com.tangosol.net.cache.OverflowMap, rather then com.tangosol.net.cache.SimpleOverflowMap. Legal values are true or false. The default value is false.
<expiry-delay>	Optional	Specifies the amount of time since the last update that entries are kept by the cache before being expired. Entries that have expired are not be accessible and are evicted the next time a client accesses the cache. 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. The default value is 0. **Note:** The expiry delay parameter (cExpiryMillis) is defined as an integer and is expressed in milliseconds. Therefore, the maximum amount of time can never exceed Integer.MAX_VALUE (2147483647) milliseconds or approximately 24 days.
<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. The default value is false.
<listener>	Optional	Specifies an implementation of a com.tangosol.util.MapListener which is notified of events occurring on the cache.

paged-external-scheme

Used in: caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme, near-scheme, overflow-scheme, read-write-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 Chapter 15, "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 configured 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 evicted. For example configuring a cache with a duration of ten minutes per page, and a maximum of six pages, results 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 used for long-term storage (persistence) and do 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 B-38.

Elements

Table B–46 describes the subelements of the `paged-external-scheme` element.

Table B–46 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 "Using Scheme Inheritance" on page 12-9 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.
`<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 B-37 for more information.
`<bdb-store-manager>`	Optional	Configures the paged external cache to use Berkeley Database JE on disk databases for cache storage.
`<custom-store-manager>`	Optional	Configures the paged external cache to use a custom storage manager implementation.
`<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.

Table B–46 (Cont.) paged-external-scheme Subelements

Element	Required/ Optional	Description
`<page-limit>`	Optional	Specifies the maximum number of pages that the cache manages before older pages are destroyed. Legal values are zero or positive integers between 2 and 3600. The default value is zero.
`<page-duration>`	Optional	Specifies the length of time, in seconds, that a page in the cache is current. After the duration is exceeded, the page is closed and a new current page is created. 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 zero or values between 5 and 604800 seconds (one week). The default value is zero.
<listener>	Optional	Specifies an implementation of a `com.tangosol.util.MapListener` which is notified of events occurring on the cache.

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 B–47 describes the subelements of the `partition-listener` element.

Table B–47 partition-listener Subelements

Element	Required/ Optional	Description
`<class-name>`	Required	The name of a class that implements the `PartitionListener` interface. This implementation must have a zero-parameter public constructor. The 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 A-59 for more information.
`<class-factory-name>`	Optional	Specifies the fully qualified name of a factory class for creating implementation class instances. This element cannot be used with the `<class-name>` element and is used with the `<method-name>` element.
`<method-name>`	Optional	Specifies the name of a static factory method on the factory class which performs object instantiation.
`<init-params>`	Optional	Contains class initialization parameters for the implementation class.

partitioned-quorum-policy-scheme

Used in: distributed-scheme

Description

The `partitioned-quorum-policy-scheme` element contains quorum policy settings for the partitioned cache service.

Elements

Table B–48 describes the subelements of the `partitioned-quorum-policy-scheme` element.

Table B–48 partitioned-quorum-policy-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 "Using Scheme Inheritance" on page 12-9 for more information.
`<distribution-quorum>`	Optional	Specifies the minimum number of ownership-enabled members of a partitioned service that must be present to perform partition distribution. The value must be a nonnegative integer.
`<restore-quorum>`	Optional	Specifies the minimum number of ownership-enabled members of a partitioned service that must be present to restore lost primary partitions from backup. The value must be a nonnegative integer.
`<read-quorum>`	Optional	specifies the minimum number of storage members of a cache service that must be present to process "read" requests. A "read" request is any request that does not mutate the state or contents of a cache. The value must be a nonnegative integer.
`<write-quorum>`	Optional	specifies the minimum number of storage members of a cache service that must be present to process "write" requests. A "write" request is any request that may mutate the state or contents of a cache. The value must be a nonnegative integer.
`<class-name>`	Optional	Specifies a class that provides custom quorum policies. This element cannot be used with the default quorum elements or the `<class-factory-name>` element. The class must implement the `com.tangosol.net.ActionPolicy` interface. Initialization parameters can be specified using the `<init-params>` element.
`<class-factory-name>`	Optional	Specifies a factory class for creating custom action policy instances. This element cannot be used with the default quorum elements or the `<class-name>` element. This element is used with the `<method-name>` element. The action policy instances must implement the `com.tangosol.net.ActionPolicy` interface.
`<method-name>`	Optional	Specifies the name of a static factory method on the factory class which performs object instantiation.
`<init-params>`	Optional	Contains class initialization parameters for the implementation class.

provider

Used in: ssl, identity-manager, trust-manager.

Description

The provider element contains the configuration information for a security provider that extends the `java.security.Provider` class.

Elements

Table B–49 describes the subelements you can define within the `provider` element.

Table B–49 provider Subelements

Element	Required/ Optional	Description
`<name>`	Optional	Specifies the name of a security provider that extends the `java.security.Provider` class. The class name can be entered using either this element or by using the `<class-name>` element or by using the `<class-factory-name>` element.
`<class-name>`	Optional	Specifies the name of a security provider that extends the `java.security.Provider` class. This element cannot be used with the `<name>` element or the `<class-factory-name>` element.
`<class-factory-name>`	Optional	Specifies a factory class for creating `Provider` instances. The instances must implement the `java.security.Provider` class. This element cannot be used with the `<name>` element or the `<class-name>` element. This element can be used with the `<method-name>` element.
`<method-name>`	Optional	Specifies the name of a static factory method on the factory class which performs object instantiation.
`<init-params>`	Optional	Contains class initialization parameters for the provider implementation. This element cannot be used with the `<name>` element.

proxy-config

Used in: proxy-scheme.

Description

The `proxy-config` element specifies the configuration information 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 B–50 describes the subelements of the `proxy-config` element.

Table B–50 proxy-config Subelements

Element	Required/ Optional	Description
<cache-service-proxy>	Optional	Specifies the configuration information for a cache service proxy managed by the proxy service.
<invocation-service-proxy>	Optional	Specifies the configuration information for an invocation service proxy managed by the proxy service.

proxy-scheme

Used in: caching-schemes.

Description

The proxy-scheme element contains the configuration information 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 B–51 describes the subelements of the proxy-scheme element.

Table B–51 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 "Using Scheme Inheritance" on page 12-9 for more information.
<service-name>	Optional	Specifies the name of the service.
<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. The default value is the value specified in the thread-count parameter of the tangosol-coherence.xml descriptor. See "ProxyService Parameters" on page A-67 for more information.
<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). The default value is the value specified in the tangosol-coherence.xml descriptor. See the task-hung-threshold parameter in "ProxyService Parameters" on page A-67 for more information.
<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). If zero is specified, the default service-guardian <timeout-milliseconds> value is used. Legal values are nonnegative integers. The default value is the value specified in the tangosol-coherence.xml descriptor. See the task-timeout parameter in "ProxyService Parameters" on page A-67.

Table B–51 (Cont.) proxy-scheme Subelements

Element	Required/ Optional	Description
`<request-timeout>`	Optional	Specifies the maximum amount of time a client waits 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). The default value is the value specified in the `tangosol-coherence.xml` descriptor. See the `request-timeout` parameter in "ProxyService Parameters" on page A-67 for more information.
`<guardian-timeout>`	Optional	Specifies the guardian timeout value to use for guarding the service and any dependent threads. If the element is not specified for a given service, the default guardian timeout (as specified by the `<timeout-milliseconds>` operational configuration element) is used. See <service-guardian>. 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.
`<service-failure-policy>`	Optional	Specifies the action to take when an abnormally behaving service thread cannot be terminated gracefully by the service guardian. Legal values are: ■ `exit-cluster` (default) – attempts to recover threads that appear to be unresponsive. If the attempt fails, an attempt is made to stop the associated service. If the associated service cannot be stopped, this policy causes the local node to stop the cluster services. ■ `exit-process` – attempts to recover threads that appear to be unresponsive. If the attempt fails, an attempt is made to stop the associated service. If the associated service cannot be stopped, this policy cause the local node to exit the JVM and terminate abruptly. ■ `logging` – causes any detected problems to be logged, but no corrective action to be taken. ■ a custom class – an `<instance>` subelement is used to provide the class configuration information for a `com.tangosol.net.ServiceFailurePolicy` implementation.

Table B–51 (Cont.) proxy-scheme Subelements

Element	Required/ Optional	Description
<member-listener>	Optional	Specifies the configuration information for a class that implements the com.tangosol.net.MemberListener interface. The implementation must have a public default constructor. The MemberListener implementation receives service lifecycle events. The <member-listener> element is used as an alternative to programmatically adding a MapListener on a service.
<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.
<load-balancer>	Optional	Specifies a pluggable strategy used by the proxy service to distribute client connections across the set of clustered proxy service members. Legal values are: ▪ proxy – (default) This strategy attempts to distribute client connections equally across proxy service members based upon existing connection count, connection limit, incoming and outgoing message backlog, and daemon pool utilization. ▪ client – This strategy relies upon the client address provider implementation to dictate the distribution of clients across proxy service members. If no client address provider implementation is provided, the extend client tries each proxy service in a random order until a connection is successful. ▪ a custom class – an <instance> subelement is used to provide the configuration information for a class that implements the com.tangosol.net.proxy.ProxyServiceLoadBalancer interface.
<proxy-quorum-policy-scheme>	Optional	Specifies quorum policy settings for the Proxy 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. The default value is false.

proxy-quorum-policy-scheme

Used in: proxy-scheme

Description

The `proxy-quorum-policy-scheme` element contains quorum policy settings for the Proxy service.

Elements

Table B–51 describes the subelements of the `proxy-quorum-policy-scheme` element.

Table B–52 proxy-quorum-policy-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 "Using Scheme Inheritance" on page 12-9 for more information.
`<connect-quorum>`	Optional	specifies the minimum number of members of a proxy service that must be present to allow client connections. The value must be a nonnegative integer.
`<class-name>`	Optional	Specifies a class that provides custom quorum policies. This element cannot be used with the `<connect-quorum>` element or the `<class-factory-name>` element. The class must implement the `com.tangosol.net.ActionPolicy` interface. Initialization parameters can be specified using the `<init-params>` element.
`<class-factory-name>`	Optional	Specifies a factory class for creating custom action policy instances. This element cannot be used with the `<connect-quorum>` element or the `<class-name>` element. This element is used with the `<method-name>` element. The action policy instances must implement the `com.tangosol.net.ActionPolicy` interface.
`<method-name>`	Optional	Specifies the name of a static factory method on the factory class which performs object instantiation.
`<init-params>`	Optional	Contains class initialization parameters for the implementation class.

ramjournal-scheme

Used in: back-scheme, backing-map-scheme, caching-schemes, front-scheme, internal-cache-scheme

Description

The ramjournal-scheme element contains the configuration information for a scheme that stores data to buffers (journal files) in-memory. A RAM journal resource manager controls RAM journal behavior. See "ramjournal-manager" on page A-53 for additional details on configuring RAM journal behavior.

This scheme uses the com.tangosol.net.cache.SimpleSerializationMap class as the backing map implementation and the com.tangosol.io.journal.JournalBinaryStore to store and retrieve binary key value pairs to a journal.

Elements

Table B–53 describes the subelements of the ramjournal-scheme element.

Table B–53 ramjournal-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 "Using Scheme Inheritance" on page 12-9 for more information.
<class-name>	Optional	Specifies a custom implementation of the simple serialization map cache. Any custom implementation must extend the com.tangosol.net.cache.SimpleSerializationMap class and declare the exact same set of public constructors as the superclass.
<init-params>	Optional	Specifies the initialization parameters for a custom serialization map cache.
<listener>	Optional	Specifies an implementation of a com.tangosol.util.MapListener which is notified of events occurring on the cache.

read-write-backing-map-scheme

Used in: caching-schemes, distributed-scheme.

Description

The read-write-backing-map-scheme defines a backing map which provides a size limited cache of a persistent store. See Chapter 14, "Caching Data Sources" 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 are read-through to the back-end store to retrieve the data. If a writable store is provided, cache writes are also propagate to the cache store.

Refresh-Ahead Caching

When enabled (see <refreshahead-factor> subelement) the cache watches 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 delays 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

Table B–54 describes the subelements of the read-write-backing-map-scheme element.

Table B–54 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 "Using Scheme Inheritance" on page 12-9 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 B–54 (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.
<internal-cache-scheme>	Required	Specifies a cache-scheme which is used to cache entries. Legal values are: ■ local-scheme ■ external-scheme ■ paged-external-scheme ■ overflow-scheme ■ class-scheme ■ flashjournal-scheme ■ ramjournal-scheme
<write-max-batch-size>	Optional	Specifies the maximum number of entries to write in a single storeAll operation. Valid values are positive integers or zero. The default value is 128 entries. This value has no effect if write behind is disabled.
<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 is maintained. Legal values are: ■ local-scheme
<cachestore-scheme>	Optional	Specifies the store to cache. If unspecified the cached data only resides within the internal cache (see <internal-cache-scheme> subelement), and only reflect operations performed on the cache itself.
<read-only>	Optional	Specifies if the cache is read only. If true the cache loads data from cachestore for read operations and do not perform any writing to the cachestore when the cache is updated. Legal values are true or false. The default value is false.

Table B–54 (Cont.) read-write-backing-map-scheme Subelements

Element	Required/ Optional	Description
`<write-delay>`	Optional	Specifies the time interval to defer asynchronous writes to the cachestore for a write-behind queue. 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) take place, otherwise the writes are asynchronous and deferred by specified time interval after the last update to the value in the cache. The default value is zero. This element cannot be used with the `<write-delay-seconds>` element.
`<write-delay-seconds>`	Optional	Specifies the number of seconds to defer asynchronous writes to the cachestore for a write-behind queue. If zero, synchronous writes to the cachestore (without queueing) take place; otherwise, the writes are asynchronous and deferred by the number of seconds after the last update to the value in the cache. This element cannot be used with the `<write-delay>` element.

Table B–54 (Cont.) read-write-backing-map-scheme Subelements

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 is 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) * D where D = write-delay interval and F = 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). **Note:** if operation bundling (<operation-bundling>) is configured, then storeAll() is always called even if there is only a single ripe entry. 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. The default value is zero.
<write-requeue-threshold>	Optional	Specifies the size of the write-behind queue at which additional actions could be taken. If zero, write-behind requeuing is disabled. Otherwise, this value controls the frequency of the corresponding log messages. For example, a value of 100 produces a log message every time the size of the write queue is a multiple of 100. Legal values are positive integers or zero. The default value 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 schedules 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 is disabled. If 1.0, then any get operation immediately triggers an asynchronous reload. Legal values are nonnegative doubles less than or equal to 1.0. The default value is zero.

Table B–54 (Cont.) read-write-backing-map-scheme Subelements

Element	Required/ Optional	Description
<cachestore-timeout>	Optional	Specifies the timeout interval to use for CacheStore read and write operations. If a CacheStore operation times out, the executing thread is interrupted and may ultimately lead to the termination of the cache service. Timeouts of asynchronous CacheStore operations (for example, refresh-ahead, write-behind) do not result in service termination. 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. If 0 is specified, the default service-guardian <timeout-milliseconds> value is used. The default value if none is specified is 0.
<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). Legal values are true or false. 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. The default value is true.
<listener>	Optional	Specifies an implementation of a com.tangosol.util.MapListener which is notified of events occurring on the cache.

remote-addresses

Used in: tcp-initiator

Description

The `remote-addresses` element contains the address (IP or DNS name) and port of one or more TCP/IP acceptors. The TCP/IP initiator uses this information to establish a connection with a proxy service on remote cluster. TCP/IP acceptors are configured within the proxy-scheme element. The TCP/IP initiator attempts to connect to the addresses in a random order until either the list is exhausted or a TCP/IP connection is established. See *Oracle Coherence Client Guide* for additional details and example configurations.

The following example configuration instructs the initiator 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>
```

Elements

Table B–57 describes the subelements of the `remote-addresses` element.

Table B–55 remote-addresses Subelements

Element	Required/ Optional	Description
`<socket-address>`	Optional	Specifies the address (IP or DNS name) and port on which a TCP/IP acceptor is listening. Multiple `<socket-address>` elements can be used within a `<remote-addresses>` element.
`<address-provider>`	Optional	Contains the configuration for a `com.tangosol.net.AddressProvider` implementation that supplies the address (IP or DNS name) and port on which the TCP/IP acceptor is listening. A `<remote-addresses>` element can include either a `<socket-address>` element or an `<address-provider>` element but not both.

remote-cache-scheme

Used in: cachestore-scheme, caching-schemes, near-scheme.

Description

The remote-cache-scheme element contains the configuration information necessary to use a clustered cache from outside the cluster by using Coherence*Extend.

Elements

Table B–56 describes the subelements of the remote-cache-scheme element.

Table B–56 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 "Using Scheme Inheritance" on page 12-9 for more information.
<service-name>	Optional	Specifies the name of the service which manages caches created from this scheme.
<operation-bundling>	Optional	Specifies the configuration information 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.
<defer-key-association-check>	Optional	Specifies whether key association processing is done by the extend client or deferred to the cluster side. Valid values are true and false. The default value is false and indicates that key association processing is done by the extend client. If the value is set to true, NET and C++ clients must include a parallel Java implementation of the key class on the cluster cache servers.

remote-invocation-scheme

Used in: `caching-schemes`

Description

The `remote-invocation-scheme` element contains the configuration information necessary to execute tasks within a cluster without having to first join the cluster. This scheme uses Coherence*Extend to connect to the cluster.

Elements

Table B–57 describes the subelements of the `remote-invocation-scheme` element.

Table B–57 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 "Using Scheme Inheritance" on page 12-9 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, overflow-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" on page 11-5 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 results in cache entries being stored in-memory.

Elements

Table B–58 describes the subelements of the replicated-scheme element.

Table B–58 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 "Using Scheme Inheritance" on page 12-9 for more information.
<service-name>	Optional	Specifies the name of the service which manages caches created from this scheme. Services are configured from within the <services> element in the tangosol-coherence.xml file. See Appendix A, "Operational Configuration Elements" for more information.
<serializer>	Optional	Specifies either: the class configuration information for a com.tangosol.io.Serializer implementation used to serialize and deserialize user types, or it references a serializer class configuration that is defined in the operational configuration file (see "serializer" on page A-56).

Table B–58 (Cont.) replicated-scheme Subelements

Element	Required/ Optional	Description
`<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 is automatically 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. The default value is the value specified for `packet-delivery/timeout-milliseconds` in the `tangosol-coherence.xml` descriptor. See "ReplicatedCache Service Parameters" on page A-65 for more information.
`<lease-granularity>`	Optional	Specifies the lease ownership granularity. 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. The default value is the `lease-granularity` value specified in the `tangosol-coherence.xml` descriptor. See "ReplicatedCache Service Parameters" on page A-65 for more information.
`<request-timeout>`	Optional	Specifies the maximum amount of time a client waits 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). The default value is the value specified in the `tangosol-coherence.xml` descriptor. See the `request-timeout` parameter in "ReplicatedCache Service Parameters" on page A-65 for more information.

Table B–58 (Cont.) replicated-scheme Subelements

Element	Required/ Optional	Description
`<guardian-timeout>`	Optional	Specifies the guardian timeout value to use for guarding the service and any dependent threads. If the element is not specified for a given service, the default guardian timeout (as specified by the `<timeout-milliseconds>` operational configuration element) is used. See <service-guardian>. 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.
`<service-failure-policy>`	Optional	Specifies the action to take when an abnormally behaving service thread cannot be terminated gracefully by the service guardian. Legal values are: ■ `exit-cluster` (default) – attempts to recover threads that appear to be unresponsive. If the attempt fails, an attempt is made to stop the associated service. If the associated service cannot be stopped, this policy causes the local node to stop the cluster services. ■ `exit-process` – attempts to recover threads that appear to be unresponsive. If the attempt fails, an attempt is made to stop the associated service. If the associated service cannot be stopped, this policy cause the local node to exit the JVM and terminate abruptly. ■ `logging` – causes any detected problems to be logged, but no corrective action to be taken. ■ a custom class – an `<instance>` subelement is used to provide the class configuration information for a `com.tangosol.net.ServiceFailurePolicy` implementation.
`<member-listener>`	Optional	Specifies the configuration information for a class that implements the `com.tangosol.net.MemberListener` interface. The implementation must have a public default constructor. The `MemberListener` implementation receives cache service lifecycle events. The `<member-listener>` element is used as an alternative to programmatically adding a `MapListener` on a service.

Table B–58 (Cont.) replicated-scheme Subelements

Element	Required/ Optional	Description
<backing-map-scheme>	Optional	Specifies what type of cache is used within the cache server to store the entries. Legal values are: ■ local-scheme ■ external-scheme ■ paged-external-scheme ■ overflow-scheme ■ class-scheme ■ flashjournal-scheme ■ ramjournal-scheme To ensure cache coherence, the backing-map of a 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.
<listener>	Optional	Specifies an implementation of a com.tangosol.util.MapListener which is notified of events occurring on the cache.
<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. The default value is false.

serializer

Used in: acceptor-config, defaults, distributed-scheme, initiator-config, invocation-scheme, optimistic-scheme, replicated-scheme, transactional-scheme,

Description

The serializer element contains the class configuration information for a `com.tangosol.io.Serializer` implementation.

The `serializer` element accepts either a reference to a serializer configuration or a full serializer configuration. The best practice is to reference a configuration which is defined in the operational configuration file. The operational configuration file contains two pre-defined serializer class configuration: one for Java (default) and one for POF. See "serializer" on page A-56.

The following example demonstrates referring to the POF serializer definition that is in the operational configuration file:

```
...
<serializer>pof</serializer>
...
```

The following example demonstrates a full serializer class configuration:

```
...
<serializer>
   <instance>
      <class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>
      <init-params>
         <init-param>
            <param-type>String</param-type>
            <param-value>my-pof-config.xml</param-value>
         </init-param>
      </init-params>
   </instance>
</serializer>
...
```

Elements

Table B–59 describes the subelements of the `serializer` element.

Table B–59 serializer Subelements

Element	Required/ Optional	Description
`<instance>`	Optional	Contains the class configuration information for a `com.tangosol.io.Serializer` implementation.

socket-address

Used in: remote-addresses

Description

The `socket-address` element specifies the address and port on which a TCP/IP acceptor is listening.

Elements

Table B–60 describes the subelements of the `socket-address` element.

Table B–60 socket-address Subelements

Element	Required/ Optional	Description
`<address>`	Required	Specifies the IP address (IP or DNS name) on which a TCP/IP acceptor socket is listening.
`<port>`	Required	Specifies the port on which a TCP/IP acceptor socket is listening. Legal values are from 1 to 65535.

socket-provider

Used in: tcp-acceptor, tcp-initiator, defaults.

Description

The <socket-provider> element contains the configuration information for a socket and channel factory that implements the com.tangosol.net.SocketProvider interface. The socket providers that are configured within the <tcp-acceptor> and <tcp-initiator> elements are for use with Coherence*Extend. Socket providers for TCMP are configured in an operational override for within the <unicast-listener> element.

The <socket-provider> element accepts either a reference to a socket provider configuration or a full socket provider configuration. The best practice is to reference a configuration which is defined in the operational configuration file. See "socket-providers" on page A-75.

Out-of-box, the operational configuration file contains two pre-defined socket provider configurations: system (default) and ssl. Additional socket providers can be defined in an operational override file as required. Socket provider configurations are referred to using their id attribute name. The following example refers to the pre-defined SSL socket provider configuration:

```
<socket-provider>ssl</socket-provider>
```

Preconfigured override is tangosol.coherence.socketprovider.

Elements

Table B–61 describes the subelements you can define within the socket-provider element.

Table B–61 socket-provider Subelements

Element	Required/ Optional	Description
<system>	Optional	Specifies a socket provider that produces instances of the JVM's default socket and channel implementations.
<ssl>	Optional	Specifies a socket provider that produces socket and channel implementations which use SSL.
<instance>	Optional	Contains the class configuration information for a com.tangosol.net.SocketProvider implementation.

ssl

Used in: socket-provider.

Description

The <ssl> element contains the configuration information for a socket provider that produces socket and channel implementations which use SSL.

Elements

Table B–62 describes the elements you can define within the ssl element.

Table B–62 ssl Subelements

Element	Required/ Optional	Description
<protocol>	Optional	Specifies the name of the protocol used by the socket and channel implementations produced by the SSL socket provider. The default value is TLS.
<provider>	Optional	Specifies the configuration for a security provider instance.
<executor>	Optional	Specifies the configuration information for an implementation of the java.util.concurrent.Executor interface. A <class-name> subelement is used to provide the name of a class that implements the Executor interface. As an alternative, a <class-factory-name> subelement can specify a factory class for creating Executor instances and a <method-name> subelement that specifies the name of a static factory method on the factory class which performs object instantiation. Either approach can specify initialization parameters using the <init-params> element.
<identity-manager>	Optional	Specifies the configuration information for initializing an identity manager instance.
<trust-manager>	Optional	Specifies the configuration information for initializing a trust manager instance.
<hostname-verifier>	Optional	Specifies the configuration information for an implementation of the javax.net.ssl.HostnameVerifier interface. During the SSL handshake, if the URL's host name and the server's identification host name mismatch, the verification mechanism calls back to this instance to determine if the connection should be allowed. A <class-name> subelement is used to provide the name of a class that implements the HostnameVerifier interface. As an alternative, a <class-factory-name> subelement can specify a factory class for creating HostnameVerifier instances and a <method-name> subelement that specifies the name of a static factory method on the factory class which performs object instantiation. Either approach can specify initialization parameters using the <init-params> element.

tcp-acceptor

Used in: acceptor-config.

Description

The `tcp-acceptor` element specifies the configuration information for a connection acceptor that accepts connections from Coherence*Extend clients over TCP/IP. See *Oracle Coherence Client Guide* for additional details and example configurations.

Elements

Table B–63 describes the subelements of the `tcp-acceptor` element.

Table B–63 tcp-acceptor Subelements

Element	Required/ Optional	Description
`<socket-provider>`	Optional	Specifies the configuration information for a socket and channel factory that implements the `com.tangosol.net.SocketProvider` interface.
`<local-address>`	Optional	Specifies the local address (IP or DNS name) and port on which the TCP/IP server socket (opened by the connection acceptor) is bound.
`<address-provider>`	Optional	Contains the configuration for a `com.tangosol.net.AddressProvider` implementation that supplies the local address (IP or DNS name) and port on which the TCP/IP server socket (opened by the connection acceptor) listens. A `<tcp-acceptor>` element can include either a `<local-address>` or an `<address-provider>` element but not both.
`<reuse-address>`	Optional	Specifies whether a TCP/IP socket can be bound to an in-use or recently used address. This setting is deprecated because the resulting behavior is significantly different across operating system implementations. The JVM, in general, selects a reasonable default which is safe for the target operating system. Valid values are `true` and `false`. The default value depends on the operating system.
`<keep-alive-enabled>`	Optional	Indicates whether `SO_KEEPALIVE`) is enabled on a TCP/IP socket. Valid values are `true` and `false`. The default value is `true`.
`<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 B–63 (Cont.) tcp-acceptor 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. The default value is O/S dependent.
`<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. The 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. The 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.
`<suspect-protocol-enabled>`	Optional	Specifies whether the suspect protocol is enabled to detect and close rogue Coherence*Extend client connections. The suspect protocol is enabled by default. Valid values are `true` and `false`.

Table B–63 (Cont.) tcp-acceptor Subelements

Element	Required/ Optional	Description
`<suspect-buffer-size>`	Optional	Specifies the outgoing connection backlog (in bytes) after which the corresponding client connection is marked as suspect. A suspect client connection is then monitored until it is no longer suspect or it is closed to protect the proxy server from running out of memory. The value of this element must be in the following format: `[\d]+[[.][\d]+]?[K|k|M|m|G|g|T|t]?[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, 2^10) ■ M or m (mega, 2^20) ■ G or g (giga, 2^30) ■ T or t (tera, 2^40) If the value does not contain a factor, a factor of one is assumed. The default value is 10000000.
`<suspect-buffer-length>`	Optional	Specifies the outgoing connection backlog (in messages) after which the corresponding client connection is marked as suspect. A suspect client connection is then monitored until it is no longer suspect or it is closed to protect the proxy server from running out of memory. The default value is `10000`.
`<nominal-buffer-size>`	Optional	Specifies the outgoing connection backlog (in bytes) at which point a suspect client connection is no longer considered to be suspect. The value of this element must be in the following format: `[\d]+[[.][\d]+]?[K|k|M|m|G|g|T|t]?[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, 2^10) ■ M or m (mega, 2^20) ■ G or g (giga, 2^30) ■ T or t (tera, 2^40) If the value does not contain a factor, a factor of one is assumed. The default value is 2000000.

Table B–63 (Cont.) tcp-acceptor Subelements

Element	Required/ Optional	Description
`<nominal-buffer-length>`	Optional	Specifies the outgoing connection backlog (in messages) at which point a suspect client connection is no longer considered to be suspect. The default value is `2000`.
`<limit-buffer-size>`	Optional	Specifies the outgoing connection backlog (in bytes) at which point the corresponding client connection must be closed to protect the proxy server from running out of memory. The value of this element must be in the following format: `[\d]+[[.][\d]+]?[K|k|M|m|G|g|T|t]?[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, 2^10) ■ M or m (mega, 2^20) ■ G or g (giga, 2^30) ■ T or t (tera, 2^40) If the value does not contain a factor, a factor of one is assumed. The default value is 100000000.
`<limit-buffer-length>`	Optional	Specifies the outgoing connection backlog (in messages) at which point the corresponding client connection must be closed to protect the proxy server from running out of memory. The default value is `60000`.

tcp-initiator

Used in: initiator-config.

Description

The tcp-initiator element specifies the configuration information for a connection initiator that enables Coherence*Extend clients to connect to a remote cluster by using TCP/IP. See *Oracle Coherence Client Guide* for additional details and example configurations.

Elements

Table B–64 describes the subelements of the tcp-initiator element.

Table B–64 tcp-initiator Subelements

Element	Required/ Optional	Description
<socket-provider>	Optional	Specifies the configuration information for a socket and channel factory that implements the com.tangosol.net.SocketProvider interface.
<local-address>	Optional	Specifies the local address (IP or DNS name) and port on which the TCP/IP client socket (opened by the TCP/IP initiator) is bound.
<remote-addresses>	Required	Contains the 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.
<reuse-address>	Optional	Specifies whether a TCP/IP socket can be bound to an in-use or recently used address. This setting is deprecated because the resulting behavior is significantly different across operating system implementations. The JVM, in general, selects a reasonable default which is safe for the target operating system. Valid values are true and false. The default value depends on the operating system.
<keep-alive-enabled>	Optional	Indicates whether SO_KEEPALIVE is enabled on a TCP/IP socket. Valid values are true and false. The default value is true.
<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. The default value is O/S dependent.

Table B–64 (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. The 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. The 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.

transactional-scheme

Used in caching-schemes

Description

The `transactional-scheme` element defines a transactional cache, which is a specialized distributed cache that provides transactional guarantees. Multiple `transactional-scheme` elements may be defined to support different configurations. Applications use transactional caches in one of three ways:

- Applications use the `CacheFactory.getCache()` method to get an instance of a transactional cache. In this case, there are implicit transactional guarantees when performing cache operations. However, default transaction behavior cannot be changed.
- Applications explicitly use the Transaction Framework API to create a `Connection` instance that uses a transactional cache. In this case, cache operations are performed within a transaction and the application has full control to change default transaction behavior as required.
- Java EE applications use the Coherence Resource Adapter to create a Transaction Framework API `Connection` instance that uses a transactional cache. In this case, cache operations are performed within a transaction that can participate as part of a distributed (global) transaction. Applications can change some default transaction behavior.

Elements

Table B–65 describes the subelements of the `transactional-scheme` element.

Table B–65 transactional-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 "Using Scheme Inheritance" on page 12-9 for more information.
`<service-name>`	Optional	Specifies the name of the service which manages caches created from this scheme. The default service name if no service name is provided is `TransactionalCache`.
`<serializer>`	Optional	Specifies either: the class configuration information for a `com.tangosol.io.Serializer` implementation used to serialize and deserialize user types, or it references a serializer class configuration that is defined in the operational configuration file (see "serializer" on page A-56).
`<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. The default value is the `thread-count` value specified in the `tangosol-coherence.xml` descriptor. See the `thread-count` parameter in "DistributedCache Service Parameters" on page A-59 for more information. Specifying the thread-count value changes the default behavior of the Transactional Framework's internal transaction caches that are used for transactional storage and recovery.

Table B–65 (Cont.) transactional-scheme Subelements

Element	Required/ Optional	Description
<local-storage>	Optional	Specifies whether a cluster node contributes 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`. The 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 A-59 for more information.

<partition-count>	Optional	Specifies the number of partitions that a partitioned (distributed) cache is "chopped up" into. Each member running the partitioned cache service that has the local-storage (`<local-storage>` subelement) option set to `true` manages a "fair" (balanced) number of partitions.
		
		The number of partitions should be a prime number and sufficiently large such that a given partition is expected to be no larger than 50MB.
		
		The following are good defaults for sample service storage sizes:
		
		`service storage partition-count`
		`_____ _____`
		`100M 257`
		`1G 509`
		`10G 2039`
		`50G 4093`
		`100G 8191`
		
		A list of first 1,000 primes can be found at
		
		`http://primes.utm.edu/lists/`
		
		Valid values are positive integers. The default value is the value specified in the `tangosol-coherence.xml` descriptor. See the `partition-count` parameter "DistributedCache Service Parameters" on page A-59 for more information.

<high-units>	Optional	Specifies the transaction storage size. Once the transactional storage size is reached, an eviction policy is used that removes 25% of eligible entries from storage. The value of this element must be in the following format: `[\d]+[[.][\d]+]?[K|k|M|m|G|g|T|t]?[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, 2^10) ▪ `M` or `m` (mega, 2^20) ▪ `G` or `g` (giga, 2^30) ▪ `T` or `t` (tera, 2^40) If the value does not contain a factor, a factor of one is assumed. The default value is 10MB.

Table B–65 (Cont.) transactional-scheme Subelements

Element	Required/ Optional	Description
`<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 makes the distribution process take longer, but reduces network bandwidth utilization during this activity. Legal values are integers greater then zero. The 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 A-59 for more information.
`<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. A value of 0 means that for abnormal termination, some portion of the data in the cache is lost. Value of N means that if up to N cluster nodes terminate immediately, the cache data is 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 is in the order of M*(N+1). Recommended values are 0 or 1. The 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 A-59 for more information.
`<partition-assignment-strategy>`	Optional	Specifies the strategy used by a partitioned service to manage partition distribution. Valid values are `legacy` or a class that implements the com.`tangosol.net.partition.PartitionAssignmentStrategy` interface. The legacy assignment strategy indicates that partition distribution is managed individually on each cluster member. Whereas; a custom strategy allows for a shared strategy across the cluster. Enter the custom strategy using the `<instance>` element. The default value is `legacy`.
`<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). The 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 A-59 for more information.
`<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). If `zero` is specified, the default service-guardian `<timeout-milliseconds>` value is used. Legal values are nonnegative integers. The default value is the value specified in the `tangosol-coherence.xml` descriptor. See the `task-timeout` parameter in "DistributedCache Service Parameters" on page A-59.

Table B–65 (Cont.) transactional-scheme Subelements

Element	Required/ Optional	Description
`<request-timeout>`	Optional	Specifies the maximum amount of time a client waits 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). The default value is the value specified in the `tangosol-coherence.xml` descriptor. See the `request-timeout` parameter in "DistributedCache Service Parameters" on page A-59 for more information.
`<guardian-timeout>`	Optional	Specifies the guardian timeout value to use for guarding the service and any dependent threads. If the element is not specified for a given service, the default guardian timeout (as specified by the `<timeout-milliseconds>` operational configuration element) is used. See <service-guardian>. 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.

Table B–65 (Cont.) transactional-scheme Subelements

Element	Required/ Optional	Description
<service-failure-policy>	Optional	Specifies the action to take when an abnormally behaving service thread cannot be terminated gracefully by the service guardian. Legal values are: ■ exit-cluster (default) – attempts to recover threads that appear to be unresponsive. If the attempt fails, an attempt is made to stop the associated service. If the associated service cannot be stopped, this policy causes the local node to stop the cluster services. ■ exit-process – attempts to recover threads that appear to be unresponsive. If the attempt fails, an attempt is made to stop the associated service. If the associated service cannot be stopped, this policy cause the local node to exit the JVM and terminate abruptly. ■ logging – causes any detected problems to be logged, but no corrective action to be taken. ■ a custom class – an <instance> subelement is used to provide the class configuration information for a com.tangosol.net.ServiceFailurePolicy implementation.
<partitioned-quorum-policy-scheme>	Optional	Specifies quorum policy settings for the partitioned cache service.
<autostart>	Optional	The 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. The default value is false.

trust-manager

Used in: ssl.

Description

The <trust-manager> element contains the configuration information for initializing a javax.net.ssl.TrustManager instance.

A trust manager is responsible for managing the trust material that is used when making trust decisions and for deciding whether credentials presented by a peer should be accepted.

A valid trust-manager configuration contains at least one child element.

Elements

Table B–66 describes the elements you can define within the trust-manager element.

Table B–66 trust-manager Subelements

Element	Required/ Optional	Description
<algorithm>	Optional	Specifies the algorithm used by the trust manager. The default value is SunX509.
<provider>	Optional	Specifies the configuration for a security provider instance.
<key-store>	Optional	Specifies the configuration for a key store implementation.

Attribute Reference

Table B–67 describes the system property attribute that is available in the cache configuration deployment descriptor.

Table B–67 Cache Configuration Deployment Descriptor Attribute

Attribute	Required/ Optional	Description
`system-property`	Optional	This attribute is used to specify a system property name for any element. The system property is used to override the element value from the Java command line. This feature enables the same operational descriptor (and override file) to be used across all cluster nodes and customize each node using the system properties. See Appendix C, "Command Line Overrides," for more information on this feature.

C

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 have predefined overrides. You can create your own or change the predefined ones.

This feature is useful when you want 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, or port, or both 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, prefixed 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 IP address for the interface (for instance, 192.168.0.301). Call this override `tangosol.coherence.localhost`.

First, add a system-property to the `cluster-config`, `unicast-listener`, or `address` element. for example:

```
<address system-property="tangosol.coherence.localhost">localhost</address>
```

Then use it by modifying the Java command line and specifying an IP address instead of the default `localhost`:

```
java -Dtangosol.coherence.localhost=192.168.0.301 -jar coherence.jar
```

Preconfigured Override Values

Table C–1 lists all of the preconfigured override values:

Table C–1 Preconfigured System Property Override Values

Override Option	Setting
`tangosol.coherence.cacheconfig`	Cache configuration descriptor filename. See "configurable-cache-factory-config" on page A-13.
`tangosol.coherence.cluster`	Cluster name. See "member-identity" on page A-36.
`tangosol.coherence.clusteraddress`	Cluster (multicast) IP address. See `<multicast-listener-address>` subelement of "multicast-listener" on page A-39
`tangosol.coherence.clusterport`	Cluster (multicast) IP port. See `<multicast-listener-port>` subelement of "multicast-listener" on page A-39.
`tangosol.coherence.distributed.backup`	Data backup storage location. See `backup-storage/type` subelement in "DistributedCache Service Parameters" on page A-59.
`tangosol.coherence.distributed.backupcount`	Number of data backups. See `backup-count` subelement in "DistributedCache Service Parameters" on page A-59.
`tangosol.coherence.distributed.localstorage`	Local partition management enabled. See local-storage subelement in "DistributedCache Service Parameters" on page A-59.
`tangosol.coherence.distributed.threads`	Thread pool size. See `thread-count` subelement in "DistributedCache Service Parameters" on page A-59.
`tangosol.coherence.distributed.transfer`	Partition transfer threshold. See `transfer-threshold` subelement in"DistributedCache Service Parameters" on page A-59.
`tangosol.coherence.edition`	Product edition. See "license-config" on page A-28.
`tangosol.coherence.invocation.threads`	Invocation service thread pool size. See `thread-count` subelement in "InvocationService Parameters" on page A-66.
`tangosol.coherence.localhost`	Unicast IP address. See `<unicast-listener-address>` subelement in "unicast-listener" on page A-80.
`tangosol.coherence.localport`	Unicast IP port. See `<unicast-listener-port>` subelement in "unicast-listener" on page A-80.
`tangosol.coherence.localport.adjust`	Unicast IP port auto assignment. See `<unicast-listener-auto>` subelement in "unicast-listener" on page A-80.
`tangosol.coherence.log`	Logging destination. See `<logging-config-destination>` subelement in "logging-config" on page A-29.
`tangosol.coherence.log.level`	Logging level. See `<logging-config-level>` subelement in "logging-config" on page A-29.
`tangosol.coherence.log.limit`	Log output character limit. See `<logging-config-limit>` subelement in "logging-config" on page A-29.
`tangosol.coherence.machine`	The computer's name as defined by the `machine-name` element. See "member-identity" on page A-36.
`tangosol.coherence.management`	JMX management mode. See "management-config" on page A-30.

Table C–1 (Cont.) Preconfigured System Property Override Values

Override Option	Setting
`tangosol.coherence.management.readonly`	JMX management read-only flag. "management-config" on page A-30.
`tangosol.coherence.management.remote`	Remote JMX management enabled flag. See"management-config" on page A-30.
`tangosol.coherence.member`	Member name. See "member-identity" on page A-36.
`tangosol.coherence.mode`	Operational mode. See "license-config" on page A-28.
`tangosol.coherence.override`	Deployment configuration override filename.
`tangosol.coherence.priority`	Priority. See "member-identity" on page A-36.
`tangosol.coherence.process`	Process name"member-identity" on page A-36.
`tangosol.coherence.proxy.threads`	Coherence*Extend service thread pool size. See `thread-count` subelement in "ProxyService Parameters" on page A-67.
`tangosol.coherence.rack`	Rack name. See "member-identity" on page A-36.
`tangosol.coherence.role`	Role name. See "member-identity" on page A-36.
`tangosol.coherence.security`	Cache access security enabled flag. See "security-config" on page A-55.
`tangosol.coherence.security.keystore`	Security access controller keystore file name. See "security-config" on page A-55.
`tangosol.coherence.security.permissions`	Security access controller permissions file name. See "security-config" on page A-55.
`tangosol.coherence.shutdownhook`	Shutdown listener action. See "shutdown-listener" on page A-71.
`tangosol.coherence.site`	Site name. See "member-identity" on page A-36.
`tangosol.coherence.tcmp.enabled`	TCMP enabled flag. See `<packet-publisher-enabled>` subelement in "packet-publisher" on page A-48.
`tangosol.coherence.ttl`	Multicast packet time to live (TTL). See `<mulitcast-listener-ttl>` subelement in "multicast-listener" on page A-39.
`tangosol.coherence.wka`	Well known IP address. See "well-known-addresses" on page A-83.
`tangosol.coherence.wka.port`	Well known IP port. See "well-known-addresses" on page A-83.

D

POF User Type Configuration Elements

This appendix provides a detailed reference of the POF configuration deployment descriptor and includes a brief overview of the descriptor. See Appendix E, "The PIF-POF Binary Format," for details of the binary format.

The following sections are included in this appendix:

- POF Configuration Deployment Descriptor
- Element Index

POF Configuration Deployment Descriptor

The POF configuration deployment descriptor is used to specify non-intrinsic types, referred to as User Types, for objects that are being serialized and deserialized using POF. The name and location of the POF configuration deployment descriptor is specified in the operational deployment descriptor and defaults to `pof-config.xml`. A sample POF configuration deployment descriptor is located in the root of the `coherence.jar` library and is used unless a custom `pof-config.xml` file is found before the `coherence.jar` library within the application's classpath. All cluster members should use identical POF configuration deployment descriptors.

The POF configuration deployment descriptor schema is defined in the `coherence-pof-config.xsd` file. This XSD file is located in the root of the `coherence.jar` library and at the following Web URL:

`http://xmlns.oracle.com/coherence/coherence-pof-config/1.0/coherence-pof-config.xsd`

The `<pof-config>` element is the root element of the POF configuration deployment descriptor and typically includes an XSD and Coherence namespace reference and the location of the `coherence-pof-config.xsd` file. For example:

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

<pof-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-pof-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-pof-config
   coherence-pof-config.xsd">
```

Notes:

- The schema located in the `coherence.jar` library is always used at run time even if the `xsi:schemaLocation` attribute references the Web URL.
- The `xsi:schemaLocation` attribute can be omitted to disable schema validation.
- When deploying Coherence into environments where the default character set is EBCDIC rather than ASCII, ensure that the deployment descriptor file is in ASCII format and is deployed into its run-time environment in the binary format.

Coherence-specific user types are defined in the `coherence-pof-config.xml` file that is also located in the root of the `coherence.jar` library. This file should always be referenced as follows when creating a `pof-config.xml` file:

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

<pof-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-pof-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-pof-config
   coherence-pof-config.xsd">
   <user-type-list>
      <include>coherence-pof-config.xml</include>
   </user-type-list>
   ...
</pof-config>
```

Element Index

Table D–1 lists all non-terminal POF configuration deployment descriptor elements.

Table D–1 POF Configuration Elements

Element	Used In
<default-serializer>	<pof-config>
<init-param>	<init-params>
<init-params>	<serializer>
<pof-config>	*root element*
<serializer>	<user-type>
<user-type>	<user-type-list>
<user-type-list>	<pof-config>

default-serializer

Used in: <pof-config>

Description

This element specifies a PofSerializer to use when serializing and deserializing all user types defined within the pof-config element. If a serializer is specified within a user-type, then that serializer is used for that user-type instead of the default serializer.

If the default serializer element is omitted, the serializer defined for the specific user type is used. If the serializer for the user type is also 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 default 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)
- ()

Elements

Table D–2 describes the subelements of the default-serializer element.

Table D–2 default-serializer Subelements

Element	Required/ Optional	Description
<class-name>	Required	Specifies the fully qualified name of the PofSerializer implementation.
<init-params>	Optional	Specifies zero or more arguments (each as an init-param) that correspond to the parameters of a constructor of the class that is being configured.

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 D–3 describes the subelements of the init-param element.

Table D–3 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 run time with a 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>, <default-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 D–4 describes the subelements of the `init-params` element.

Table D–4 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.

pof-config

root element

Description

The `pof-config` element is the root element of the POF user type configuration descriptor.

Elements

Table D–5 describes the subelements of the `pof-config` element.

Table D–5 pof-config Subelements

Element	Required/ Optional	Description
`<user-type-list>`	Required	The `user-type-list` element contains zero or more user-type elements. Each POF user type that is 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.
<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`. The 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` is permitted at run time and automatically mapped to the specified super-class for purposes of obtaining a serializer. Valid values are `true` or `false`. The default value is `false`.
<enable-references>	Optional	The `enable-references` element indicates whether or not Identity/Reference type support is enabled. Valid values are `true` or `false`. Default value is `false`.
`<default-serializer>`	Optional	The `default-serializer` specifies what serializer to use to serialize and deserialize all user types defined in the POF configuration file. If a user-type defines a specific serializer, then that serializer is used instead of the default serializer.

serializer

Used in: <user-type>

Description

The serializer element specifies what POF serializer to use to serialize and deserialize a specific user type. A PofSerializer implementation is used to serialize and deserialize user type values to and from a POF stream.

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 POF serializer. If POF annotations are used, then the PofAnnotationSerializer implementation is used as the POF serializer.

If the init-params element is omitted, then the following four constructors are attempted (in this order) on the specific PofSerializer implementation:

- (int nTypeId, Class clz, ClassLoader loader)
- (int nTypeId, Class clz)
- (int nTypeId)
- ()

Elements

Table D–6 describes the subelements of the serializer element.

Table D–6 serializer Subelements

Element	Required/ Optional	Description
<class-name>	Required	Specifies the fully qualified name of the PofSerializer implementation.
<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.

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 D–7 describes the subelements of the user-type element.

Table D–7 user-type Subelements

Element	Required/ Optional	Description
<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 are 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. The first 1000 IDs are reserved for Coherence internal use and cannot be used.
<class-name>	Required	The class-name element specifies the fully qualified name of a Java class or interface that all values of the user type are type-assignable to.
<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) ▪ ()

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 is 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

Table D–8 describes the subelements of the user-type-list element.

Table D–8 user-type-list Subelements

Element	Required/ Optional	Description
<user-type>	Optional	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. Any number of <user-type> elements may be specified. 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.
<include>	Optional	The include element specifies the location of a POF cofiguration 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 file. Any number of <include> elements may be specified.

E

The PIF-POF Binary Format

The Portable Object Format (POF) allows object values to be encoded into a binary stream in such a way that the platform/language origin of the object value is both irrelevant and unknown. The Portable Invocation Format (PIF) allows method invocations to be similarly encoded into a binary stream. These two formats (referred as PIF-POF) are derived from a common binary encoding substrate. The binary format is provided here for informative purposes and is not a requirement for using PIF-POF. See Chapter 19, "Using Portable Object Format," for more information on using PIF-POF.

The following sections are included in this appendix:

Stream Format

The PIF-POF stream format is octet-based; a PIF-POF stream is a sequence of octet values. For the sake of clarity, this documentation treats all octets as unsigned 8-bit integer values in the range 0x00 to 0xFF (decimal 0 to 255). Byte-ordering is explicitly not a concern since (in PIF-POF) a given octet value that is represented by an unsigned 8-bit integer value is always written and read as the same unsigned 8-bit integer value.

A PIF stream contains exactly one Invocation. An Invocation consists of an initial POF stream that contains an Integer Value for the remaining length of the Invocation, immediately followed by a POF stream that contains an Integer Value that is the conversation identifier, immediately followed by a POF stream that contains a User Type value that is the message object. The remaining length indicates the total number of octets used to encode the conversation identifier and the message object; the remaining length is provided so that a process receiving an Invocation can determine when the Invocation has been fully received. The conversation identifier is used to support multiple logical clients and services multiplexed through a single connection, just as TCP/IP provides multiple logical port numbers for a given IP address. The message object is defined by the particular high-level conversational protocol.

A POF stream contains exactly one Value. The Value contains a Type Identifier, and if the Type Identifier does not imply a value, then it is immediately trailed by a data structure whose format is defined by the Type Identifier.

Integer Values

The stream format relies extensively on the ability to encode integer values in a compact form. Coherence refers to this integer binary format as a *packed integer*. This format uses an initial octet and one or more trailing octets as necessary; it is a variable-length format.

Table E–1 describes the three regions in the first octet.

Table E–1 Regions in the First Octet of a Packed Integer

Region Mask	Description
0x80	Continuation indicator
0x40	Negative indicator
0x3F	integer value (6 binary LSDs)

Table E–2 describes the two regions in the trailing octets.

Table E–2 Regions in the Trailing Octet of a Packed Integer

Region Mask	Description
0x80	Continuation indicator
0x7F	integer value (next 7 binary LSDs)

Example E–1 illustrates writing a 32-bit integer value to an octet stream as supported in Coherence.

Example E–1 Writing a 32-bit Integer Value to an Octet Stream

```
public static void writeInt(DataOutput out, int n)
        throws IOException
    {
    int b = 0;
    if (n < 0)
        {
        b = 0x40;
        n = ~n;
        }
    b |= (byte) (n & 0x3F);
    n >>>= 6;
    while (n != 0)
        {
        b |= 0x80;
        out.writeByte(b);
        b = (n & 0x7F);
        n >>>= 7;
        }
    out.writeByte(b);
    }
```

Example E–2 illustrates reading a 32-bit integer value from an octet stream as supported in Coherence.

Example E–2 Reading a 32-bit Integer Value from an Octet Stream

```
public static int readInt(DataInput in)
        throws IOException
```

```
{
int b = in.readUnsignedByte();
int n = b & 0x3F;
int cBits = 6;
boolean fNeg = (b & 0x40) != 0;
while ((b & 0x80) != 0)
    {
    b = in.readUnsignedByte();
    n |= ((b & 0x7F) << cBits);
    cBits += 7;
    }
if (fNeg)
    {
    n = ~n;
    }
return n;
}
```

Integer values used within this documentation without an explicit Type Identifier are assumed to be 32-bit signed integer values that have a decimal range of -2^{31} to $2^{31}-1$.

Table E–3 illustrates some integer value examples.

Table E–3 Binary Formats for Integer Values Without a Type Identifier

Value	Binary Format
0	0x00
1	0x01
2	0x02
99	0xA301
9999	0x8F9C01
-1	0x40
-2	0x41
-99	0xE201
-9999	0xCE9C01

Type Identifiers

A Type Identifier is encoded in the binary stream as an Integer Value. Type Identifiers greater than or equal to zero are user Type Identifiers. Type Identifiers less than zero are predefined ("intrinsic") type identifiers.

Table E–4 lists the predefined identifiers.

Table E–4 Predefined Type Identifiers

Type ID	Description
-1 (0x40)	int16
-2 (0x41)	int32
-3 (0x42)	int64
-4 (0x43)	int128*
-5 (0x44)	float32
-6 (0x45)	float64

Table E–4 (Cont.) Predefined Type Identifiers

Type ID	Description
-7 (0x46)	float128*
-8 (0x47)	decimal32*
-9 (0x48)	decimal64*
-10 (0x49)	decimal128*
-11 (0x4A)	boolean
-12 (0x4B)	octet
-13 (0x4C)	octet-string
-14 (0x4D)	char
-15 (0x4E)	char-string
-16 (0x4F)	date
-17 (0x50)	year-month-interval*
-18 (0x51)	time
-19 (0x52)	time-interval*
-20 (0x53)	datetime
-21 (0x54)	day-time-interval*
-22 (0x55)	collection
-23 (0x56)	uniform-collection
-24 (0x57)	array
-25 (0x58)	uniform-array
-26 (0x59)	sparse-array
-27 (0x5A)	uniform-sparse-array
-28 (0x5B)	map
-29 (0x5C)	uniform-keys-map
-30 (0x5D)	uniform-map
-31 (0x5E)	identity
-32 (0x5F)	reference

Type Identifiers less than or equal to -33 are a combination of a type and a value. This form is used to reduce space for these commonly used values.

Table E–5 lists the type identifiers that combine type and value.

Table E–5 Type Identifiers that Combine a Type and a Value

Type ID	Description
-33 (0x60)	boolean:false
-34 (0x61)	boolean:true
-35 (0x62)	string:zero-length
-36 (0x63)	collection:empty
-37 (0x64)	reference:null

Table E–5 (Cont.) Type Identifiers that Combine a Type and a Value

Type ID	Description
-38 (0x65)	floating-point:+infinity
-39 (0x66)	floating-point:-infinity
-40 (0x67)	floating-point:NaN
-41 (0x68)	int:-1
-42 (0x69)	int:0
-43 (0x6A)	int:1
-44 (0x6B)	int:2
-45 (0x6C)	int:3
-46 (0x6D)	int:4
-47 (0x6E)	int:5
-48 (0x6F)	int:6
-49 (0x70)	int:7
-50 (0x71)	int:8
-51 (0x72)	int:9
-52 (0x73)	int:10
-53 (0x74)	int:11
-54 (0x75)	int:12
-55 (0x76)	int:13
-56 (0x77)	int:14
-57 (0x78)	int:15
-58 (0x79)	int:16
-59 (0x7A)	int:17
-60 (0x7B)	int:18
-61 (0x7C)	int:19
-62 (0x7D)	int:20
-63 (0x7E)	int:21
-64 (0x7F)	int:22

Binary Formats for Predefined Types

This section describes the binary formats for the predefined ("intrinsic") type identifiers that are supported with PIF-POF. The types are: int, Decimal, Floating Point, Boolean, Octet, Octet String, Char, Char String, Date, Year-Month Interval, Time, Time Interval, Date-Time, Date-Time Interval, Collections, Arrays, Sparse Arrays, Key-Value Maps (Dictionaries), Identity, and Reference.

Int

Four signed integer types are supported: `int16`, `int32`, `int64`, and `int128`. If a type identifier for a integer type is encountered in the stream, it is immediately followed by an Integer Value.

The four signed integer types vary only by the length that is required to support the largest value of the type using the common "twos complement" binary format. The Type Identifier, one of `int16`, `int32`, `int64`, or `int128` is followed by an Integer Value in the stream. If the Integer Value is outside of the range supported by the type (-2^{15} to $2^{15}-1$ for `int16`, -2^{31} to $2^{31}-1$, for `int32`, -2^{63} to $2^{63}-1$ for `int64`, or -2^{127} to $2^{127}-1$ for `int128`,) then the result is undefined and may be bitwise truncation or an exception.

Additionally, there are some Type Identifiers that combine the `int` designation with a value into a single byte for purpose of compactness. As a result, these Type Identifiers are not followed by an Integer Value in the stream, since the value is included in the Type Identifier.

Table E–6 illustrates these type identifiers.

Table E–6 Type Identifiers that Combine an int Data Type with a Value

Value	int16	int32	int64	int128
0	0x69	0x69	0x69	0x69
1	0x6A	0x6A	0x6A	0x6A
2	0x6B	0x6B	0x6B	0x6B
99	0x40A301	0x41A301	0x42A301	0x43A301
9999	0x408F9C01	0x418F9C01	0x428F9C01	0x438F9C01
-1	0x68	0x68	0x68	0x68
-2	0x4041	0x4141	0x4241	0x4341
-99	0x40E201	0x41E201	0x42E201	0x43E201
-9999	0x40CE9C01	0x41CE9C01	0x42CE9C01	0x43CE9C01

The Java type equivalents are `short` (`int16`), `int` (`int32`), `long` (`int64`) and `BigInteger` (`int128`). Since `BigInteger` can represent much larger values, it is not possible to encode all `BigInteger` values in the `int128` form; values out of the `int128` range are basically unsupported, and would result in an exception or would use a different encoding, such as a string encoding.

Coercion of Integer Types

To enable the efficient representation of numeric data types, an integer type is coerced into any of the following types by a stream recipient:

Table E–7 Type IDs of Integer Types that can be Coerced into Other Types

Type ID	Description
-1 (0x40)	int16
-2 (0x41)	int32
-3 (0x42)	int64
-4 (0x43)	int128
-5 (0x44)	float32
-6 (0x45)	float64
-7 (0x46)	float128
-8 (0x47)	decimal32

Table E–7 (Cont.) Type IDs of Integer Types that can be Coerced into Other Types

Type ID	Description
-9 (0x48)	decimal64
-10 (0x49)	decimal128
-12 (0x4B)	octet
-14 (0x4D)	char

In other words, if the recipient reads any of the above types from the stream and it encounters an encoded integer value, it automatically converts that value into the expected type. This capability allows a set of common (that is, small-magnitude) octet, character, integer, decimal and floating-point values to be encoded using the single-octet integer form (Type Identifiers in the range -41 to -64).

For purposes of unsigned types, the integer value -1 is translated to 0xFF for the `octet` type, and to 0xFFFF for the `char` type. (In the case of the `char` type, this does unfortunately seem to imply a UTF-16 platform encoding; however, it does not violate any of the explicit requirements of the stream format.)

Decimal

There are three floating-point decimal types supported: `decimal32`, `decimal64`, and `decimal128`. If a type identifier for a decimal type is encountered in the stream, it is immediately followed by two packed integer values. The first integer value is the unscaled value, and the second is the scale. These values are equivalent to the parameters to the constructor of Java's `BigDecimal` class: `java.math.BigDecimal(BigInteger unscaledVal, int scale)`.

In addition to the coercion of integer values into decimal values supported as described in "Coercion of Integer Types" on page E-6, the constant type+value identifiers listed in Table E–8 are used to indicate special values supported by IEEE 754r.

Table E–8 Type Identifiers that can Indicate Decimal Values

Type ID	Description
-38 (0x65)	floating-point:+infinity
-39 (0x66)	floating-point:-infinity
-40 (0x67)	floating-point:NaN

Java does not provide a standard (that is, portable) decimal type; rather, it has the awkward `BigDecimal` implementation that was intended originally for internal use in Java's cryptographic infrastructure. In Java, the decimal values for positive and negative infinity, and not-a-number (`NaN`), are not supported.

Floating Point

Three base-2 floating point types are supported: `float32`, `float64`, and `float128`. If a type identifier for a floating point type is encountered in the stream, it is immediately followed by a fixed-length floating point value, whose binary form is defined by IEEE 754/IEEE754r. IEEE 754 format is used to write floating point numbers to the stream, and IEEE 754r format is used for the `float128` type.

In addition to the coercion of integer values into decimal values as described in "Coercion of Integer Types" on page E-6, the constants in Table E–9 are used to indicate special values supported by IEEE-754

Table E–9 Type Identifiers that can Indicate IEEE 754 Special Values

Type ID	Description
-38 (0x65)	floating-point:+infinity
-39 (0x66)	floating-point:-infinity
-40 (0x67)	floating-point:NaN

Other special values defined by IEEE-754 are encoded using the full 32-bit, 64-bit or 128-bit format, and may not be supported on all platforms. Specifically, by not providing any means to differentiate among them, Java only supports one `NaN` value.

Boolean

If the type identifier for Boolean occurs in the stream, it is followed by an integer value, which represents the Boolean value `false` for the integer value of zero, or `true` for all other integer values.

While it is possible to encode Boolean values as described in "Coercion of Integer Types" on page E-6, the only values for the Boolean type are `true` and `false`. As such, the only expected binary formats for Boolean values are the predefined (and compact) forms described in Table E–10.

Table E–10 Type Identifiers that can Indicate Boolean Values

Type ID	Description
-33 (0x60)	boolean:false
-34 (0x61)	boolean:true

Octet

If the type identifier for Octet occurs in the stream, it is followed by the octet value itself, which is by definition in the range 0 to 255 (0x00 to 0xFF). As described in "Coercion of Integer Types" on page E-6, the compact form of integer values can be used for Octet values, with the integer value -1 being translated as 0xFF.

Table E–11 lists the integer values that may be used as Octet values.

Table E–11 Integer Values that may be Used for Octet Values

Value	Octet
0 (0x00)	0x69
1 (0x01)	0x6A
2 (0x02)	0x6B
99 (0x63)	0x4B63
254 (0xFE)	0x4BFE
255 (0xFF)	0x68

Octet String

If the type identifier for Octet String occurs in the stream, it is followed by an Integer Value for the length *n* of the string, and then *n* octet values.

An Octet String of zero length is encoded using the "string:zero-length" Type Identifier.

Char

If the type identifier for Char occurs in the stream, it is followed by a UTF-8 encoded character. As described in the section on "Coercion of Integer Types" on page E-6, the compact form of integer values may be used for Char values, with the integer value -1 being translated as 0xFFFF.

Example E–3 illustrates writing a character value to an octet stream.

Example E–3 Writing a Character Value to an Octet Stream

```
public static void writeChar(DataOutput out, int ch)
        throws IOException
    {
    if (ch >= 0x0001 && ch <= 0x007F)
        {
        // 1-byte format: 0xxx xxxx
        out.write((byte) ch);
        }
    else if (ch <= 0x07FF)
        {
        // 2-byte format: 110x xxxx, 10xx xxxx
        out.write((byte) (0xC0 | ((ch >>> 6) & 0x1F)));
        out.write((byte) (0x80 | ((ch ) & 0x3F)));
        }
    else
        {
        // 3-byte format: 1110 xxxx, 10xx xxxx, 10xx xxxx
        out.write((byte) (0xE0 | ((ch >>> 12) & 0x0F)));
        out.write((byte) (0x80 | ((ch >>> 6) & 0x3F)));
        out.write((byte) (0x80 | ((ch ) & 0x3F)));
        }
    }
```

Example E–4 illustrates reading a character value from an octet stream.

Example E–4 Reading a Character Value from an Octet Stream

```
public static char readChar(DataInput in)
        throws IOException
    {
    char ch;

    int b = in.readUnsignedByte();
    switch ((b & 0xF0) >>> 4)
        {
        case 0x0: case 0x1: case 0x2: case 0x3:
        case 0x4: case 0x5: case 0x6: case 0x7:
            // 1-byte format: 0xxx xxxx
            ch = (char) b;
            break;

        case 0xC: case 0xD:
```

```
            {
            // 2-byte format: 110x xxxx, 10xx xxxx
            int b2 = in.readUnsignedByte();
            if ((b2 & 0xC0) != 0x80)
                {
                throw new UTFDataFormatException();
                }
            ch = (char) (((b & 0x1F) << 6) | b2 & 0x3F);
            break;
            }

        case 0xE:
            {
            // 3-byte format: 1110 xxxx, 10xx xxxx, 10xx xxxx
            int n = in.readUnsignedShort();
            int b2 = n >>> 8;
            int b3 = n & 0xFF;
            if ((b2 & 0xC0) != 0x80 || (b3 & 0xC0) != 0x80)
                {
                throw new UTFDataFormatException();
                }
            ch = (char) (((b & 0x0F) << 12) |
                        ((b2 & 0x3F) << 6) |
                          b3 & 0x3F);
            break;
            }

        default:
            throw new UTFDataFormatException(
                    "illegal leading UTF byte: " + b);
        }

    return ch;
    }
```

Char String

If the type identifier for Char String occurs in the stream, it is followed by an Integer Value for the length *n* of the UTF-8 representation string **in octets**, and then *n* octet values composing the UTF-8 encoding described above. Note that the format length-encodes the octet length, not the character length.

A Char String of zero length is encoded using the `string:zero-length` Type Identifier. Table E–12 illustrates the Char String formats.

Table E–12 Values for Char String Formats

Values	Char String Format
	0x62 (or 0x4E00)
"ok"	0x4E026F6B

Date

Date values are passed using ISO8601 semantics. If the type identifier for Date occurs in the stream, it is followed by three Integer Values for the year, month and day, in the ranges as defined by ISO8601.

Year-Month Interval

If the type identifier for Year-Month Interval occurs in the stream, it is followed by two Integer Values for the number of years and the number of months in the interval.

Time

Time values are passed using ISO8601 semantics. If the type identifier for Time occurs in the stream, it is followed by five Integer Values, which may be followed by two more Integer Values. The first four Integer Values are the hour, minute, second and fractional second values. Fractional seconds are encoded in one of three ways:

- 0 indicates no fractional seconds.
- [1..999] indicates the number of milliseconds.
- [-1..-999999999] indicates the negated number of nanoseconds.

The fifth Integer Value is a time zone indicator, encoded in one of three ways:

- 0 indicates no time zone.
- 1 indicates Universal Coordinated Time (UTC).
- 2 indicates a time zone offset, which is followed by two more Integer Values for the hour offset and minute offset, as described by ISO8601.

The encoding for variable fractional and time zone does add complexity to the parsing of a Time Value, but provide for much more complete support of the ISO8601 standard and the variability in the precision of clocks, while achieving a high degree of binary compactness. While time values tend to have no fractional encoding or millisecond encoding, the trend over time is toward higher time resolution.

Time Interval

If the type identifier for Time Interval occurs in the stream, it is followed by four Integer Values for the number of hours, minutes, seconds and nanoseconds in the interval.

Date-Time

Date-Time values are passed using ISO8601 semantics. If the type identifier for Date-Time occurs in the stream, it is followed by eight or ten Integer Values, which correspond to the Integer Values that compose the Date and Time values.

Coercion of Date and Time Types

Date Value can be coerced into a Date-Time Value. Time Value can be coerced into a Date-Time Value. Date-Time Value can be coerced into either a Date Value or a Time Value.

Day-Time Interval

If the type identifier for Day-Time Interval occurs in the stream, it is followed by five Integer Values for the number of days, hours, minutes, seconds and nanoseconds in the interval.

Collections

A collection of values, such as a bag, a set, or a list, are encoded in a POF stream using the Collection type. Immediately following the Type Identifier, the stream contains the

Collection Size, an Integer Value indicating the number of values in the Collection, which is greater than or equal to zero. Following the Collection Size, is the first value in the Collection (if any), which is itself encoded as a Value. The values in the Collection are contiguous, and there is exactly *n* values in the stream, where *n* equals the Collection Size.

If all the values in the Collection have the same type, then the Uniform Collection format is used. Immediately following the Type Identifier (uniform-collection), the uniform type of the values in the collection writes to the stream, followed by the Collection Size *n* as an Integer Value, followed by *n* values **without their Type Identifiers**. Note that values in a Uniform Collection cannot be assigned an identity, and that (as a side-effect of the explicit type encoding) an empty Uniform Collection has an explicit content type.

Table E–13 illustrates examples of Collection and Uniform Collection formats for several values.

Table E–13 Collection and Uniform Collection Formats for Various Values

Values	Collection Format	Uniform Collection Format
	0x63 (or 0x5500)	n/a
1	0x55016A	0x56410101
1,2,3	0x55036A6B6C	0x564103010203
1, "ok"	0x55026A4E026F6B	n/a

Arrays

An indexed array of values is encoded in a POF stream using the Array type. Immediately following the Type Identifier, the stream contains the Array Size, an Integer Value indicating the number of elements in the Array, which must be greater than or equal to zero. Following the Array Size is the value of the first element of the Array (the zero index) if there is at least one element in the array which is itself encoded using as a Value. The values of the elements of the Array are contiguous, and there must be exactly *n* values in the stream, where *n* equals the Array Size.

If all the values of the elements of the Array have the same type, then the Uniform Array format is used. Immediately following the Type Identifier (uniform-array), the uniform type of the values of the elements of the Array writes the stream, followed by the Array Size *n* as an Integer Value, followed by *n* values **without their Type Identifiers**. Note that values in a Uniform Array cannot be assigned an identity, and that (as a side-effect of the explicit type encoding) an empty Uniform Array has an explicit array element type.

Table E–14 illustrates examples of Array and Uniform Array formats for several values.

Table E–14 Array and Uniform Array Formats for Various Values

Values	Array Format	Uniform Array Format
	0x63 (or 0x5700)	0x63 (or 0x584100) – This example assumes an element type of Int32.
1	0x57016A	0x58410101
1,2,3	0x57036A6B6C	0x584103010203
1, "ok"	0x57026A4E026F6B	n/a

Sparse Arrays

For arrays whose element values are sparse, the Sparse Array format allows indexes to be explicitly encoded, implying that any missing indexes have a default value. The default value is false for the Boolean type, zero for all numeric, octet and char types, and null for all reference types. The format for the Sparse Array is the Type Identifier (sparse-array), followed by the Array Size *n* as an Integer Value, followed by not more than *n* index/value pairs, each of which is composed of an array index encoded as an Integer Value *i* (0 <= *i* < *n*) whose value is greater than the previous element's array index, and an element value encoded as a Value; the Sparse Array is finally terminated with an illegal index of -1.

If all the values of the elements of the Sparse Array have the same type, then the Uniform Sparse Array format is used. Immediately following the Type Identifier (uniform-sparse-array), the uniform type of the values of the elements of the Sparse Array writes the stream, followed by the Array Size *n* as an Integer Value, followed by not more the *n* index/value pairs, each of which is composed of an array index encoded as an Integer Value *i* (0 <= *i* < *n*) whose value is greater than the previous element's array index, and a element value encoded as a Value **without a Type Identifier**; the Uniform Sparse Array is finally terminated with an illegal index of -1. Note that values in a Uniform Sparse Array cannot be assigned an identity, and that (as a side-effect of the explicit type encoding) an empty Uniform Sparse Array has an explicit array element type.

Table E–15 illustrates examples of Sparse Array and Uniform Sparse Array formats for several values.

Table E–15 Sparse Array and Uniform Sparse Array Formats for Various Values

Values	Sparse Array format	Uniform Sparse Array format
	0x63 (or 0x590040)	0x63 (or 0x5A410040) – This example assumes an element type of Int32.
1	0x5901006A40	0x5A4101000140
1,2,3	0x5903006A016B026C40	0x5A410300010102020340
1,,,,5,,,,9	0x5909006A046E087240	0x5A410900010405080940
1,,,,"ok"	0x5905006A044E026F6B40	n/a

Key-Value Maps (Dictionaries)

For key/value pairs, a Key-Value Map (also known as Dictionary data structure) format is used. There are three forms of the Key-Value Map binary encoding:

- The generic `map` encoding is a sequence of keys and values;
- The `uniform-keys-map` encoding is a sequence of keys of a uniform type and their corresponding values;
- The `uniform-map` encoding is a sequence of keys of a uniform type and their corresponding values of a uniform type.

The format for the Key-Value Map is the Type Identifier (map), followed by the Key-Value Map Size *n* as an Integer Value, followed by *n* key/value pairs, each of which is composed of a key encoded as Value, and a corresponding value encoded as a Value.

Table E–16 illustrates several examples of key/value pairs and their corresponding binary format.

Table E–16 Binary Formats for Key/Value Pairs

Values	Binary format
	0x63 (or 0x5B00)
1="ok"	0x5B016A4E026F6B
1="ok", 2="no"	0x5B026A4E026F6B6B4E026E6F

If all of the keys of the Key-Value Map are of a uniform type, then the encoding uses a more compact format, starting with the Type Identifier (uniform-keys-map), followed by the Type Identifier for the uniform type of the keys of the Key-Value Map, followed by the Key-Value Map Size *n* as an Integer Value, followed by *n* key/value pairs, each of which is composed of a key encoded as a Value **without a Type Identifier**, and a corresponding value encoded as a Value.

Table E–17 illustrates several examples of the binary formats for Key/Value pairs where the Keys are of uniform type.

Table E–17 Binary Formats for Key/Value Pairs where Keys are of Uniform Type

Values	Binary format
	0x63 (or 0x5C4100)
1="ok"	0x5C4101014E026F6B
1="ok", 2="no"	0x5C4102014E026F6B024E026E6F

If all of the keys of the Key-Value Map are of a uniform type, and all the corresponding values of the map are also of a uniform type, then the encoding uses a more compact format, starting with the Type Identifier (uniform-map), followed by the Type Identifier for the uniform type of the keys of the Key-Value Map, followed by the Type Identifier for the uniform type of the values of the Key-Value Map, followed by the Key-Value Map Size *n* as an Integer Value, followed by *n* key/value pairs, each of which is composed of a key encoded as a Value **without a Type Identifier**, and a corresponding value encoded as a Value **without a Type Identifier**.

Table E–18 illustrates several examples of the binary formats for Key/Value pairs where the Keys and Values are of uniform type.

Table E–18 Binary Formats for Key/Value Pairs where Keys and Values are of Uniform Type

Values	Binary format
	0x63 (or 0x5D414E00)
1="ok"	0x5D414E0101026F6B
1="ok", 2="no"	0x5D414E0201026F6B02026E6F

Identity

If the type identifier for Identity occurs in the stream, it is followed by an Integer Value, which is the Identity. Following the Identity is the value that is being identified, which is itself encoded as a Value.

Any value within a POF stream that occurs multiple times, is labeled with an Identity, and subsequent instances of that value within the same POF stream are replaced with a Reference. For platforms that support "by reference" semantics, the identity represents a serialized form of the actual object identity.

An Identity is an Integer Value that is greater than or equal to zero. A value within the POF stream has at most one Identity. Values within a uniform data structure can be assigned an identity.

Reference

A Reference is a pointer to an Identity that has been encountered inside the current POF stream, or a null pointer.

For platforms that support "by reference" semantics, the reference in the POF stream becomes a reference in the realized (deserialized) object, and a null reference in the POF stream becomes a null reference in the realized object. For platforms that do not support "by reference" semantics, and for cases in which a null reference is encountered in the POF stream for a non-reference value (for example, a primitive property in Java), the default value for the type of value is used.

Table E–19 illustrates examples of binary formats for several "by reference" semantics.

Table E–19 Binary Formats for "By Reference" Semantics

Value	Binary Format
Id #1	0x5F01
Id #350	0x5F9E05
null	0x60

Support for forward and outer references is not required by POF. In POF, both the identity that is referenced and the value that is being referenced by the identity have occurred within the POF stream. In the first case, a reference is not made to an identity that has not yet been encountered, and in the second case, a reference is not made within a complex value (such as a collection or a user type) to that complex value itself.

Binary Format for User Types

All non-intrinsic types are referred to as User Types. User Types are composed of zero or more indexed values (also known as fields, properties, and attributes), each of which has a Type Identifier. Furthermore, User Types are versioned, supporting both forward and backward compatibility.

User Types have a Type Identifier with a value greater than or equal to zero. The Type Identifier has no explicit or self-describing meaning within the stream itself; in other words, a Value does not contain a type (or "class") definition. Instead, the encoder (the sender) and the decoder (the receiver) share an implicit understanding, called a *Context*, which includes the necessary metadata, including the user type definitions.

The binary format for a User Type is very similar to that of a Sparse Array; conceptually, a User Type can be considered a Sparse Array of property values. The format for User Types is the Type Identifier (an Integer Value greater than or equal to zero), followed by the Version Identifier (an Integer Value greater than or equal to zero), followed by index/value pairs, each of which is composed of a Property Index encoded as an Integer Value *i* (0 <= *i*) whose value is greater than the previous Property Index, and a Property Value encoded as a Value; the User Type is finally terminated with an illegal Property Index of -1.

Like the Sparse Array, any property that is not included as part of the User Type encoding is assumed to have a default value. The default value is false for the Boolean type, zero for all numeric, octet and char types, and null for all reference types.

Versioning of User Types

Versioning of User Types supports the addition of properties to a User Type, but not the replacement or removal of properties that existed in previous versions of the User Type. By including the versioning capability as part of the general binary contract, it is possible to support both backward and forward compatibility.

When a sender sends a User Type value of a version *v1* to a receiver that supports version *v2* of the same User Type, the receiver uses default values for the additional properties of the User Type that exist in *v2* but do not exist in *v1*.

When a sender sends a User Type value of a version *v2* to a receiver that only supports version *v1* of the same User Type, the receiver treats the additional properties of the User Type that exist in *v2* but do not exist in *v1* as opaque. If the receiver must store the value (persistently), or if the possibility exists that the value is ever sent at a later point, then the receiver stores those additional opaque properties for later encoding. Sufficient type information is included to allow the receiver to store off the opaque property values in either a typed or binary form; when the receiver re-encodes the User Type, it must do so using the Version Indicator *v2*, since it is including the unaltered *v2* properties.