Oracle® Fusion Middleware

Developing Applications with Oracle Coherence

14c (14.1.2.0.0)
F79650-10
June 2025

Oracle Fusion Middleware Developing Applications with Oracle Coherence, 14c (14.1.2.0.0)

F79650-10

Primary Author: Oracle Corporation

Contents

2 Building Your First Coherence Application

3 Understanding Configuration

4 Extending Configuration Files

5 Debugging in Coherence

8 Starting and Stopping Coherence

9 Dynamically Managing Cluster Membership

10 Tuning TCMP Behavior

11 Using the Service Guardian

14 Implementing Storage and Backing Maps

15 Caching Data Sources

16 Serialization Paged Cache

17 Using Quorum

18 Cache Configurations by Example

19 Using Cache Configuration Override

24 Introduction to Coherence Programming

25 Performing Basic Cache Operations

26 Performing Basic Topic Publish and Subscribe Operations

27 Using Portable Object Format

28 Querying Data in a Cache

31 Using Map Events

32 Controlling Map Operations with Triggers

33 Using Live Events

34 Using Coherence Query Language

35 Performing Transactions

36 Working with Partitions

37 Managing Thread Execution

38 Constraints on Re-Entrant Calls

39 Using Timeout with Cache Operations

40 Using the Repository API

41 Implementing Concurrency in a Distributed Environment

42 Using Contexts and Dependency Injection

45 Performing Basic Coherence JCache Tasks

46 Using JCache Events

49 Upgrade Considerations

A Operational Configuration Elements

B Cache Configuration Elements

C Topic Configuration Element Reference

D POF User Type Configuration Elements

E System Property Overrides

F The PIF-POF Binary Format

Preface

Developing Applications with Oracle Coherence provides contextual information, instructions, and examples that are designed to teach developers and architects how to use Coherence and develop Coherence-based applications.

This preface includes the following sections:

Audience

Developing Applications with Oracle Coherence 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. For details about installing Coherence, see *Installing Oracle Coherence*. 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 `https://www.oracle.com/corporate/accessibility/`.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `https://support.oracle.com/portal/` or visit `Oracle Accessibility Learning and Support` if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to

build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documents

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

- *Installing Oracle Coherence*
- *Release Notes for Oracle Coherence*
- *Developing Oracle Coherence Applications for Oracle WebLogic Server*
- *Administering Oracle Coherence*
- *Managing Oracle Coherence*
- *Developing Remote Clients for Oracle Coherence*
- *Integrating Oracle Coherence*
- *Securing Oracle Coherence*
- *Administering HTTP Session Management with Oracle Coherence*Web*
- *Java API Reference for Oracle Coherence*
- *C++ API Reference for Oracle Coherence*
- *.NET API Reference for Oracle Coherence*
- *REST API for Managing 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

Learn about Coherence development, configuration, and debugging. Try creating a simple Coherence application.

Part I contains the following chapters:

- Introduction to Coherence
- Building Your First Coherence Application
  You can follow step-by-step instructions for building and running a basic Coherence example that demonstrates a few fundamental Coherence concepts.
- Understanding Configuration
- Extending Configuration Files
- Debugging in Coherence

1

Introduction to Coherence

Prior to developing Coherence applications, take some time to learn about important Coherence concepts and features.
This chapter includes the following sections:

- Basic Concepts
  Learn about Coherence clustering, configuration, caching, data storage, and serialization.
- Read/Write Caching
- Querying the Cache
- Invocation Service
- Event Programming
- Transactions
  Coherence includes various transaction options that provide different transaction guarantees.
- HTTP Session Management
- Object-Relational Mapping Integration
- C++/.NET Integration
- Management and Monitoring
- Using Java Modules to Build a Coherence Application
  From Coherence 14c (14.1.1.0.0), most of the Coherence jars are Java modules (with the `module-info.java` file in each jar). To see the module name, characteristics, and dependencies of each jar, use the `--describe-module` operation modifier of the jar command.

Basic Concepts

Learn about Coherence clustering, configuration, caching, data storage, and serialization.

This section includes the following topics:

- Clustered Data Management
- A single API for the logical layer, XML configuration for the physical layer
- Caching Strategies
- Data Storage Options
- Serialization Options
- Configurability and Extensibility
- Namespace Hierarchy

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. See Understanding Local 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. See Understanding Distributed Caches.
- 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. See Understanding Near Caches.
- View Cache – Perfect for small, read-heavy caches. See Understanding View Caches.

In-process caching provides the highest level of raw performance, because objects are managed within the local JVM. This benefit is most directly realized by the Local, View, 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 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.
- **Journal**—A combination of RAM storage and disk storage, optimized for solid state disks, that uses a journaling technique. Journal-based storage requires serialization/deserialization.
- **File-based**—Uses a Berkeley Database JE storage system.

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 Implementing Storage and Backing Maps.

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 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). See Introduction to Coherence Clusters . Under the cluster service are the various services that comprise the Coherence API. These include the various caching services (Distributed, Federated, 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.

Read/Write Caching

The Coherence `NamedCache` API is the primary interface used by applications to get and interact with cache instances.
This section includes the following topics:

- NamedCache
- NamedCache Usage Patterns

NamedCache

The following source code returns a reference to a `NamedCache` instance. The underlying cache service is started if necessary.

```
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 `-Dcoherence.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.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 Querying Data in a Cache.
- `com.tangosol.util.InvocableMap`—methods for server-side processing of cache data. See Processing Data In a Cache.

- `com.tangosol.util.ObservableMap`—methods for listening to cache events. See Using Map Events.
- `com.tangosol.util.ConcurrentMap`—methods for concurrent access such as `lock()` and `unlock()`. See Performing Transactions.

When getting a reference to a `NamedCache`, a single thread must ensure that:

- a cluster is running
- a service is running
- a cache is created

Coherence was originally designed to perform these tasks using `synchronized()` blocks (intrinsic java locks) because multiple threads can perform the above simultaneously.

Because intrinsic locks are non-interruptible, this approach could lead to application threads blocking indefinitely, waiting for the lock when, for example, a network outage occurs.

Interruptible locks provide the ability to do a try-lock that times out. This allows an application to not be indefinitely blocked. With this improvement, you can now start a cluster or cache within a try-with-resources block using a thread local timeout. See Class Timeout.

For example:

```
try (Timeout t = Timeout.after(20, TimeUnit.SECONDS))
    {
    CacheFactory.getCache("dist"); // or CacheFactory.ensureCluster();
    }
catch (Exception e)
    {
    // doSomething else.
    }
```

The default lock timeout is `Long.MAX_VALUE`. If the operation (`getCache`, `ensureCluster`, and so on) does not return within the specified timeout, an `InterruptedException` is displayed. If the corresponding locks cannot be acquired within the specified timeout, a `RequestTimeoutException` is displayed. The service request timeout is also honored for the service lock, if specified. For example, when references to caches are maintained and member restarts occur, underlying resources must be obtained again on the next call performed on a `NamedCache` reference. In this case, a configured request timeout will prevent blocking indefinitely.

If both request timeout and thread local timeout are specified, the latter takes precedence.

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 Caching Data Sources .

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.

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` instance, 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. See Querying Data in a Cache.
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.

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 Processing Data In a Cache.
The invocation service is accessed through the `InvocationService` interface and includes the following two methods:

```
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 `CacheFactory` class.

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

Event Programming

Coherence supports two event programming models that allow applications to receive and react to notifications of cluster operations. Applications observe events as logical concepts regardless of which computer caused the event. Events provide a common way of extending Coherence with application-specific logic.
The event programming models are:

- Live Events – The live event programming model uses user-defined event interceptors that are registered to receive different types of events. Applications decide what action to take based on the event type. Many events that are available through the use of map events are also supported using live events. See Using Live Events.
- Map Events – The map event programming model uses user-defined map listeners that are attached to the underlying map implementation. Map events offer customizable server-based filters and lightweight events that can minimize network traffic and processing. Map listeners follow the JavaBean paradigm and can distinguish between system cache events (for example, eviction) and application cache events (for example, get/put operations). See Using Map Events.

Transactions

Coherence includes various transaction options that provide different transaction guarantees.

Coherence transaction options include: basic data concurrency using the `ConcurrentMap` interface and `EntryProcessor` API, partition-level transactions using implicit locking and the `EntryProcessor` API, atomic transactions using the Transaction Framework API, and atomic transactions with full XA support using the Coherence resource adapter. See Performing Transactions.

HTTP Session Management

Coherence*Web is an HTTP session-management module with support for a wide range of application servers. See Introduction to Coherence*Web in *Administering HTTP Session Management with Oracle Coherence*Web*.
Using Coherence session management does not require any changes to the application. Coherence*Web uses near caching to provide fully fault-tolerant caching, with almost unlimited scalability (to several hundred cluster nodes without issue).

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 Integrating JPA Using the Coherence API in *Integrating 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 Overview of Coherence*Extend in *Developing Remote Clients for Oracle Coherence*.

Management and Monitoring

Coherence offers management and monitoring facilities using Java Management Extensions (JMX). See Introduction to Oracle Coherence Management in *Managing Oracle Coherence*.

Using Java Modules to Build a Coherence Application

From Coherence 14c (14.1.1.0.0), most of the Coherence jars are Java modules (with the `module-info.java` file in each jar). To see the module name, characteristics, and dependencies of each jar, use the `--describe-module` operation modifier of the jar command.

Example of the jar command:

```
jar --file=coherence.jar --describe-module
```

> **Note:**
>
> The following Coherence jars are not Java modules:
>
> - `coherence-http-grizzly.jar`
> - `coherence-http-jetty.jar`
> - `coherence-http-netty.jar`
> - `coherence-http-simple.jar`
> - `coherence-web.jar`

Each of the Coherence modules is open, which grants reflective access to all of its packages to other modules. However, Coherence may require you to open or export modules it depends on or explicitly add transitive dependencies.

For instance, when using lambdas in a distributed environment, as described in About Lambdas in a Distributed Environment, you need to open any application module(s) containing

distributed lambda(s) to module `com.oracle.coherence`. This allows for resolving the distributed lambda to the application's lambda(s) during deserialization.

In general, the `java.lang.IllegalAccessError` or `java.lang.IllegalAccessException` error message provides descriptive information about the module you require to open or add exports to.

2

Building Your First Coherence Application

You can follow step-by-step instructions for building and running a basic Coherence example that demonstrates a few fundamental Coherence concepts.

The sample application is a simple "Hello, World!" application and is implemented as a standalone Java application.

Depending on the deployment type, framework, or language that you use, see Using Different Deployment Models, Frameworks, or Languages with Coherence.

> **Note:**
>
> The example in this chapter is basic and is intended only to teach general concepts. For more advanced examples, refer to the examples available in the Coherence GitHub Repository. See Running the Coherence Examples in *Installing Oracle Coherence*.

This chapter includes the following sections:

- Task 1: Install Coherence
  If you haven't already done so, install Coherence.
- Task 2: Configure and Start the Example Cluster
  Caches are hosted on a Coherence cluster. At runtime, 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.
- Task 3: Create and Run a Basic Coherence Standalone Application
- Using Different Deployment Models, Frameworks, or Languages with Coherence
  The previously described "Hello, World!" example application is a simple demonstration on how to use Coherence and Java together.

Task 1: Install Coherence

If you haven't already done so, install Coherence.

If you want to install the commercial edition of Coherence, then see Installing Oracle Coherence for Java in *Installing Oracle Coherence*.

Make sure to replace `COHERENCE_HOME` with the full path of the Coherence directory of your installation.

Task 2: Configure and Start the Example Cluster

Caches are hosted on a Coherence cluster. At runtime, 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.

In the example, 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 examples use the default cache configuration which is sufficient for most applications.

This example uses system properties to modify the default configuration, but you can also modify it using the configuration file. Specifically, 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 cache configuration is sufficient for most applications.

To configure and start the example cluster:

> **Note:**
>
> Throughout this example, replace *COHERENCE_HOME* with the Coherence installation directory.

- Start a cache server instance.

  Open a command-line utility and enter:

  ```
  java -Dcoherence.wka=127.0.0.1 -jar COHERENCE_HOME/lib/coherence.jar
  ```

  Setting `java -Dcoherence.wka=127.0.0.1` specifies that the cluster is running only on localhost.

  By default, Coherence auto-generates the cluster name. If you want to specify the name of the cluster yourself, then add `-Dcoherence.cluster=myCluster` to all of the commands that include the server and client.

Task 3: Create and Run a Basic Coherence Standalone Application

Task 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 cluster 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. This section includes the following topics:

- Create the Sample Standalone Application
- Run the Sample Standalone Application
- Verify the Example Cache

Create the Sample Standalone Application

Applications use the Coherence API to access and interact with a cache. The `CoherenceSession` class creates a `Session` instance using a default session provider then gets a reference to a `NamedCache` instance using the `getCache` method. The `NamedCache` instance is then used to retrieve and store objects in the cache. The Hello World application is very basic, but it does demonstrate using the `Session` and `NamedCache` APIs.

Example 2-1 The Sample HelloWorld Standalone Application

```
import com.tangosol.net.CoherenceSession;
import com.tangosol.net.NamedCache;
import com.tangosol.net.Session;

public class HelloWorld {
   public static void main(String[] args) throws Exception {

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

      Session session = new CoherenceSession();
      NamedCache<String, String> cache = session.getCache("hello-example");

      cache.put(key, value);
      System.out.println(cache.get(key));
      session.close();

   }
}
```

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 by setting the `coherence.distributed.localstorage` property to `false`. Also, specify the system properties to ensure the client joins the correct cluster. For example:

   ```
   java -cp .:COHERENCE_HOME\lib\coherence.jar -
   Dcoherence.distributed.localstorage=false -Dcoherence.wka=127.0.0.1
   HelloWorld
   ```

   > **Note:**
   >
   > If you are on Windows, then replace the `:` separator with `;`
   >
   > .

The "Hello, World!" application starts. The CoherenceSession 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 session is closed 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/lib/coherence.jar -Dcoherence.wka=127.0.0.1  -
   Dcoherence.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!
   ```

Using Different Deployment Models, Frameworks, or Languages with Coherence

The previously described "Hello, World!" example application is a simple demonstration on how to use Coherence and Java together.

You can also use Coherence with other common frameworks, deployment types, and languages.

For general information on Coherence integration, see Oracle Coherence - Integrate.

Otherwise, review the following sections for additional integration possibilities.

Helidon

Project Helidon is a set of Java Libraries for writing microservices. Coherence provides out of the box integration with Helidon.

For information on getting started with Helidon MP and Coherence, see Quick Start at the *Coherence Community* website.

Spring

Oracle Coherence can be integrated with Spring, which is a platform for building and running Java-based enterprise applications.

For information on getting started with Spring and Coherence, see the Coherence Spring Integration project on GitHub.

Micronaut

Oracle Coherence can be integrated with Micronaut, which is an open source framework for building lightweight modular applications and microservices.

For information on getting started with Micronaut and Coherence, see the Micronaut Coherence project on GitHub.

Kubernetes

Oracle enables organizations using Coherence to move their clusters into the cloud. By supporting industry standards, such as Docker and Kubernetes, Oracle facilitates running Coherence on cloud-neutral infrastructure.

For information on getting started with Coherence and Kubernetes, see the Coherence Operator repository on GitHub.

WebLogic Server

For information on developing and deploying Coherence applications in WebLogic Server, see Introduction to Coherence Applications in *Developing Oracle Coherence Applications for Oracle WebLogic Server*.

.NET

The Coherence .NET client allows .NET based applications to act as cache clients using the Coherence*Extend protocol.

For information on getting started with the Coherence .NET client, see Oracle Coherence for .NET - Community Edition on GitHub.

C++

The Coherence C++ client allows C++ applications to act as cache clients using the Coherence*Extend protocol.

For information on getting started with the Coherence C++ client, see Oracle Coherence for C++ Community Edition on GitHub.

JavaScript

The Coherence JavaScript client allows Node applications to act as cache clients to a Coherence cluster using gRPC as the network transport.

For information on getting started with the Coherence JavaScript client, see Coherence JavaScript Client on GitHub.

Python

The Coherence Python Client allows Python applications to act as cache clients to an Oracle Coherence cluster using gRPC as the network transport.

For information on getting started with the Coherence Python client, see Coherence Python Client on GitHub.

Golang

The Coherence Go Client allows Go applications to act as cache clients to an Oracle Coherence cluster using gRPC as the network transport.

For information on getting started with the Coherence Go client, see Coherence Go Client on GitHub.

3
Understanding Configuration

The Coherence distribution includes default configuration files that applications and solutions override when creating their own Coherence configurations.

> **Note:**
>
> Coherence XML configuration files are validated at runtime using the corresponding XML schemas. It is important that any custom configuration files conform to the schema, because something as simple as having elements in the wrong order will cause validation to fail. As validation errors produced by XML schema validators can be difficult to understand, we recommend that custom configuration files are edited in a development environment capable of validating the XML against the schema as it is being written. This can save a lot of time compared to debugging validation errors at runtime.

This chapter includes the following sections:

Determining the Coherence Version

You can find out the Coherence version without starting a Coherence cluster.

For example, if you are in an Oracle installation home, use the following command to determine the version:

```
java -jar coherence/lib/coherence.jar --version
```

About Coherence Configuration File Formatting

For proper configuration validation, all configuration files should reference a schema (xsd). The schema dictates the order of elements.

Ensure that you specify the configuration parameters by following the sequence defined in the schema (XSD). Else, schema validation fails and prevents Coherence from starting.

When creating a configuration file, do not use the default configuration file names to ensure that configuration file selection is not based on classpath ordering.

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 file 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 Operational Configuration Elements.
- `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-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 Cache Configuration Elements.
- `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 POF User Type Configuration Elements.
- 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 Report File Configuration Elements and Report Group Configuration Elements in *Managing Oracle Coherence*.

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.

See also Using Coherence Clusters.

This section includes the following topics:

- Using the Default Operational Override File
- Specifying an Operational Override File
- Defining Override Files for Specific Operational Elements
- Viewing Which Operational Override Files are Loaded

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="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.Coherence
   ```

> **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 `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 `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 join timeout:

   ```
   <?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>
   ```

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 `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 -Dcoherence.override=cluster.xml -cp
   COHERENCE_HOME;COHERENCE_HOME\lib\coherence.jar com.tangosol.net.Coherence
   ```

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.

In general, override files for specific operational elements provide fine-grained control over which portions of the operational deployment descriptor may be modified and allow 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. See Using the Default Operational Override File.
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.Coherence
   ```

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.

See also Using Caches.

This section includes the following topics:

- 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

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.Coherence
   ```

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 and 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. See Using the Default Operational Override File.
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="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.Coherence
   ```

Using the Cache Configuration File System Property

The `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 `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 -Dcoherence.cacheconfig=MyConfig.xml -cp
COHERENCE_HOME;COHERENCE_HOME\lib\coherence.jar com.tangosol.net.Coherence
```

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 `pof-config.xml` file 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.

See also Using Portable Object Format .

This section includes the following topics:

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

Overriding the Default POF Configuration File

The default `pof-config.xml` POF configuration file is located in the root of the `coherence.jar` library. Coherence is configured out-of-box to use the first `pof-config.xml` file that is found on the classpath. To use a different `pof-config.xml` file, the file must be located on the classpath and must precede the `coherence.jar` library; otherwise, the default `pof-config.xml` file that is located in the `coherence.jar` library is used.

The POF configuration file should be customized for a particular application. The default POF configuration file references the `coherence-pof-config.xml` file. This is where the Coherence specific user types are defined and should always be included when creating a POF configuration file.

To override the Default 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 the file as `pof-config.xml` and close the file.
4. Ensure that the location of the POF configuration file is located in the classpath at run time.

   The following example demonstrates starting a cache server and using a POF configuration file that is located in *`COHERENCE_HOME`*.

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

Using the POF Configuration File System Property

The `coherence.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 `coherence.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 -Dcoherence.pof.config=MyPOF.xml -cp
COHERENCE_HOME;COHERENCE_HOME\lib\coherence.jar com.tangosol.net.Coherence
```

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.Coherence
```

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. The management configuration files 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` library. Additional report group configuration files 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` library 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.

See also Introduction to Oracle Coherence Management.

This section includes the following topics:

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.

This section includes the following topics:

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. See Using the Default Operational Override File.
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="coherence.management">
         all</managed-nodes>
      <reporter>
         <configuration system-property="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.Coherence
```

Using the Report Group Configuration File System Property

The `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 `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 -Dcoherence.management=all -Dcoherence.management.report.configuration=my-group.xml
-cp COHERENCE_HOME;COHERENCE_HOME\lib\coherence.jar com.tangosol.net.Coherence
```

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.

This section includes the following topics:

- Using the Default MBean Configuration Override File
- 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.Coherence
   ```

Using the MBean Configuration File System Property

The `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 `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 -Dcoherence.mbeans=my-mbeans.xml -cp
COHERENCE_HOME;COHERENCE_HOME\lib\coherence.jar com.tangosol.net.Coherence
```

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

You can use the XML override feature to change operational settings 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.

This section includes the following topics:

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

Using the Predefined Override Files

Multiple 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.
- `cache-factory-config.xml` – This override file is defined for the `<configurable-cache-factory-config>` element and is used to customize a configurable cache factory. This override file is typically only used to support container integrations. The `<configurable-cache-factory-config>` element is not commonly overridden.
- `cache-factory-builder-config.xml` – This override file is defined for the `<cache-factory-builder-config>` element and is used to customize a cache factory builder. This override file is typically only used to support container integrations. The `<cache-factory-builder-config>` element is not commonly overridden.

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="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="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 Operational Configuration Attribute Reference.

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 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="{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.
This section includes the following topics:

- 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 E-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 `coherence.log.level` system property is defined in the `tangosol-coherence.xml` file as follows:

```
<logging-config>
   ...
   <severity-level system-property="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 -Dcoherence.log.level=3 -cp COHERENCE_HOME\lib\coherence.jar
com.tangosol.net.Coherence
```

> **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

Extending Configuration Files

You can extend Coherence cache and operational configuration files using XML namespaces and namespace handler classes.The instructions in this chapter assume a general understanding of XML namespaces and XML processing.

Introduction to Extending Configuration Files

Coherence configuration files can include user-defined XML elements and attributes. The elements and attributes are declared within an XML namespace and are processed by a namespace handler at runtime. The namespace handler allows application logic to be executed based on the processing of the elements and attributes.

Coherence configuration files are typically extended to allow applications to perform custom initialization, background tasks, or perform monitoring and maintenance of a cluster. Extending the configuration files can be used for several different tasks.

By extending the cache configuration file, an application can:

- Configure elements using injected beans, for example CDI or Spring beans
- Establish domain-specific cache entry indexes
- Preload cached information
- Load configuration into a cluster
- Run or schedule background tasks against the cluster
- Integrate with external systems

By extending the operational configuration file, an application can:

- Configure elements using injected beans, for example CDI or Spring beans
- Add custom socket providers
- Add custom serializers
- Add custom security elements

Extending configuration files offers applications a common and consolidated place for configuration. In addition, application logic is embedded and managed in a cluster, which is a high-availability and scalable environment that can provide automated recovery of failed application logic if required.

> **Note:**
>
> Although the examples in this section are generally based on cache configuration files, you can extend operational configuration files and their elements in the same way.

Extending the Cache Configuration File

The following example extends a cache configuration file by declaring a `run` namespace that is associated with a `RunNamespaceHandler` namespace handler class. At runtime, the handler class processes the `<run:runnable>` element and its attributes and executes any logic on the cluster member as required.

```
<?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"
   xmlns:run="class://com.examples.RunNamespaceHandler">
   <run:runnable classname="MyRunnable" every="10m"/>
   ...
</cache-config>
```

Extending the Operational Configuration File

The following example extends a Coherence operational configuration file by declaring a `run` namespace that is associated with a `RunNamespaceHandler` namespace handler class. At runtime, the handler class processes the `<run:runnable>` element and its attributes and executes any logic on the cluster member as required.

```
<?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"
    xmlns:run="class://com.examples.RunNamespaceHandler">

   ...
</coherence>
```

Declaring XML Namespaces

Namespaces are declared in a configuration file by using a namespace declaration.The use of XML namespaces must adhere to the XML specification. At runtime, the XML syntax and XML namespaces are validated and checks are performed to ensure that the namespace prefixes have a corresponding `xmlns` declaration. Errors that are encountered in the cache configuration file result in the Coherence member failing to start.
The following example declares a namespace that uses the prefix `ex`:

```
<?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"
   xmlns:ex="URI">
   ...
```

The URI value for a namespace must be in the format `class://FullyQualifiedClassName`. If an incorrect format is provided, then a failure occurs and the Coherence member fails to start. The handler class provided in the declaration must be a class that implements the `NamespaceHandler` interface. See Creating Namespace Handlers.

The handler class must be able to process the associated XML elements and attributes as they occur within the cache configuration file. More specifically, during the processing of the XML DOM for the cache configuration, all XML elements and attributes that occur within a namespace are passed to the associated handler instance for processing. The following example uses the `MyNamespaceHandler` to process all elements that use the `ex` namespace prefix.

```
<?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"
   xmlns:ex="class://MyNamespaceHandler">

   <ex:myelement/>
   ...
```

Guidelines for Declaring an XML Namespace

Use the following guidelines when declaring an XML namespace:

- A URI class must implement the `NamespaceHandler` interface; otherwise, the processing of the cache configuration file fails.
- XML elements and attributes must not use an undeclared namespace prefix; otherwise, the processing of the cache configuration file fails.
- The default handler that is used for Coherence elements and attributes cannot be overridden; otherwise, the processing of the cache configuration file fails. The default namespace is reserved for Coherence.

Creating Namespace Handlers

Namespace handlers are used to process XML elements and attributes that belong to a specific XML namespace.Each unique namespace that is used in a cache configuration file requires a namespace handler. Namespace handlers must implement the `NamespaceHandler` interface. Typically, namespace handlers extend the base `AbstractNamespaceHandler` implementation class, which provides convenience methods that can simplify the processing of complex namespaces. Both of these APIs are discussed in this section and are included in the `com.tangosol.config.xml` package.
This section includes the following topics:

- Implementing the Namespace Handler Interface
- Extending the Namespace Handler Abstract Class

Implementing the Namespace Handler Interface

Namespace handlers process the elements and attributes that are used within an XML namespace. Namespace handlers can directly implement the `NamespaceHandler` interface. The interface relies on the `DocumentPreprocessor`, `ElementProcessor`, and `AttributeProcessor` interfaces. XML processing is performed within a processing context as defined by the `ProcessingContext` interface.

Elements and attributes that are encountered in a namespace must be processed by a processor implementation. Element and attribute processors are responsible for processing, parsing, and type conversion logic. Document preprocessors are used to mutate elements, if required, before they are processed.

Example 4-1 provides a basic `NamespaceHandler` implementation. The `GreetingNamespaceHandler` implementation processes a `<message>` element using an `ElementProcessor` implementation (`MessageProcessor`), which is included as an inner class. For the example, the following XML is assumed:

```
<?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"
   xmlns:ex="class://com.examples.GreetingNamespaceHandler">

   <ex:message>hello</ex:message>
   ...
```

Example 4-1 Handler Implementation Using the NamespaceHandler Interface

```
import com.tangosol.config.ConfigurationException;
import com.tangosol.config.xml.AttributeProcessor;
import com.tangosol.config.xml.DocumentPreprocessor;
import com.tangosol.config.xml.ElementProcessor;
import com.tangosol.config.xml.NamespaceHandler;
import com.tangosol.config.xml.ProcessingContext;
import com.tangosol.run.xml.XmlAttribute;
import com.tangosol.run.xml.XmlElement;
import java.net.URI;

public class GreetingNamespaceHandler implements NamespaceHandler
{
   public AttributeProcessor getAttributeProcessor(XmlAttribute xmlAttribute)
   {
      return null;
   }

   public DocumentPreprocessor getDocumentPreprocessor()
   {
      return null;
   }

   public ElementProcessor getElementProcessor(XmlElement xmlElement)
   {
      if (xmlElement.getName().equals("ex:message"))
      {
         MessageProcessor  mp = new MessageProcessor();
         return mp;
      }
      else
      {
```

```
            throw new RuntimeException("Unknown element type " +
               xmlElement.getQualifiedName());
         }
      }

      public void onEndNamespace(ProcessingContext processingContext,
         XmlElement xmlElement, String string, URI uRI)
      {
      }

      public void onStartNamespace(ProcessingContext processingContext,
         XmlElement xmlElement, String string, URI uRI)
      {
      }

      public class MessageProcessor implements ElementProcessor
      {
         public Object process(ProcessingContext processingContext,
            XmlElement xmlElement) throws ConfigurationException
         {
            System.out.println("Greeting is: " + xmlElement.getString());
            return null;
         }
      }
}
```

The above class handles the `<ex:message>` element content. However, it does not distinguish between the different types of elements that may occur. All elements that occur within the default XML namespace are provided to the same `process` method. In order to process each type of XML element, conditional statement are required within the `process` method. This technique may be sufficient for trivial XML content; however, more complex XML content may require many conditional statements and can become overly complicated. A more declarative approach is provided with the `AbstractNamespaceHandler` class.

Namespace Handler Callback Methods

The `NamespaceHandler` interface provides the `onStartNamespace` and the `onEndNamespace` callback methods. These methods allow additional processing to be performed on the first and last encounter of a namespace in a cache configuration file.

Extending the Namespace Handler Abstract Class

The `AbstractNamespaceHandler` class provides a useful and extensible base implementation of the `NamespaceHandler`, `ElementProcessor` and `AttributeProcessor` interfaces together with mechanisms to register processors for specifically named elements and attributes. The class simplifies the processing of elements and attributes and can, in most cases, remove the requirement to directly implement the element and attribute processor interfaces.

This section contains the following topics:

- Registering Processors
- Using Injection to Process Element Content

Registering Processors

The `AbstractNamespaceHandler` class provides methods for declaratively registering element and attribute processors. The methods alleviate the need to check element names and types. There are two registration mechanisms: explicit registration and implicit registration.

Explicit Processor Registration

To use explicit processor registration, call the `registerProcessor` method within a sub-class constructor and manually register both element and attribute processors. Example 4-2 re-implements Example 4-1 and uses the `registerProcessor` method to register the `MessageProcessor` element processor.

Example 4-2 AbstractNamespaceHandler Implementation with Explicit Registration

```
import com.tangosol.config.ConfigurationException;
import com.tangosol.config.xml.AbstractNamespaceHandler;
import com.tangosol.config.xml.ElementProcessor;
import com.tangosol.config.xml.ProcessingContext;
import com.tangosol.run.xml.XmlElement;

public class GreetingNamespaceHandler extends AbstractNamespaceHandler
{
   public GreetingNamespaceHandler()
   {
      registerProcessor("message", new MessageProcessor());
   }

   public class MessageProcessor implements ElementProcessor
   {
      public Object process(ProcessingContext processingContext,
         XmlElement xmlElement) throws ConfigurationException
      {
         System.out.println("Greeting is: " + xmlElement.getString());
         return null;
      }
   }
}
```

Implicit Processor Registration

To use implicit processor registration, annotate processor classes with the `@XmlSimpleName` annotation. The processors are automatically registered for use within the associated namespace.The `@XmlSimpleName` annotation is used to determine which of the processor implementations are appropriate to handle XML content encountered during XML DOM processing. If an XML element or attribute is encountered for which there is no defined processor, then the `onUnknownElement` or `onUnknownAttribute` methods are called, respectively. The methods allow corrective action to be taken if possible. By default, a `ConfigurationException` exception is raised if unknown XML content is discovered during processing.

Example 4-3 re-implements Example 4-1 and uses the `@XmlSimpleName` annotation to register the `MessageProcessor` element processor.

Example 4-3 AbstractNamespaceHandler Implementation with Implicit Registration

```
import com.tangosol.config.ConfigurationException;
import com.tangosol.config.xml.AbstractNamespaceHandler;
import com.tangosol.config.xml.ElementProcessor;
import com.tangosol.config.xml.ProcessingContext;
import com.tangosol.config.xml.XmlSimpleName;
import com.tangosol.run.xml.XmlElement;

public class GreetingNamespaceHandler extends AbstractNamespaceHandler
{
   public GreetingNamespaceHandler()
```

```
   {
   }

   @XmlSimpleName("message")
   public class MessageProcessor implements ElementProcessor
   {
      public Object process(ProcessingContext processingContext,
         XmlElement xmlElement) throws ConfigurationException
      {
         System.out.println("Greeting is: " + xmlElement.getString());
         return null;
      }
   }
}
```

Using Injection to Process Element Content

Element and attribute processors are used to hand-code the processing of XML content. However, the task can be repetitive for complex namespaces. To automate the task, the `ProcessingContext.inject` method is capable of injecting strongly typed values into a provided object, based on identifiable setter methods and values available for a specified XML element.

For example, given the following XML:

```
<ex:message>
   <ex:english>hello</ex:english>
</ex:message>
```

An element processor can be used to hand-code the processing of the `<english>` element:

```
import com.tangosol.config.ConfigurationException;
import com.tangosol.config.xml.AbstractNamespaceHandler;
import com.tangosol.config.xml.ElementProcessor;
import com.tangosol.config.xml.ProcessingContext;
import com.tangosol.config.xml.XmlSimpleName;
import com.tangosol.run.xml.XmlElement;

public class GreetingNamespaceHandler extends AbstractNamespaceHandler
{
   public GreetingNamespaceHandler()
   {
   }

   @XmlSimpleName("message")
   public class MessageProcessor implements ElementProcessor
   {
      public Object process(ProcessingContext processingContext,
         XmlElement xmlElement) throws ConfigurationException
      {
         String engMsg = processingContext.getMandatoryProperty("english",
            String.class, xmlElement);
         Message message = new Message();
         message.setEnglish(engMsg);
         System.out.println("Greeting is: "+ message.getEnglish());

         return message;
      }
   }
}
```

As an alternative, the `inject` method can perform the processing:

```
@XmlSimpleName("message")
public class MessageProcessor implements ElementProcessor
{
   public Object process(ProcessingContext processingContext,
      XmlElement xmlElement) throws ConfigurationException
   {
      return processingContext.inject(new Message(), xmlElement);
   }
}
```

The `inject` method uses Java reflection, under the assumption that the object to be configured follows the Java bean naming conventions. First the `inject` method identifies the appropriate setter methods that may be called on the Java bean. Typically, this is achieved by locating setter methods that are annotated with the `@Injectable` annotation. Next, it determines the appropriate types of values required by the setter methods. Lastly, it uses the provided `XmlElement` to locate, parse, convert, coerce, and then set appropriately typed values into the object using the available setter methods. The `inject` method supports primitives, enumerations, formatted values, complex user-defined types, and collection types (sets, lists, and maps). A `ConfigurationException` exception is thrown if the `inject` method fails. For example, it fails to format a value into the expected type.

The following example demonstrates a `Message` class for the above example that supports injection with the `inject` method:

```
import com.tangosol.config.annotation.Injectable;

public class Message
{

   private String m_sEnglish = "a greeting";
   public Message()
   {
   }

   @Injectable("english")
   public void setEnglish (String sEnglish)
   {
      m_sEnglish = sEnglish;
   }

   public String getEnglish()
   {
      return m_sEnglish;
   }
}
```

> **Note:**
>
> If the `@Injectable` annotation property is omitted, then the `inject` method tries to use the setter method Java bean name. For the above example, `@injectable("")` results in the use of `english`.

Typically, element and attribute processors follow the same pattern when using the `inject` method. For example:

```
@XmlSimpleName("element")
public class XProcessor implements ElementProcessor
{
   public Object process(ProcessingContext processingContext,
      XmlElement xmlElement) throws ConfigurationException
   {
      return processingContext.inject(new X(), xmlElement);
   }
}
```

Declarations and registrations for such implementations can be automated using the `registerElementType` and `registerAttributeType` methods for elements and attributes, respectively. These methods are available in the `AbstractNamespaceHandler` class and are often used in constructors of `AbstractNamespaceHandler` sub-classes. The following example demonstrates using the `registerElementType` method and does not require a processor implementation.

> **Note:**
>
> To support type-based registration of classes, the specified class must provide a no-argument constructor.

```
import com.tangosol.config.xml.AbstractNamespaceHandler;

public class GreetingNamespaceHandler extends AbstractNamespaceHandler
{

   public GreetingNamespaceHandler()
   {
   registerElementType("message", Message.class);
   }
}
```

Example: the JNDI Resource Namespace Handler

The JNDI resource namespace handler provides the ability to lookup and reference resources defined by a JNDI context.The use of the namespace handler is often used to replace the need to statically create resources using the `<class-scheme>` or `<instance>` elements in the cache configuration file.
This section includes the following topics:

- Create the JNDI Resource Namespace Handler
- Declare the JNDI Namespace Handler
- Use the JNDI Resource Namespace Handler

Create the JNDI Resource Namespace Handler

The JNDI resource namespace handler is used at runtime to process `<resource>` elements that are found in a cache configuration file. The handler extends the `AbstractNamespaceHandler` class and registers the `JndiBasedParameterizedBuilder` class for the `<resource>` element. The following example shows the namespace handler definition.

```
import com.tangosol.coherence.config.builder.ParameterizedBuilder;
import com.tangosol.config.xml.AbstractNamespaceHandler;
```

```
public class JndiNamespaceHandler extends AbstractNamespaceHandler
{
    public JndiNamespaceHandler()
    {
        registerElementType("resource", JndiBasedParameterizedBuilder.class);
    }
}
```

The `JndiBasedParameterizedBuilder` class performs a JNDI context lookup to locate and create an object using the name and initialization parameters that are provided in the `<resource-name>` and `<init-parms>` elements, respectively. The setter methods for these elements (`setResourceNameExpression` and `setParameterList`) use the `@Injectable` annotation to pass the values configured in the cache configuration files.

```
import com.tangosol.coherence.config.ParameterList;
import com.tangosol.coherence.config.SimpleParameterList;
import com.tangosol.coherence.config.builder.ParameterizedBuilder;
import com.tangosol.coherence.config.builder.ParameterizedBuilder.
   ReflectionSupport;
import com.tangosol.config.annotation.Injectable;
import com.tangosol.config.expression.Expression;
import com.tangosol.config.expression.LiteralExpression;
import com.tangosol.config.expression.Parameter;
import com.tangosol.config.expression.ParameterResolver;
import com.tangosol.util.Base;
import java.util.Hashtable;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class JndiBasedParameterizedBuilder implements
   ParameterizedBuilder<Object>, ReflectionSupport
{   private static final Logger logger =
      Logger.getLogger(JndiBasedParameterizedBuilder.class.getName());

   private Expression<String> m_exprResourceName;

   private ParameterList m_parameterList;

   public JndiBasedParameterizedBuilder()
   {
      m_exprResourceName = new LiteralExpression<String>("");
      m_parameterList = new SimpleParameterList();
   }

   public Expression<String> getResourceNameExpression()
   {
      return m_exprResourceName;
   }

   @Injectable("resource-name")
   public void setResourceNameExpression(Expression<String> exprResourceName)
   {
      m_exprResourceName = exprResourceName;
   }

   public ParameterList getParameterList()
   {
      return m_parameterList;
   }
```

```
   @Injectable("init-params")
   public void setParameterList(ParameterList parameterList)
   {
      m_parameterList = parameterList;
   }

   public boolean realizes(Class<?> clazz,
                           ParameterResolver parameterResolver,
                           ClassLoader classLoader)
   {
      return clazz.isAssignableFrom(realize(parameterResolver, classLoader,
         null).getClass());
    }

   public Object realize(ParameterResolver parameterResolver,
                         ClassLoader classLoader,
                         ParameterList parameterList)
   {
      InitialContext initialContext;
       try
      {
         String sResourceName = m_exprResourceName.evaluate(parameterResolver);
         Hashtable<String, Object> env = new Hashtable<String, Object>();

         for (Parameter parameter : m_parameterList)
         {
            env.put(parameter.getName(), parameter.evaluate(parameterResolver));
         }

         initialContext = new InitialContext(env);

         if (logger.isLoggable(Level.FINE))
         {
            logger.log(Level.FINE,
                       "Looking up {0} using JNDI with the environment {1}",
                        new Object[] {sResourceName, env});
         }

         Object resource = initialContext.lookup(sResourceName);

         if (logger.isLoggable(Level.FINE))
         {
            logger.log(Level.FINE, "Found {0} using JNDI", resource);
         }

         return resource;
      }
      catch (NamingException e)
      {
         throw Base.ensureRuntimeException(e, "Unable to resolve the JNDI
            resource: " + m_exprResourceName.toString());
      }
   }

   public String toString()
   {
      return String.format("%s{resourceName=%s, parameters=%s}",
                           this.getClass().getName(), m_exprResourceName,
                           m_parameterList);
   }
}
```

Declare the JNDI Namespace Handler

The JNDI resource namespace handler must be declared within the cache configuration file. Declare the namespace by providing the URI to the handler class and assigning a namespace prefix. Any prefix can be used. The following example uses `jndi` as the prefix:

```
<?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"
   xmlns:jndi="class://com.examples.JndiNamespaceHandler">
...
```

Use the JNDI Resource Namespace Handler

The JNDI resource namespace handler can be used whenever an application requires a resource to be located and created. In addition, the namespace can be used when defining custom implementations using the `<class-scheme>` or `<instance>` elements. Based on the handler implementation, the `<resource>` and `<resource-name>` element are required and the `<init-params>` element is optional.

The following example uses a JNDI resource for a cache store when defining a distributed scheme:

```
<?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"
   xmlns:jndi="class://com.examples.JndiNamespaceHandler">
...

   <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>
                     <jndi:resource>
                        <jndi:resource-name>MyCacheStore</jndi:resource-name>
                        <init-params>
                           <init-param>
                              <param-type>java.lang.String</param-type>
                              <param-value>{cache-name}</param-value>
                           </init-param>
                        </init-params>
                     </jndi:resource>
                  </class-scheme>
               </cachestore-scheme>
            </read-write-backing-map-scheme>
         </backing-map-scheme>
      </distributed-scheme>
   <caching-schemes>
</cache-config>
```

The following example uses a JNDI resource to resolve a DNS record:

```
<?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"
   xmlns:jndi="class://com.examples.JndiNamespaceHandler">
...
<jndi:resource>
   <jndi:resource-name>dns:///www.oracle.com</jndi:resource-name>
</jndi:resource>
```

The following example uses a JNDI resource to resolve a JMS connection factory:

```
<?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"
   xmlns:jndi="class://com.examples.JndiNamespaceHandler">
...

<jndi:resource>
   <jndi:resource-name>ConnectionFactory</jndi:resource-name>
   <init-params>
      <init-param>
         <param-name>java.naming.factory.initial</param-name>
         <param-value>org.apache.activemq.jndi.ActiveMQInitialContextFactory
         </param-value>
      </init-param>
      <init-param>
         <param-name>java.naming.provider.url</param-name>
         <param-value system-property="java.naming.provider.url"></param-value>
      </init-param>
   </init-params>
</jndi:resource>
```

5

Debugging in Coherence

An important part of Coherence application development is debugging, troubleshooting, tracing, and logging.
This chapter includes the following sections:

- Overview of Debugging in Coherence
- Configuring Logging
- 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.
- Distributed Tracing
- Troubleshooting Coherence-Based Applications
  Troubleshooting Coherence-based applications is, for the most part, no different than troubleshooting other Java applications.

Overview of Debugging in Coherence

Coherence applications are typically developed on a single computer. A cache server and application are started within an IDE and the application is debugged as required.This type of development environment is easy to set up, performs well, and is easy to debug. A majority of applications can be created and tested this way. See Enabling Single-Server Mode.
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 VisualVM and JConsole, 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 affects the application. Remote debugging with Java Debug Wire Protocol (JDWP) together with Coherence's JMX management and reporting capabilities facilitates debugging and troubleshooting in a distributed environment.

Using Oracle Support

My Oracle Support can help debug issues. 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 Log4j2, SLF4J, 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.
This section includes the following topics:

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

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

```
-Dcoherence.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 JDK, Log4j2, and SLF4J to allow an application and Coherence to share a common logging framework. See Using JDK Logging for Coherence Logs, Using Log4j2 Logging for Coherence Logs, and Using SLF4J for Coherence Logs, respectively.

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

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

```
-Dcoherence.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. Processes 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="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 parameter 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 parameter shows the text of the message.
`{ecid}`	This parameter shows the Execution Context ID (ECID). The ECID is a globally unique ID that is attached to requests between Oracle components. The ECID is an Oracle-specific diagnostic feature and is used to correlate log messages across log files from Oracle components and products and is also used to track log messages pertaining to the same request within a single component when multiple requests are processed in parallel. Coherence clients that want to include the ECID in their logs must have an activated Dynamic Monitoring Service (DMS) execution context when invoking Coherence. **Note:** If JDK logging is used with an Oracle Diagnostic Logging (ODL) handler, then the `{ecid}` parameter does not apply because the ECID automatically becomes part of the ODL record form.

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

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

```
-Dcoherence.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 Java Logging Overview in *Java SE Core Libraries*.

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 and file is configured to use the `FINEST` log level. For the file handler pattern, the specified path must already exist. Also, ensure that the specified directory can be accessed and has write permissions.

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

   .level=INFO
   Coherence.level=FINEST

   java.util.logging.FileHandler.pattern=/tmp/coherence%u.log
   java.util.logging.FileHandler.limit=50000
   ```

```
java.util.logging.FileHandler.count=1
java.util.logging.FileHandler.formatter=java.util.logging.SimpleFormatter

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

> **Note:**
>
> - In the above example, `Coherence` is used as the logger object name and is the default name that is used by the Coherence logging framework. 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 `coherence.log.logger` system property.
> - Set the JDK logging level to `FINEST` and allow the Coherence logging settings to determine which log messages to construct for JDK logging.

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

3. Make sure the `logging.properties` file is specified using the `java.util.logging.config.file` system property. For example:

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

Mapping JDK Log Levels with Coherence Log Levels

Table 5-1 provides a mapping of how JDK log levels are mapped to Coherence log levels.

Table 5-1 Mapping JDK Log Levels

JDK Log Level	Coherence Log Level
OFF	NONE
FINEST	INTERNAL
SEVERE	ERROR
WARNING	WARNING
INFO	INFO
FINE	LEVEL_D4
FINER	LEVEL_D5
FINEST	LEVEL_D6
FINEST	LEVEL_D7
FINEST	LEVEL_D8
FINEST	LEVEL_D9
ALL	ALL

Using Log4j2 Logging for Coherence Logs

Applications that use the Log4j2 logging framework can configure Coherence to use Log4j2 logging as well. Detailed information about Log4j2 logging is beyond the scope of this documentation. For details on Log4j2 logging, see http://logging.apache.org/log4j/2.x/manual/index.html.

To use Log4j2 logging for Coherence logs:

1. Create a `log4j2.properties` file. The following example configures the Log4j2 logger to emit messages to both the console and to a file. Output to the console and file are configured to use the `FATAL` log level. For the file appender, make sure the specified directory can be accessed and has write permissions.

```
name = PropertiesConfig
property.filename = /tmp/coherence.log

appender.console.type = Console
appender.console.name = STDOUT
appender.console.layout.type = PatternLayout
appender.console.layout.pattern = %m%n

appender.file.type = File
appender.file.name = LogFile
appender.file.fileName = ${filename}
appender.file.layout.type = PatternLayout
appender.file.layout.pattern = %m%n

loggers = file
logger.file.name = Coherence
logger.file.level = FATAL
logger.file.appenderRefs = file
logger.file.appenderRef.file.ref = LogFile
```

> **Note:**
>
> - In the above example, `Coherence` is used as the logger object name and is the default name that is used by the Coherence logging framework. 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 `coherence.log.logger` system property.
> - Set the Log4j2 logging level to `FATAL` and allow the Coherence logging settings to determine which log messages to construct for Log4j2 logging.

2. Configure Coherence to use Log4j2 logging by specifying `log4j2` as the value of the `<destination>` element in an operational override file. For example:

```
...
<logging-config>
   <destination system-property="coherence.log">log4j2</destination>
   ...
</logging-config>
...
```

3. Ensure that the Log4j2 API jar, Log4j2 core jar, and `log4j2.properties` file are found on the classpath at runtime.

Mapping Log4j2 Log Levels with Coherence Log Levels

Table 5-2 provides a mapping of how Log4j2 log levels are mapped to Coherence log levels.

Table 5-2 Mapping Log4j2 Log Levels

Log4j2 Log Level	Coherence Log Level
OFF	NONE
DEBUG	INTERNAL
ERROR	ERROR
WARN	WARNING
INFO	INFO
DEBUG	LEVEL_D4
DEBUG	LEVEL_D5
DEBUG	LEVEL_D6
DEBUG	LEVEL_D7
DEBUG	LEVEL_D8
DEBUG	LEVEL_D9
ALL	ALL

Using SLF4J for Coherence Logs

Applications that use SLF4J logging can configure Coherence to use SLF4J logging as well. Detailed information about SLF4J logging is beyond the scope of this documentation. For details on SLF4J logging, see `http://www.slf4j.org/`.

To use SLF4J logging:

1. specify `slf4j` as the value of the `<destination>` element in an operational override file. For example:

```
...
<logging-config>
   <destination system-property="coherence.log">slf4j</destination>
   ...
</logging-config>
...
```

2. Make sure both the `slf4j-api.jar` file and the appropriate logging framework SLF4J binding JAR is located on the classpath.

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 holds which data, 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.

Distributed Tracing

Coherence can use OpenTelemetry or OpenTracing APIs to give developers visibility into the cache operations within the cluster.
Coherence does not include any tracing implementation libraries. Therefore, the developer needs to provide the desired tracing runtime. Since OpenTracing is no longer maintained, Oracle recommends that you use OpenTelemetry. A minimum version of OpenTelemetry for Java version 1.48.0 is recommended. Even though OpenTracing is deprecated in Coherence, it is still a supported option when using the latest OpenTracing 0.33.0.

If you are using OpenTracing, and want Coherence to initialize the tracing runtime, include OpenTracing's TracerResolver in your project's classpath.

If you are using OpenTelemetry, and want Coherence to initialize the tracing runtime, include OpenTelemetry's SDK Autoconfigure in your project's classpath.

In either case, if these dependencies are not satisfied, it is assumed that the configuration of the tracing runtime is being managed outside of Coherence. Therefore, Coherence will take no action to initialize the runtime itself.

If using `coherence-grpc-proxy` or `coherence-java-client`, you may opt to trace the gRPC calls made by these libraries by including an additional dependency to your project. For OpenTracing, include OpenTracing gRPC Instrumentation and for OpenTelemetry, include Library Instrumentation for gRPC 1.6.0+.

This section includes the following topics:

- Configuring Tracing
- Traced Operations in Coherence
- User-initiated Tracing

Configuring Tracing

To configure tracing, edit the operational override `tangosol-coherence-override.xml` file and add a `<tracing-config>` element with a child `<sampling-ratio>` element.

For example:

```
...
<tracing-config>
  <sampling-ratio>0</sampling-ratio> <!-- user-initiated tracing -->
```

```
</tracing-config>
...
```

Tracing operates in three modes:

- **-1** - This value disables tracing.
- **0** - This value enables user-initiated tracing. This means that Coherence will not initiate tracing on its own and the application should start an outer tracing span, from which Coherence will collect the inner tracing spans. If the outer tracing span is not started, the tracing activity will not be performed.
- **0.01-1.0** - This range indicates the tracing span being collected. For example, a value of 1.0 will result in all spans being collected, while a value of 0.1 will result in roughly 1 out of every 10 spans being collected.

The `coherence.tracing.ratio` system property can be used to specify the tracing sampling ratio instead of using the operational override file. For example:

```
-Dcoherence.tracing.ratio=0
```

Configuring OpenTelemetry

OpenTelemetry will be configured according to the options defined by their SDK configuration documentation. However, Coherence applies defaults to the following properties:

OpenTelemetry Option	Coherence Default Value
`otel.service.name`	A value combining the name of the cluster and the member's role: `cluster-name.member-role-name`. If these two values can't be resolved, this value will fall back to Coherence.
`otel.traces.exporter`	Defaults to `none`. In order to capture traces, explicit configuration of the exporter is required.
`otel.metrics.exporter`	Defaults to `none`. In order to capture metrics, explicit configuration of the exporter is required.
`otel.logs.exporter`	Defaults to `none`. In order to capture logs, explicit configuration of the exporter is required.

As of 14.1.2.0.3, Coherence will no longer initialize OpenTelemetry using `io.opentelemetry.api.GlobalOpenTelemetry`. If you want to continue using `GlobalOpenTelemetry`, include `-Dotel.java.global-autoconfigure.enabled=true` when starting the JVM.

Traced Operations in Coherence

The following Coherence traced operations may be captured:

- All operations exposed by the `NamedCache` API when using partitioned caches.
- Events processed by event listeners (such as `EventInterceptor` or `MapListener)` .
- Persistence operations.
- CacheStore operations.
- Coherence gRPC calls (both client and server).

User-initiated Tracing

When the sampling ratio is set to zero, the application will be required to start a tracing span prior to invoking a Coherence operation.

For example:

```
Tracer tracer = GlobalOpenTelemetry.getTracer("your-tracer");

Span span = tracer.spanBuilder("test").startSpan();

NamedCache cache = CacheFactory.getCache("some-cache")

try (Scope scope = span.makeCurrent())
    {
    cache.put("a", "b");
    cache.get("a");
    }
finally
    {
    span.end();
    }
```

> **Note:**
>
> When user-initiated tracing is enabled, no tracing spans will be captured unless the application starts an active outer tracing span.

Troubleshooting Coherence-Based Applications

Troubleshooting Coherence-based applications is, for the most part, no different than troubleshooting other Java applications.

Most IDEs provide features that facilitate the process. In addition, many tools, such as: VisualVM, JConsole, and third-party tools provide easy ways to monitor and troubleshoot Java applications. See Prepare Java for Troubleshooting in *Java Platform, Standard Edition Troubleshooting Guide*.

Troubleshooting a Coherence application on a single server cluster is typically straightforward. 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.

This section includes the following topics:

- Using Coherence Logs
- Using JMX Management and Coherence Reports
- Using JVM Options to Help Debug
- Using Distributed Tracing
- 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. See Log Message Glossary in *Administering Oracle Coherence*. 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 Log4j2 logging) to ensure that all log messages are emitted. Also, consider using either JDK or Log4j2 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.

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 Configuring JMX Management and Enabling Oracle Coherence Reporting on a Cluster Member in *Managing Oracle 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.

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.

Using Distributed Tracing

Distributed tracing allows developers to profile and monitor applications. This is particularly useful in a clustered environment such as Coherence. See Distributed Tracing .

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.

Coherence provides a native `logClusterState` JMX operation that is located on the `ClusterMBean` MBean and a native `logNodeState` JMX operation that is located on the `ClusterNodeMBean` MBean. These operations initiate a thread dump (including outstanding polls) on multiple cluster members or on a single cluster member, respectively. See ClusterMBean and ClusterNodeMBean in *Managing Oracle Coherence*.

To perform a thread dump locally 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 VisualVM (`visualvm`) and JConsole (`jconsole`) 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.

> **Note:**
>
> By default, the thread dumps generated by Coherence, either through the `ClusterMBean` MBean or the service guardian, do not include lock and deadlock analysis. Thread lock and deadlock analysis may have a significant performance impact. To include the analysis for troubleshooting purposes, you can set the system property `com.oracle.coherence.common.util.Threads.dumpLocks` to `true`.

The following WLST script can be used to trigger cluster wide thread dumps when running Coherence in a WebLogic domain:

```
# connect to domain runtime mbean server
connect(adminUser, adminPasswd, "t3://%s:%s" % (adminHost, adminPort))
domainRuntime()

# obtain coherece cluster mbean
cohClusterON=ObjectName("Coherence:type=Cluster,cluster=%s" % cohClusterName)
cohClusterMBean=list(mbs.queryMBeans(cohClusterON, None))

# obtain all mbean of coherence nodes associated with this cluster
cohClusterNodes=list(mbs.queryMBeans(ObjectName("Coherence:type=Node,cluster=%
s,*" % cohClusterName), None))

# take a thread dump on each coherence member
types=["java.lang.String"]
for node in cohClusterNodes:
  roleName=mbs.getAttribute(node.getObjectName(), "RoleName")
  mbs.invoke(cohClusterMBean[0].getObjectName(), 'logClusterState',
[roleName], types)
```

Capturing Heap Dumps

Heap dumps are used to see detailed information for all the objects in a JVM heap. The information includes how many instances 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 VisualVM (`visualvm`) and JConsole (`jconsole`) 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 Operating System Tuning in *Administering Oracle Coherence*.

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 Performing a Network Performance Test in *Administering Oracle Coherence*.

Part II
Using Coherence Clusters

Learn how to setup Coherence clusters, configure cluster members, tune cluster communication, and use the service guardian.
Part II contains the following chapters:

6

Introduction to Coherence Clusters

Before creating a Coherence cluster, take some time to understand Coherence clustered services and how cluster members use the Tangosol Cluster Management Protocol (TCMP) to communicate with each other.
This chapter includes the following sections:

Cluster Overview

A Coherence cluster is a network of JVM processes that run Coherence. JVMs automatically join together to form a cluster and are called cluster members or cluster nodes.Cluster members communicate using the 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 root 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 Clustered Services

Coherence functionality is based on the concept of services. Each cluster member can register, provide, and consume services.Multiple services can be running on a cluster member. A cluster member always contains a single root cluster service and can also contain any number of grid services. Grid services have 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 instances of each service type (other than the root cluster service).
The services are categorized below based on functionality. The categories are used for clarity and do not represent actual components or imply a relationship between services.

Connectivity Services

- **Cluster Service:** This service is automatically started when a cluster node must join the cluster and is often referred to as the root cluster service; 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 de-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. See Understanding Distributed Caches.
- **Federated Cache Service:** This service is a version of the distributed cache service that replicates and synchronizes cached data across geographically dispersed clusters that are participants in a federation. Replication between clusters participants is controlled by the federation topology. Topologies include: active-active, active-passive, hub and spoke, and central replication. Custom topologies can also be created as required. See Federating Caches Across Clusters in *Administering Oracle Coherence*.

A clustered service can perform all tasks on the service thread, a caller's thread (if possible), or any number of daemon (worker) threads. Daemon threads are managed by a dynamic thread pool that provides the service with additional processing bandwidth. For example, the invocation service and the distributed cache service both support thread pooling to accelerate database load operations, parallel distributed queries, and agent invocations.

A dynamic thread pool has a resize task to automatically adjust the daemon pool size based on the current load and throughput of the service. The task is created at the start of the daemon pool. When a daemon pool is stopped, its resizing task is also stopped.

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`.

Understanding TCMP

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

Cluster Service Communication

TCMP for cluster service communication can be configured to use:

- A combination of UDP/IP multicast and UDP/IP unicast. This is the default cluster protocol for cluster service communication.

- UDP/IP unicast only (that is, no multicast). See Disabling Multicast Communication. 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. This configuration is used for network environments that favor TCP.
- SDP/IP only (no UDP/IP multicast or UDP/IP unicast). See Using the SDP Socket Provider. This configuration is used for network environments that favor SDP.
- SSL over TCP/IP or SDP/IP. See Using the SSL Socket Provider. This configuration is used for network environments that require highly secure communication between cluster members.

Data Service Communication

TCMP for data service communication can be configured to use a reliable transport:

- datagram – Specifies the use of the TCMP reliable UDP protocol.
- tmb (default)– Specifies the TCP Message Bus (TMB) protocol. TMB provides support for TCP/IP.
- tmbs – TCP/IP message bus protocol with SSL support. TMBS requires the use of an SSL socket provider. See Using the SSL Socket Provider.
- sdmb – Specifies the Sockets Direct Protocol Message Bus (SDMB).
- sdmbs – SDP message bus with SSL support. SDMBS requires the use of an SSL socket provider. See Using the SSL Socket Provider.

Use of Multicast

The cluster protocol makes very minimal and judicious 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. Only the two most senior cluster members join a multicast group. Additional cluster members do not need to join the multicast group to discover the cluster.
- 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. The cluster heart beat is used to detect member failure and helps avoid when a cluster splits into two independently operating clusters (referred to as split brain).

Use of Unicast

Cluster members use unicast for direct member-to-member (point-to-point) communication, which makes up the majority of communication on the cluster. Unicast is used as follows:

- Data transfer: Unicast is used to transfer data between service members on a shared bus instance.
- Message delivery: Unicast is used for communication such as asynchronous acknowledgments (ACKs), asynchronous negative acknowledgments (NACKs) and peer-to-peer heartbeats.
- 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:

- TCMP uses a shared TCP/IP Message Bus (TMB) for data transfers.
- 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).

Protocol Reliability

Bus-based transport protocols inherently support message reliability. For UDP transport, Coherence provides fully reliable and in-order delivery of all messages; Coherence uses a queued and 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 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.

> **Note:**
>
> For TCMP/TMB, which is the default protocol for point-to-point data communication, each cluster member binds to a single port.

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.

7

Setting Up a Cluster

You can change the default out-of-box cluster settings to set up and configure unique Coherence clusters for your solution.
This chapter includes the following sections:

- Overview of Setting Up Clusters
- Specifying a Cluster's Name
- Specifying a Cluster Member's Identity
- Configuring Multicast Communication
- Specifying a Cluster Member's Unicast Address
- 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.
- Enabling Single-Server Mode
- Configuring Death Detection
- Specifying Cluster Priorities
- Configuring Firewalls for Cluster Members
  Firewalls are not typically setup between Coherence cluster members because it is assumed that all cluster members communicate behind the firewall in a secured environment. Multiple ports must be opened in a firewall if you intend to secure communication between cluster members. Note that firewalls tend to be used less and less in cloud environments, where network security is achieved at the infrastructure level (seclists).

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 testing and demonstration, the default setup should not be used. You should set up unique clusters 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 basic development.
Setting up a cluster includes defining the cluster's name. 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 and Operational Configuration Elements.

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 form a cluster. A cluster member does not start if the wrong name is specified when attempting to join an existing cluster.

> **Note:**
>
> If a name is not explicitly specified, then a cluster name is automatically generated based on the operating system user name. The recommended best practice is to not use the system generated cluster name.

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="coherence.cluster">MyCluster
         </cluster-name>
      </member-identity>
   </cluster-config>
</coherence>
```

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

```
-Dcoherence.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. The site name 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 site name also helps determine where to back up data when using distributed caching and the default partition assignment strategy. Lastly, the name is useful for displaying management information (for example, JMX) and interpreting log entries.

- Rack Name – the name of the location within a geographic site that the member is hosted at and is often a cage, rack, or bladeframe identifier. The rack name 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 rack name also helps determine where to back up data when using distributed caching and the default partition assignment strategy. Lastly, the name is useful for displaying management information (for example, JMX) and interpreting log entries.
- 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.

In general, you need to only set `member-name` and `role-name`. If you are an advanced user and your cluster environment is across multiple sites, racks, and such, then you also can set `site-name`, `rack-name`, and such, accordingly. 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="coherence.site">site-name</site-name>
         <rack-name system-property="coherence.rack">rack-name</rack-name>
         <machine-name system-property="coherence.machine">machine-name
         </machine-name>
         <process-name system-property="coherence.process">JVM1
         </process-name>
         <member-name system-property="coherence.member">Storage1</member-name>
         <role-name system-property="coherence.role">CacheServer</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.

```
-Dcoherence.site=pa-1
-Dcoherence.rack=100A
-Dcoherence.machine=prod001
-Dcoherence.process=JVM1
```

```
-Dcoherence.member=C1
-Dcoherence.role=Server
```

Configuring Multicast Communication

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

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

Changing the Multicast Socket Interface

The multicast socket network interface (NIC) is automatically selected. For configurations which use multicast only for discovery, the default value is calculated using the `<discovery-address>` value that is specified as part of the `<unicast-listener>` configuration. For configurations which use multicast for both discovery and data transmission (that is, the `<multicast-threshold-percent>` is set to a value less than 100), the default value is the unicast listener interface. Using a different NIC for multicast is not a best practice and is strongly discouraged as it can lead to partial failure of the cluster and prolongs failure detection and failover.

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>
```

Specifying a Cluster's Multicast Address and Port

A multicast address and port can be specified for a cluster member. Cluster members must use the same multicast address and port to join and cluster. The default multicast address is 239.192.0.0. The default cluster port is 7574.

> **Note:**
>
> - The multicast cluster address and port may be safely shared by multiple Coherence clusters. However, clusters that are configured to use SSL cannot share a multicast address and port. In addition, all clusters must be configured to use the same IP protocol (for example, either IPv6 or IPv4).
> - The cluster port is also used by clusters that are configured to use Well Known Addresses (WKA) instead of multicast. See Using Well Known Addresses.

The Coherence default cluster port is registered with the Internet Assigned Numbers Authority (IANA) and, for most clusters, the port does not need to be changed. If a different port is required, then the recommended best practice is to select a value between 1024 and 8999; these values typically fall outside of most operating systems' ephemeral port range. Ephemeral ports can be randomly assigned to other processes resulting in Coherence not being able to bind to the port and startup will fail. Refer to the documentation for you operating system to ensure that the selected port is not within the ephemeral port range.

To specify a cluster multicast address and port, 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="coherence.clusteraddress">224.7.2.9
         </address>
         <port system-property="coherence.clusterport">3206</port>
      </multicast-listener>
   </cluster-config>
</coherence>
```

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

```
-Dcoherence.clusteraddress=224.7.2.9
-Dcoherence.clusterport=3206
```

Specifying the Multicast Time-to-Live

The time-to-live value (TTL) setting designates how far multicast 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.

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="coherence.ttl">3</time-to-live>
      </multicast-listener>
   </cluster-config>
</coherence>
```

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

```
-Dcoherence.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. Generally, there is no need to change the default join timeout value. However, if a server starts a new cluster instead of joining an existing cluster, then the join timeout value can be increased to provide additional time for the server to join the cluster.

> **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.

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>
```

> **Note:**
>
> The `<join-timeout-milliseconds>` setting will be used for both multicast and unicast communication.

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>
```

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; unicast (point-to-point) is used instead. Coherence is designed to use point-to-point communication as much as possible even when multicast is

enabled, so most application profiles do not see a substantial performance impact. See Using Well Known Addresses.

> **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 a Cluster Member's Unicast Address

Unicast communication is configured in an operational override file within the `<unicast-listener>` element node. System properties are also available to configure unicast communication when starting a cluster member.
This section includes the following topics:

- Changing the Default Unicast Address
- Changing the Default Unicast Port

Changing the Default Unicast Address

Cluster members attempt to obtain the IP to bind to using the `java.net.InetAddress.getLocalHost()` call. Coherence automatically selects a routable IP with the highest MTU for computers that have multiple IPs or NICs. If WKA is configured, then Coherence selects the IP which is routable to the IPs on the WKA list. If you do not want to use the selected NIC, then manual configuration is needed to override the default.

> **Note:**
>
> The multicast socket binds to the same interface as defined by the unicast address if multicast is used for both member discovery and data transmission. See Changing the Multicast Socket Interface.

Unicast addresses can be entered using Classless Inter-Domain Routing (CIDR) notation, which uses a subnet and mask pattern for a local IP address to bind to instead of specifying an exact IP address. CIDR simplifies configuration by allowing a single address configuration to be shared across computers on the same subnet. Each cluster member specifies the same CIDR address block and a local NIC on each computer is automatically found that matches the address pattern. For example, to specify a unicast address for multiple multi-NIC computers that are located on the same network and that will run a cluster on their `192.168.1.*` address, specify an address such as `192.168.1.0/24` and each node finds a local NIC that matches the pattern. The `/24` prefix size matches up to 256 available addresses: from 192.168.1.0 to 192.168.1.255. The `<address>` element also supports external NAT addresses that route to local addresses; however, both addresses must use the same port number.

To specify a cluster member's unicast address, edit the operational override file and add an `<address>` element that includes the unicast address. For example:

```
<cluster-config>
   <unicast-listener>
      <address system-property="coherence.localhost">192.168.1.0/24
```

```
      </address>
   </unicast-listener>
</cluster-config>
```

The `coherence.localhost` system property is used to specify the unicast address instead of using the operational override file. For example:

```
-Dcoherence.localhost=192.168.1.0/24
```

Changing the Default Unicast Port

Cluster member unicast ports are automatically assigned from the operating system's available ephemeral port range. This ensures that Coherence cannot accidentally cause port conflicts with other applications. However, if a firewall is required between cluster members (an atypical configuration), then the port must be manually configured. See Configuring Firewalls for Cluster Members.

When manually configuring a unicast port, a single port is specified. If the port is not available, then the default behavior is to select the next available port. For example, if port 9000 is configured for the port and it is not available, then the next available port is automatically selected. Automatic port adjustment can be disabled. In this case, the specified port must be available. Automatic port adjustment can also be used to specify the upper limit of the port range.

To specify a cluster member's unicast ports, edit the operational override file and add a `<port>` element that includes a port value. For example:

```
<cluster-config>
   <unicast-listener>
      <port system-property="coherence.localport">9000</port>
   </unicast-listener>
</cluster-config>
```

To disable automatic port adjustment, add a `<port-auto-adjust>` element that includes the value `false`. Or, to specify a range of ports from which ports are selected, include a port value that represents the upper limit of the port range. The following example sets a port range from 9000 to 9200:

```
<cluster-config>
   <unicast-listener>
      <port system-property="coherence.localport">9000</port>
      <port-auto-adjust system-property="coherence.localport.adjust">9200
      </port-auto-adjust>
   </unicast-listener>
</cluster-config>
```

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

```
-Dcoherence.localport=9000 -Dcoherence.localport.adjust=9200
```

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.

Both UDP and TCP protocols are used with WKA, with UDP also being used in unicast mode. In cloud environments, for example, this needs to be set up in security lists (or in other means to secure networks) on subnets used by VM instances. For additional information, see Configuring Firewalls for Cluster Members.

WKA is enabled by specifying a small subset of cluster member addresses that are able to start a cluster. The optimal number of WKA addresses varies based on the cluster size. Generally, WKA addresses should be less than 10% of the cluster. One or two WKA addresses for each switch is recommended.

WKA addresses 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 address 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 addresses are stopped. However, new cluster members are not able to join the cluster unless they themselves are hosted on a WKA address or until a cluster member that is on a WKA address is started. In this case, the senior-most member of the cluster polls the WKA address list and allows the WKA address to rejoin the existing cluster.

There are two ways to specify WKA addresses. The first method specifies a list of WKA 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.

This section includes the following topics:

- Specifying WKA Addresses
- Handling Failure to Resolve Well Known Addresses
- Specifying a WKA Address Provider

Specifying WKA Addresses

WKA addresses (IP address or DNS name) are specified using the `<address>` element. Any number of WKA addresses can be specified and a unique `id` attribute must be included for each address. The list of WKA addresses should be the same for every cluster member to ensure that different cluster members do not operate independently from the rest of the cluster. If a cluster member specifies its own address, then it can start a cluster.

> **Note:**
>
> WKA uses the cluster port. See Specifying a Cluster's Multicast Address and Port.

The following example specifies two WKA addresses:

```
<?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 id="1">Server1</address>
            <address id="2">Server2</address>
         </well-known-addresses>
      </unicast-listener>
   </cluster-config>
</coherence>
```

While either an IP address or DNS name can be used, DNS names have an additional advantage: any IP addresses that are associated with a DNS name are automatically resolved at runtime. This allows the list of WKA addresses to be stored in a DNS server and centrally managed and updated in real time. For example, if the WKA address list for a cluster that is named `cluster1` is going to be `192.168.1.1`, `192.168.1.2`, `192.168.1.3`, then a single DNS entry for hostname `cluster1` can contain those addresses and a single address named `cluster1` can be specified for the WKA address:

```
<?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>cluster1</address>
         </well-known-addresses>
      </unicast-listener>
   </cluster-config>
</coherence>
```

For networks that use Network Address Translation (NAT), you can build the WKA addresses list using external NAT addresses that route to local addresses. The Coherence network layer automatically discovers the WKA cluster members that map to the external NAT addresses and also discovers external NAT addresses for cluster members that are not on the WKA list. These learned addresses are then automatically used for other Coherence services. When using NAT addresses, the external addresses and local addresses must use the same port number.

- Using WKA System Properties

Using WKA System Properties

A single WKA address can be specified using the `coherence.wka` system properties instead of specifying the address in an operational override file. For example:

```
-Dcoherence.wka=192.168.0.100
```

If multiple WKA addresses are required, you can provide these addresses as a comma delimited list for the `coherence.wka` system property value. For example, if Coherence runs on two servers with the host names `server1.acme.com` and `server2.acme.com`, you can set the `coherence.wka` property as follows:

```
-Dcoherence.wka=server1.acme.com,server2.acme.com
```

> **Note:**
>
> You can now supply a comma separated list of hostnames in the WKA. For example: `Dcoherence.wka=192.168.0.100,192.168.0.101`.

Alternatively, you can specify multiple WKA addresses by adding additional `<address>` elements to an operational override file. Then, define a system-property attribute for each WKA address element. The attributes must include the system property names to be used to override the elements. The following example defines two addresses including system properties:

> **Note:**
>
> Defining additional system properties to specify a list of WKA addresses 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 addresses in production environments. This ensures that the same list of WKA addresses 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>
            <address id="1" system-property="coherence.wka"></address>
            <address id="2" system-property="coherence.wka2"></address>
         </well-known-addresses>
      </unicast-listener>
   </cluster-config>
</coherence>
```

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

```
-Dcoherence.wka=192.168.0.102 -Dcoherence.wka2=192.168.0.103
```

See Creating Custom System Properties.

Handling Failure to Resolve Well Known Addresses

If Coherence fails to resolve any of the well-known addresses that have been configured, then it fails to start and displays an exception. In some environments, this exception can cause issues. For example, in Kubernetes, when using a Kubernetes service name for the WKA address, there can be a delay before Pod IP addresses are mapped to the service, resulting in

a DNS look up returning no IP addresses. In this case, it is possible to configure Coherence to retry the WKA resolution.

Setting the `coherence.wka.dns.resolution.retry` system property to `true` causes Coherence to retry the WKA resolution every 10 milliseconds, for a maximum of six minutes. You can change the default timeout by setting the `coherence.wka.dns.resolution.timeout` system property to the timeout value in milliseconds.

For example, to set the timeout to three minutes:

```
-Dcoherence.wka.dns.resolution.timeout=180000
```

The default 10ms retry frequency can be changed by setting the `coherence.wka.dns.resolution.frequency` system property to a new retry value in milliseconds.

For example, to retry resolution every five seconds:

```
-Dcoherence.wka.dns.resolution.frequency=5000
```

Specifying a WKA Address Provider

A WKA address provider offers a programmatic way to define WKA addresses. 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 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 for development and unit testing.
To enable single-server mode, edit the operational override file and add 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="coherence.localhost">127.0.0.1
         </address>
      </unicast-listener>
   </cluster-config>
</coherence>
```

The `coherence.localhost` system property is used to enable single-server mode instead of using the operational override file. For example:

```
-Dcoherence.localhost=127.0.0.1
```

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 identifies both process failures (TcpRing component) and hardware failure (IpMonitor component). Process failure is detected using a ring of TCP connections opened on the same port that is used for cluster communication. Each cluster member issues a unicast heartbeat, and the most senior cluster member issues the cluster heartbeat, which is a broadcast message. Hardware failure is detected using the Java `InetAddress.isReachable` method which either issues a trace ICMP ping, or a pseudo ping and uses TCP port 7. Death detection is enabled by default and is configured within the `<tcp-ring-listener>` element.

This section includes the following topics:

- 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. These include changing the amount of attempts and time before determining that a computer that is hosting cluster members has become unreachable. The default values are 3 attempts and 5

seconds allowing for a network disconnect of up to 15 seconds. 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="coherence.ipmonitor.pingtimeout">
            25s</ip-timeout>
         <ip-attempts>5</ip-attempts>
         <listen-backlog>10</listen-backlog>
      </tcp-ring-listener>
   </cluster-config>
</coherence>
```

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

```
-Dcoherence.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 emitted 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 packets. By default, the timeout interval is set to 5 minutes. See Changing the Packet Resend Timeout.

> **Note:**
>
> Using the packet publisher's resend timeout to detect a failed cluster member is error prone 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.
This section includes the following topics:

- Specifying a Cluster Member's Priority
- Specifying Communication Thread Priorities
- Specifying Thread Priorities for Services

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 10 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="coherence.priority">1</priority>
      </member-identity>
   </cluster-config>
</coherence>
```

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

```
-Dcoherence.priority=1
```

Specifying Communication Thread Priorities

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.

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 10 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>
```

Specifying Thread Priorities for Services

Cluster services support thread priority. The priority is used as the basis for determining Java thread execution importance and indicates which threads of a service are considered critical. There are three types of threads that can have a priority: service threads, event dispatcher threads, and worker threads. The default setup gives service and event dispatcher threads precedence followed by worker threads.

Thread priorities for services can be changed for all services in a cluster by overriding the `<service>` element in an operational override file. However, a better practice is to configure

thread priorities for a service instance within in a cache configuration file when defining a cache scheme. See Defining Cache Schemes. Use the `<service-priority>`, `<event-dispatcher-priority>`, and `<worker-priority>` subelements, respectively and enter a value between 1 and 10 where 10 is the highest priority. For example:

```
...
<distributed-scheme>
   <scheme-name>distributed</scheme-name>
   <service-name>MyDistributedService</service-name>
   <service-priority>10</service-priority>
   <event-dispatcher-priority>10</event-dispatcher-priority>
   <worker-priority>5</worker-priority>
   ...
</distributed-scheme>
...
```

Configuring Firewalls for Cluster Members

Firewalls are not typically setup between Coherence cluster members because it is assumed that all cluster members communicate behind the firewall in a secured environment. Multiple ports must be opened in a firewall if you intend to secure communication between cluster members. Note that firewalls tend to be used less and less in cloud environments, where network security is achieved at the infrastructure level (seclists).

If a solution requires the use of a firewall between cluster members, then ensure the following:

- The cluster port (7574 by default) is open for both UDP and TCP for both multicast and unicast configurations.
- TCP port 7 is open for the Coherence TcpRing/IpMonitor death detection feature.
- The unicast port range is open for both UDP and TCP traffic. Ensure that the unicast listen port range is explicitly set rather then relying upon a system assigned ephemeral port. See Changing the Default Unicast Port.

8
Starting and Stopping Coherence

You can start and stop the cache servers and cache clients that are part of a cluster. This chapter includes the following sections:

- Using the Bootstrap API
  Coherence has a simple bootstrap API that enables you to configure and start a Coherence application by building a `com.tangosol.net.Coherence` instance and starting this instance.
- Using the com.tangosol.net.Coherence Class
  Cache servers are cluster members that are responsible for storing cached data.
- Using the Legacy CacheFactory Client
- Stopping Cluster Members
- Performing a Rolling Restart

Using the Bootstrap API

Coherence has a simple bootstrap API that enables you to configure and start a Coherence application by building a `com.tangosol.net.Coherence` instance and starting this instance.

The Coherence instance provides access to one or more `com.tangosol.net.Session` instances. A `com.tangosol.net.Session` gives access to Coherence clustered resources, such as `NamedMap`, `NamedCache`, `NamedTopic`, and so on. Sessions can be of different types. For example, a session can be:

- related to a `ConfigurableCacheFactory`.
- configured from a configuration file.
- a client-side gRPC session.

Example 8-1 Application Bootstrap Code

```
import com.tangosol.net.Coherence;
import com.tangosol.net.CoherenceConfiguration;
import com.tangosol.net.SessionConfiguration;

public class Main
    {
    public static void main(String[] args)
        {
        // Create a session configuration
        SessionConfiguration session = SessionConfiguration.builder()
                .named("Carts")
                .withConfigUri("cache-config.xml")
                .build();

        // Create a Coherence instance configuration
        CoherenceConfiguration cfg = CoherenceConfiguration.builder()
                .withSession(SessionConfiguration.defaultSession())
```

```
                .withSession(session)
                .build();

        // Create the Coherence instance from the configuration
        Coherence coherence = Coherence.clusterMember(cfg);

        // Start Coherence
        coherence.start();
        }
    }
```

In this example:

- The 'Create a session configuration' part creates `SessionConfiguration` from the `cache-config.xml` configuration file with the `Session` name `Carts`.
- The 'Create a Coherence instance configuration' part creates `CoherenceConfiguration` to configure the `Coherence` instance. This configuration contains the `Carts` session configuration.
- The 'Create the Coherence instance from the configuration' part creates the `Coherence` cluster member instance from `CoherenceConfiguration`.
- The 'Start Coherence' part starts the `Coherence` instance.

This section includes the following topics:

Running a Coherence Server

The `com.tangosol.net.Coherence` contains a `main` method that you can use to run a Coherence server. This method is a more powerful alternative to the legacy `DefaultCacheServer` class.

```
$ java -cp coherence.jar com.tangosol.net.Coherence
```

Without any other configuration, the default `Coherence` instance started using the above command runs a server that is identical to the server that is run using `DefaultCacheServer`.

Session Configurations

You can create a create a session configuration named `SessionConfiguration` by using the `SessionConfiguration` builder. See the example in Using the Bootstrap API.

This section includes the following topics:

About the Default Session

When running Coherence, if you have not specified any configuration, the default configuration file is used to configure Coherence. This behavior continues to apply to the bootstrap API.

If you start a `Coherence` instance without specifying any session configurations, it creates a single default `Session`. This default `Session` wraps the cache factory interface `ConfigurableCacheFactory` that is created from the default configuration file. See Interface ConfigurableCacheFactory. The default file name of the default configuration file is `coherence-cache-config.xml` unless the name is overridden with the `coherence.cacheconfig` system property.

When creating a Coherence configuration instance named `CoherenceConfiguration`, you can add the default session by using the `SessionConfiguration.defaultSession()` helper method. This method returns a `SessionConfiguration` that is configured to create the default session `Session`.

In the following example, the default session configuration is specifically added to `CoherenceConfiguration`:

```
CoherenceConfiguration cfg = CoherenceConfiguration.builder()
        .withSession(SessionConfiguration.defaultSession())
        .build();
```

Naming a Session

All sessions have a name that must be unique within the application. If you have not specified a name when the `SessionConfiguration` is built, the default name of `$Default$` will be used. A `Coherence` instance fails to start if duplicate `Session` names exist.

Examples:

- This configuration will have the default name (`$Default$`):

```
SessionConfiguration session = SessionConfiguration.builder()
        .build();
```

- This configuration will have the name `Test`:

```
SessionConfiguration session = SessionConfiguration.builder()
        .name("Test")
        .build();
```

Specifying the Session Configuration URI

The most common type of session is a wrapper around `ConfigurableCacheFactory`. When using the `SessionConfiguration` builder, you can specify the configuration file URI by using the `withConfigUri()` method. This method accepts a string value for specifying the location of the configuration file.

The following example uses the configuration file named `cache-config.xml`:

```
SessionConfiguration session = SessionConfiguration.builder()
        .withConfigUri("cache-config.xml")
        .build();
```

If you do not specify a configuration URI, the default value will be used. The default value is `coherence-cache-config.xml` unless this value is overridden with the `coherence.cacheconfig` system property.

Adding Session Event Interceptors

Coherence provides many types of events. For example, life-cycle events for Coherence itself, cache life-cycle events, cache entry events, partition events, and so on. These events can be listened to by implementing an `EventInterceptor` that receives specific types of event. Event interceptors can be registered with a `Session` as part of its configuration.

For example, suppose the application has an interceptor class called `CacheInterceptor` that listens to the `CacheLifecycleEvent` event when caches get created or destroyed. You can add this interceptor to the session as shown below:

```
SessionConfiguration session = SessionConfiguration.builder()
        .withInterceptor(new CacheInterceptor())
        .build();
```

The interceptor will receive cache life-cycle events for all caches that are created using the session.

Scoping a Session Configuration

Scope is a concept that helps you scope services and isolate them from other services with the same name. For example, you can have multiple `ConfigurableCacheFactory` instances loaded from the same XML configuration file but with different scope names. Scoping ensures that each `ConfigurableCacheFactory` instance has its own services in the cluster.

Unless you require multiple sessions, a scope will not generally be used in a configuration.

You can configure a scope for a session using the configuration's `withScopeName()` method, as shown in the following example:

```
SessionConfiguration session = SessionConfiguration.builder()
        .withScopeName("Test")
        .build();
```

The session (and any `ConfigurableCacheFactory` it wraps) created from the above configuration will have a scope name of `Test`.

You can also set a scope name in the `<defaults>` section of the XML configuration file. An example of `scoped-configuration.xml` file is shown below:

```
<?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>
    <scope-name>Test</scope-name>
  </defaults>
```

A `ConfigurableCacheFactory` instance created from this XML file, and any `Session` that wraps it, will have a scope name of `Test`.

Note:

When using the bootstrap API, any scope name specifically configured in `SessionConfiguration` (that is not the default scope name) overrides the scope name in the XML file.

For example, the following scenarios show the different ways of defining the scope name using the `scoped-configuration.xml` file (see the above example):

- In this case, the scope name will be `Foo` because the scope name has been explicitly set in `SessionConfiguration`:

```
SessionConfiguration session = SessionConfiguration.builder()
        .withConfigUri("scoped-configuration.xml")
        .withScopeName("Foo")
        .build();
```

- In this case, the scope name will be `Foo` because, although no scope name has been explicitly set in `SessionConfiguration`, the name has been set to `Foo`. Therefore, the scope name will default to `Foo`.

```
SessionConfiguration session = SessionConfiguration.builder()
        .named("Foo")
        .withConfigUri("scoped-configuration.xml")
        .build();
```

- In this case, the scope name will be `Test` as no scope name or session name has been explicitly set in `SessionConfiguration`. Therefore, the scope name of `Test` will be used from the XML configuration.

```
SessionConfiguration session = SessionConfiguration.builder()
        .withConfigUri("scoped-configuration.xml")
        .build();
```

- In this case, the scope name will be `Test` as the session name has been set to `Foo`. However, the scope name has been explicitly set to the default scope name using the constant `Coherence.DEFAULT_SCOPE`. Therefore, the scope name of `Test` will be used from the XML configuration.

```
SessionConfiguration session = SessionConfiguration.builder()
        .named("Foo")
        .withScopeName(Coherence.DEFAULT_SCOPE)
        .withConfigUri("scoped-configuration.xml")
        .build();
```

Configuring a Coherence Instance

You can start a Coherence application by creating a Coherence instance from `CoherenceConfiguration`. An instance of `CoherenceConfiguration` is created using the builder. For example:

```
CoherenceConfiguration cfg = CoherenceConfiguration.builder()
        .build();
```

This section includes the following topics:

- Adding Sessions
- Configuring a Session for Auto-Discovery
- Naming Coherence Instances
- Adding Global Event Interceptors

Adding Sessions

A `Coherence` instance manages one or more `Session` instances. You can add these session instances to `CoherenceConfiguration` by adding the `SessionConfiguration` instances to the builder.

If you do not add any sessions to the builder, the `Coherence` instance runs a single `Session` that uses the default configuration file.

```
CoherenceConfiguration cfg = CoherenceConfiguration.builder()
        .build();
```

The above configuration configures a `Coherence` instance with the default name and with a single `Session` that uses the default configuration file.

You can also explicitly add the default session to `CoherenceConfiguration`:

```
CoherenceConfiguration cfg = CoherenceConfiguration.builder()
        .withSession(SessionConfiguration.defaultSession())
        .build();
```

You may also add other session configurations to `CoherenceConfiguration`:

```
SessionConfiguration session = SessionConfiguration.builder()
        .named("Carts")
        .withConfigUri("cache-config.xml")
        .build();

CoherenceConfiguration cfg = CoherenceConfiguration.builder()
        .withSession(session)
        .build();
```

While there is no limit to the number of sessions you can configure, the majority of applications require only a single session, which is most likely the default session.

Configuring a Session for Auto-Discovery

You can configure a `CoherenceConfiguration` to automatically discover the `SessionConfiguration` instances. These instance are discovered using the Java `ServiceLoader`. Any instances of `SessionConfiguration` or `SessionConfiguration.Provider` that are configured as services in the `META-INF/services/` files, will be loaded.

This feature is useful if you are building modular applications where you want to include the functionality in a separate application module that uses its own `Session`. The `SessionConfiguration` instance for the module is made discoverable by the Java `ServiceLoader`. When the module's jar file is on the classpath, the `Session` gets created and the module's functionality becomes available to the application.

For example:

```
CoherenceConfiguration cfg = CoherenceConfiguration.builder()
        .discoverSessions()
        .build();
```

In this example, the call to `discoverSessions()` loads the `SessionConfiguration` instances that have been discovered by the Java `ServiceLoader`.

Naming Coherence Instances

Each `Coherence` instance must be uniquely named. You can specify a name by using the `named()` method on the builder. If you do not specify a name, then the default name of `$Default$` will be used.

In the majority of use-cases, an application requires only a single `Coherence` instance. Therefore, there will be no requirement to specify a name.

In the following code, the configuration creates a `Coherence` instance with the name `Carts`.

```
CoherenceConfiguration cfg = CoherenceConfiguration.builder()
        .named("Carts")
        .build();
```

Adding Global Event Interceptors

You can add event interceptors to a `SessionConfiguration` instance to receive events for a session. See Adding Session Event Interceptors. Event interceptors can also be added to the `Coherence` instance to receive events for all `Session` instances managed by that `Coherence` instance.

For example, if you reuse the `CacheInterceptor` class for caches in all sessions, then the interceptor receives events for both the default session and the `Certs` session:

```
SessionConfiguration cartsSession = SessionConfiguration.builder()
         .named("Carts")
         .withConfigUri("cache-config.xml")
         .build();

CoherenceConfiguration cfg = CoherenceConfiguration.builder()
```

```
        .withSession(SessionConfiguration.defaultSession())
        .withSession(cartsSession)
        .withInterceptor(new CacheInterceptor())
        .build();
```

Creating a Coherence Instance

You can use the `CoherenceConfiguration` instance to create a `Coherence` instance.

You can create a `Coherence` instance in one of the following two modes:

- cluster member
- client

The mode chosen affects how some types of `Session` are created and whether auto-start services have started.

As the name suggests, a "cluster member" is a `Coherence` instance that expects to start or join a Coherence cluster. In a cluster member, any `Session` that wraps a `ConfigurableCacheFactory` has its services auto-started and monitored (this is the same behavior as seen when you use the legacy `DefaultCacheServer` to start a server).

A "client" `Coherence` instance is typically not a cluster member, it is a Coherence*Extend or a gRPC client. As such, `Session` instances that wrap a `ConfigurableCacheFactory` are not auto-started. Instead, they start on demand as resources such as maps, caches, or topics are requested from them.

The `com.tangosol.net.Coherence` class has static factory methods to create `Coherence` instances in different modes.

Examples:

- To create a `Coherence` instance that is a cluster member, use the `Coherence.clusterMember` method, as shown below:

```
CoherenceConfiguration cfg = CoherenceConfiguration.builder()
        .build();

Coherence coherence = Coherence.clusterMember(cfg);
```

- To create a `Coherence` instance that is a client, use the `Coherence.client` method, as shown below:

```
CoherenceConfiguration cfg = CoherenceConfiguration.builder()
        .build();

Coherence coherence = Coherence.client(cfg);
```

This section includes the following topic:

- Creating a Default Coherence Instance

Creating a Default Coherence Instance

You can create a `Coherence` instance without specifying any configuration.

```
Coherence coherence = Coherence.clusterMember();


Coherence coherence = Coherence.client();
```

In the above examples, the `Coherence` instance will have the default `Session` and any discovered sessions.

Starting Coherence

A `Coherence` instance must be started to start all the sessions that the `Coherence` instance is managing. You can start the `Coherence` instance by calling the `start()` method, as shown below:

```
Coherence coherence = Coherence.clusterMember(cfg);
coherence.start();
```

Obtaining a Coherence Instance

To avoid having to pass around the instance of `Coherence` that was used to bootstrap an application, the `Coherence` class has some static methods that make it simple to retrieve an instance.

If only a single instance of `Coherence` is being used in an application (which is true for most use-cases), use the `getInstance()` method:

```
Coherence coherence = Coherence.getInstance();
```

You can also retrieve an instance by name by configuring it as shown below:

```
CoherenceConfiguration cfg = CoherenceConfiguration.builder()
        .named("Carts")
        .build();

Coherence.create(cfg);
```

And then retrieve the instace:

```
Coherence coherence = Coherence.getInstance("Carts");
```

Ensuring Coherence Has Started

If an application code needs to ensure that a `Coherence` instance has started before doing work, use the `whenStarted()` method. This method obtains a `CompletableFuture` that will be completed when the `Coherence` instance starts.

For example:

```
Coherence                coherence = Coherence.getInstance("Carts");
CompletableFuture<Void> future    = coherence.whenStarted();

future.join();
```

There is also a corresponding `whenStopped()` method that returns a future that will be completed when the `Coherence` instance stops.

Adding Coherence Lifecycle Interceptors

Besides using the future methods as described in Ensuring Coherence Has Started, you can add an `EventInterceptor` to the configuration of a `Coherence` instance to receive life-cycle events.

Following is an example interceptor that implements `Coherence.LifecycleListener`.

```
public class MyInterceptor implements Coherence.LifecycleListener {
    public void onEvent(CoherenceLifecycleEvent event) {
        // process event
    }
}
```

You can add the interceptor to the following configuration:

```
CoherenceConfiguration cfg = CoherenceConfiguration.builder()
        .withSession(SessionConfiguration.defaultSession())
        .withInterceptor(new MyInterceptor())
        .build();
```

When you start or stop a `Coherence` instance created from this configuration, the `MyInterceptor` instance starts receiving the life-cycle events.

Using the com.tangosol.net.Coherence Class

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.

This section includes the following topics:

- Overview of the Coherence Class
- Starting Cache Servers From the Command Line
- Starting Cache Servers Programmatically
- Coherence Lifecycle Listeners

Overview of the Coherence Class

The `com.tangosol.net.Coherence` class is used to start a cache server. A cache server can be started from the command line or can be started programmatically. The following arguments are used when starting a cache server:

- The `--version` argument will display the version of Coherence being used. If this is the *only* parameter passed to the Coherence main method, then the version will be displayed and Coherence will exit.
- The name of a cache configuration file that is found on the classpath or the path to a Grid ARchive (GAR). If both are provided, then the GAR takes precedence. A GAR includes the artifacts that comprise a Coherence application and adheres to a specific directory structure. A GAR can be left as a directory or can be archived with a `.gar` extension. See Building a Coherence GAR Module in *Administering Oracle Coherence*.
- An optional application name for the GAR. If no name is provided, then the archive name is used (the directory name or the file name without the `.gar` extension). The name provides an application scope that is used to separate applications on a cluster.

Starting Cache Servers From the Command Line

Cache servers are often started from the command line using the Coherence class main method. Using the various lifecycle listeners in Coherence to initialize application code means that there is not typically a need for any application specific main class. Use the Java `-cp` option to indicate the location of the `coherence.jar` file, any application `.jar` files, and the location where any Coherence configuration files are located. The location of the configuration files must precede the `coherence.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 Understanding Configuration.

The following example starts a cache server member, uses any configuration files that are placed in the *COHERENCE_HOME*`\config` directory.

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

The following example starts a cache server member and uses the Coherence application artifacts that are packaged in the `MyGar.gar` file. The default name (`MyGAR`) is used as the application name.

```
java -server -Xms512m -Xmx512m -cp
COHERENCE_HOME\config;COHERENCE_HOME\lib\coherence.jar com.tangosol.net.Coherence
D:\example\MyGAR.gar
```

> **Note:**
>
> The cache configuration file that is packaged in a GAR file takes precedence over a cache configuration file that is located on the classpath.

The *COHERENCE_HOME*`\bin\cache-server` script, from the Oracle Coherence commercial version installer, is provided as a convenience and example script to start a cache server instance. The script is available for both Windows (`cache-server.cmd`) and UNIX-based platforms (`cache-server.sh`). The script sets up a basic environment and then runs the `Coherence` class. The scripts are typically modified as required for a particular application's requirements.

> **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.

Lastly, a cache server can be started on the command line by using the `java -jar` command with the `coherence.jar` library. Cache servers are typically started this way for testing and demonstration purposes. For example:

```
java -jar COHERENCE_HOME\lib\coherence.jar
```

Starting Cache Servers Programmatically

Coherence cluster members that are started using the Bootstrap API, or by running the `com.tangosol.net.Coherence.main()` method, can be stopped by calling the static `com.tangosol.net.Coherence.closeAll()` method.

The legacy `DefaultCacheServer` class provides the following 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.

Coherence Lifecycle Listeners

When starting Coherence cluster members or clients, Coherence offers various lifecycle events that can be used to trigger application specific logic before, during, and after the Coherence start-up lifecycle. Using lifecycle event listeners can often be used in place of a custom application main class to perform application specific start-up and shutdown logic.

Lifecycle events are available for the following instances:

- `Coherence` instances
- `Session` instances
- `ConfigurableCacheFactory` instances
- `DefaultCacheServer` instances

For more information about descriptions of these events and listeners, see Coherence Lifecycle Listeners.

Using the Legacy CacheFactory Client

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.
This section includes the following topics:

- 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 `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 -
Dcoherence.distributed.localstorage=false com.MyApp
```

Using the CacheFactory Class to Start a Cache Client

Applications that use the `com.tangosol.net.CacheFactory` class to get an instance of a cache become cluster members and are considered cache clients. 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 `coherence.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 Understanding Configuration.

The following example starts an application that is a cache client, uses any configuration files that are placed in the *COHERENCE_HOME*`\config` directory, and disables storage on the member.

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

The *COHERENCE_HOME*`\bin\coherence` script is provided for testing purposes and can start a cache client instance. The script is available for both Windows (`coherence.cmd`) and UNIX-based platforms (`coherence.sh`). The script sets up a basic environment, sets storage to be disabled, and then runs the `CacheFactory` class, which returns a prompt. The prompt is used to enter commands for interacting with a cache and a cluster. The scripts are typically modified as required for a particular cluster. The class can also be started directly from the command line instead of using the script. For example:

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

If a Coherence application is packaged as a GAR, the GAR can be loaded by the `CacheFactory` instance using the `server` command at the prompt after the client member starts.

```
server [<path-to-gar>] [<app-name>]
```

The following example loads the Coherence application artifacts that are packaged in the `MyGar.gar` file. The default name (`MyGAR`) is used as the application name.

```
Map (?) server D:\example\MyGAR.gar
```

Stopping Cluster Members

You can stop cluster members from the command line or programmatically.
This section includes the following topics:

- Prerequisites for Stopping All Cluster Members
- Stopping Cluster Members From the Command Line
- Stopping Cache Servers Programmatically

Prerequisites for Stopping All Cluster Members

To ensure there is no data loss during a controlled shutdown, you can leverage the service suspend feature. A service is considered suspended only after all the data is fully written, including active persistence mode, asynchronous persistence tasks, entries in the write-behind queue of a read-write backing map, and other asynchronous operations. Outstanding operations are completed and no new operations are allowed against the suspended services.

Thus, for a controlled complete shutdown of a cluster, Oracle recommends to execute the Coherence ClusterMBean operation `suspendService("Cluster")`, which shuts down all services gracefully before shutting down the cluster members.

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.

- `force` - Perform a hard-stop on the node by calling `Cluster.stop()`.
- `graceful` - (Default) Perform a normal shutdown by calling `Cluster.shutdown()`.
- `true` - Same as `force`.
- `false` - Same as `none`.

The `coherence.shutdown.timeout` system property configures the duration to wait for shutdown to complete before timing out. The default value is 2 minutes. If a Coherence application uses persistence, write-behind cache store, or any other asynchronous operation that takes longer than 2 minutes, then it is necessary to configure the timeout to be longer to allow the pending asynchronous operations to complete. The system property value is configured as a time duration such as "3s" for 3 seconds, "5m" for 5 minutes, or "1hr" for 1 hour.

When the time duration to complete graceful shutdown exceeds the `coherence.shutdown.timeout` time, the JVM process is considered hung and the JVM process terminates abruptly using halt. There exists a chance that not all outstanding asynchronous operations completed when the shutdown times out. Therefore, it is important to ensure that `coherence.shutdown.timeout` is configured to a time duration that is sufficiently long for the number of outstanding asynchronous operations an application environment may have.

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="coherence.shutdownhook">none</enabled>
      </shutdown-listener>
   </cluster-config>
</coherence>
```

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

```
-Dcoherence.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.

Performing a Rolling Restart

A rolling restart is a technique for restarting cache servers in a cluster that ensures no data is lost during the restart. A rolling restart allows the data on a cache server to be redistributed to other cache servers on the cluster while the cache server is restarted. Each cache server in the cluster can be restarted in turn to effectively restart the whole cluster.
Rolling restarts are commonly performed when a cache server or its host computer must be updated or when upgrading a cache server to a new patch set release or patch set update release. However, the technique can also be used whenever you want to restart a cache server that is currently managing a portion of cached data.

> **Note:**
>
> When upgrading a cluster, a rolling restart can only be used to upgrade patch set releases or patch set update releases, but not major or minor releases. See Release Number Format in *Administering Oracle Fusion Middleware*.

This section includes the following topics:

- Prerequisites for a Rolling Restart
- Considerations for Application Upgrades
- Restarting Cache Servers for a Rolling Restart
- Persistence Roll Back After or During a Rolling Restart

Prerequisites for a Rolling Restart

A rolling restart requires initial considerations and setup before you restart a cluster. A rolling restart cannot be performed on a cluster that does not meet the following prerequisites:

- When changing the JDK version, you can do a rolling restart only between patch versions of the same major JDK version, for example, patches within JDK 17, such as 17.07 to 17.0.8. However, you cannot perform a rolling upgrade if you move between major JDK versions, for example, from JDK 11 to JDK 17.
- Do not change the serialization type, for example, moving from Java to POF serialization or conversely because it is not supported.
- If you add fields to the cache classes and they do not support evolvability (see Class Versioning and Evolution), then you cannot perform a rolling upgrade.
- If you are performing an application upgrade, to determine if your application upgrade can be done using a rolling restart, see Considerations for Application Upgrades.
- If you are changing any operational or cache configuration during the upgrade, to determine if it is a supported change during a rolling upgrade, see the documentation on the operational or cache configuration elements. If it is not able to be changed, then note similar to "You cannot change this element during a rolling restart." will be present.
- The cache servers in a cluster must provide enough capacity to handle the shutdown of a single cache server (*n* minus 1 where *n* is the number of cache servers in the cluster). An out-of-memory exception or data eviction can occur during a redistribution of data if the

cache servers are running at capacity. See Cache Size Calculation Recommendations in *Administering Oracle Coherence*.

- Remote JMX management must be enabled on all cache servers and at least two cache servers must contain an operational MBean server. Ensure that you can connect to the MBean servers using an MBean browser such as JConsole. See Using JMX to Manage Oracle Coherence in *Managing Oracle Coherence*.
- If your environment is using static lambda serialization, then it is recommended that you configure dynamic lambda serialization mode before attempting a rolling restart. See Considerations for a Rolling Upgrade for details.

If a cache service is configured to use asynchronous backups, use the `shutdown` method to perform an orderly shut down instead of the `stop` method or `kill -9`. Otherwise, a member may shutdown before asynchronous backups are complete. The `shutdown` method guarantees that all updates are complete.

If using persistence, while a rolling restart is agnostic to the format of the data stored on disk, there are scenarios where you can take precautionary steps to ensure that there is a 'savepoint' to which the cluster can be rolled back if faced with a catastrophic event. On-disk persisted files may become unreadable when going from a later to an earlier version.

Therefore, Oracle recommends you to perform a persistent snapshot of the relevant services before the roll. See Using Snapshots to Persist a Cache Service. This can be performed while suspending the service (to ensure global consistency) or not based on the tolerance of the application. The snapshot offers a 'savepoint' to which the cluster can be rolled back if an issue occurs during the roll.

> **Note:**
>
> Coherence may change the format of the data saved on disk. When rolling from a version that includes a change in the storage format, Oracle strongly recommends creating a snapshot before roll/upgrade.

Considerations for Application Upgrades

If you want to carry out a rolling restart to upgrade or change your application, you need to make sure that the changes you are performing are supported for rolling restarts. Depending on the features and functionality you use in your application, see the following topics for more information:

- Considerations for Building Upgradable Coherence Applications
- Federation Upgrade Considerations
- Persistence Upgrade Considerations

Restarting Cache Servers for a Rolling Restart

Use these instructions to restart a cache server. If you are restarting the host computer, then make sure all cache server processes are shutdown before shutting down the computer.

> **Note:**
>
> The following instructions assume that none of the cache servers share the same Coherence JAR or *ORACLE_HOME*. If some of the cache servers share the same Coherence JAR or *ORACLE_HOME*, then treat the servers as a logical unit and perform the steps on each of the servers at the same time.

To restart a cache server:

1. Connect to a Coherence MBean server using an MBean browser. Ensure that the MBean server is not hosted on the cache server that is being restarted.
2. From the Coherence Service MBean, select a cluster service that corresponds to a cache that is configured in the cache configuration file.
3. Check the `StatusHA` attribute for any cluster member to ensure that the attribute's value is `MACHINE-SAFE`. The `MACHINE-SAFE` state indicates that all the cache servers running on any given computer could be stopped without data loss. If the attribute is not `MACHINE-SAFE`, then additional cache servers, possibly on different computers, must be started before performing a restart. See ServiceMBean in *Managing Oracle Coherence*.
4. Shutdown the cache server.
5. From the MBean browser, recheck the `StatusHA` attribute and wait for the state to return to `MACHINE-SAFE`.
6. Restart the cache server.
7. From the MBean browser, recheck the `StatusHA` attribute and wait for the state to return to `MACHINE-SAFE`.
8. Repeat steps 4 to 7 for additional cache servers that are to be restarted.

> **Note:**
>
> If you have enabled management over REST, then you also can use the Coherence CLI to monitor rolling restarts. For more information, see Rolling Restarts.

Persistence Roll Back After or During a Rolling Restart

If an issue is observed during the rolling restart, it is advisable to roll back the entire cluster. You can roll back the cluster by rolling the nodes with the new version of the application (and/or Coherence) back to their prior versions, or in more severe cases, a complete restart of the cluster.

If the new version being migrated to includes a Coherence patch that modifies the storage format (as highlighted in the Oracle Coherence Release Notes) the persistent stores could be in a mixed state of the old and new format.

During a full cluster restart, the mixed state can result in a failure to recover, causing the failed stores being moved into the trash. At this point, you can choose the `forceRecovery` operation of the `PersistenceManagerMBean` (in JConsole, for example, `Coherence/Persistence/<servicename>/PersistenceCoordinator`) to force Coherence to reinstate empty

versions of those partitions that could not be recovered. If you have created a snapshot of the relevant services, it is possible to recover the snapshot, reverting the state of those services.

9

Dynamically Managing Cluster Membership

You can programmatically manage cluster and service members and listen to member and service events.
This chapter includes the following sections:

Overview of 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 membership 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.

Using the Cluster and Service Objects

Applications can use local representations of a cluster and cache service to discover important information about a cluster. From any cache, an 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:

```
...
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.

Using the Member Object

Applications can use several APIs to discover the members that are part of a cluster, the local member, and the members in a service.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();
```

Listening to Member Events

Applications can listen to cluster and service events to determine membership changes. As of Coherence 14c (14.1.2.0.0), remote client membership changes are sent by proxy to all active services configured on the proxy when a remote client joins and leaves the proxy. Applications must implement the `MemberListener` interface (see Example 9-1). The listener class is then added on a service by either using the `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 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 9-1 demonstrates a `MemberListener` implementation that prints out all the membership events that it receives:

Example 9-1 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. To distinguish between a cluster member and a remote member joining and leaving the service, see the usage of `Member.isRemoteClient()` in Example 9-3. If your environment has both cluster members and remote clients using the same service, it might be necessary to take different actions based on the member type. Also, note that since a remote member is not considered a cluster member, these events are not delivered by the proxy to the cluster service.

> **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 9-2 illustrates how information encapsulated in a `MemberEvent` object can be used.

Example 9-2 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)
        {
        }
    }
```

Example 9-3 illustrates how to identify a remote client `MEMBER_JOINED` or `MEMBER_LEFT`.

Example 9-3 Using MemberEvent Objects Member Information to Identify a Remote Client

```
public class RemoteClientMemberListener
        extends Base
        implements MemberListener
    {
    public void memberJoined(MemberEvent evt)
        {
        Member member = evt.getMember();
        if (member.isRemoteClient())
            {
            out("remote client joined the service.  id=" + member.getUuid());
            }
        }

    public void memberLeaving(MemberEvent evt)
        {
        // Not generated for a remote client.
        }

    public void memberLeft(MemberEvent evt)
        {
        Member member = evt.getMember();
```

```
        if (member.isRemoteClient())
            {
            out("remote client left the service.  id=" + member.getUuid());
            }
        }
    }
```

For the above listener, it conditionally processes remote client membership `MEMBER_JOINED` and `MEMBER_LEFT` event when a remote client joins and leaves a proxy. The proxy sends a membership event to all services running on the proxy.

> **Note:**
>
> If the proxy is terminated gracefully or abnormally, a `MEMBERED_LEFT` `MemberEvent` is not sent. It is expected that the remote client will rejoin on another proxy. Therefore, no remote client `MEMBER_LEFT` processing should take place.

10

Tuning TCMP Behavior

You can change the default TCMP settings to increase throughput or better utilize network resources.
This chapter includes the following sections:

- Overview of TCMP Data Transmission
- Throttling Data Transmission
- Bundling Packets to Reduce Load
- Changing Packet Retransmission Behavior
- Configuring the Size of the Packet Buffers
- Adjusting the Maximum Size of a Packet
- Changing the Packet Speaker Volume Threshold
- Configuring the Incoming Message Handler
- Changing the TCMP Socket Provider Implementation
- Changing Transport Protocols
  Coherence uses a TCP/IP message bus for reliable point-to-point communication between data service members. Additional transport protocol implementations are available and can be specified as required.
- Enabling CRC Validation for TMB
  TCP/IP Message Bus (TMB) includes a cyclic redundancy check (CRC) implementation that validates network messages that are sent on the wire. CRC is commonly used to detect and handle message corruption in networks.

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 Performing a Network Performance Test in *Administering Oracle Coherence*.

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 Operational Configuration Elements.

See also Understanding TCMP.

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.
This section includes the following topics:

- 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.

This section includes the following topics:

- 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.

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 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).
This section includes the following topics:

- 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. See Adjusting the Maximum Size of a Packet.
- `<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 Socket Buffer Sizes in *Administering Oracle Coherence*.

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 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 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 Datagram size (Microsoft Windows) in *Administering Oracle Coherence*.

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 are sent and received on the unicast and multicast sockets.

  This value can be larger or smaller than 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. A default value is automatically calculated based on the local nodes MTU. An MTU of 1500 is used if the MTU cannot be obtained and is adjusted for the packet headers (1468 for UDP and 1448 for TCP).

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.

> **Note:**
>
> The packet speaker is not used for TCMP/TMB, which is the default protocol for data communication.

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. 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>
         <enabled>true</enabled>
         <volume-threshold>
            <minimum-packets>32</minimum-packets>
         </volume-threshold>
      </packet-speaker>
   </cluster-config>
</coherence>
```

Configuring the Incoming Message Handler

The incoming message handler assembles 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.
This section includes the following topics:

- Changing the Time Variance
- Disabling Negative Acknowledgments

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.

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>
```

Changing the TCMP Socket Provider Implementation

Coherence uses a combination of UDP/IP multicast and UDP/IP unicast for TCMP communication between cluster service members. Additional socket provider implementations are available and and can be specified as required. Socket providers for use by TCMP are configured for the unicast listener within the `<unicast-listener>` element.
This section includes the following topics:

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="coherence.socketprovider">tcp
         </socket-provider>
      </unicast-listener>
   </cluster-config>
</coherence>
```

The `coherence.socketprovider` system property is used to specify the socket provider instead of using the operational override file. For example:

```
-Dcoherence.socketprovider=tcp
```

Using the SDP Socket Provider

The SDP socket provider is a socket provider which, whenever possible, produces SDP-based sockets provided that the JVM and underlying network stack supports SDP. This socket provider creates `DatagramSocket` instances which are backed by SDP. When used with the WKA feature (mulitcast disabled), TCMP functions entirely over SDP without the need for UDP.

> **Note:**
>
> if this socket provider is used without the WKA feature (multicast enabled), SDP is used for all unicast communications; while, multicast is utilized for group based communications.

To specify the SDP socket provider, edit the operational override file and add a `<socket-provider>` element that includes the `sdp` 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="coherence.socketprovider">sdp
         </socket-provider>
      </unicast-listener>
   </cluster-config>
</coherence>
```

The `coherence.socketprovider` system property is used to specify the socket provider instead of using the operational override file. For example:

```
-Dcoherence.socketprovider=sdp
```

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 or SSL/SDP. 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 SSL keystore. More elaborate configuration can be defined with alternate identity and trust managers to allow for Certificate Authority trust validation. See Using SSL to Secure TCMP Communication in *Securing Oracle Coherence*.

Changing Transport Protocols

Coherence uses a TCP/IP message bus for reliable point-to-point communication between data service members. Additional transport protocol implementations are available and can be specified as required.

This section includes the following topics:

- Overview of Changing Transport Protocols
- Changing the Shared Transport Protocol
- Changing Transport Protocols Per Service Type

Overview of Changing Transport Protocols

Coherence supports multiple transport protocols. By default, all data services use the transport protocol that is configured for the unicast listener. This configuration results in a shared transport instance. You can also explicitly specify a transport protocol for a service which results in a service-specific transport instance. A service-specific transport instance can result in higher performance but at the cost of increased resource consumption. In general, a shared transport instance uses less resource consumption than service-specific transport instances.

Coherence supports the following reliable transport protocols:

- `datagram` – UDP protocol
- `tmb` (default) – TCP/IP message bus protocol
- `tmbs` – TCP/IP message bus protocol with SSL support. TMBS requires the use of an SSL socket provider. See Using the SSL Socket Provider.
- `sdmb` – Socket Direct Protocol (SDP) message bus.
- `sdmbs` – SDP message bus with SSL support. SDMBS requires the use of an SSL socket provider. See Using the SSL Socket Provider.

Changing the Shared Transport Protocol

You can override the default transport protocol that is used for reliable point-to-point communication. All data services use a shared instance of the specified transport protocol.

To specify the shared transport protocol, edit the operational override file and add a `<reliable-transport>` element within the `<unicast-listener>` 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>
         <reliable-transport system-
property="coherence.transport.reliable">tmbs
         </reliable-transport>
      </unicast-listener>
   </cluster-config>
</coherence>
```

The value can also be set using the `coherence.transport.reliable` system property.

Changing Transport Protocols Per Service Type

You can change the transport protocol that a data service uses for reliable point-to-point communication. A transport instance is created for the service and is not shared with other services. Use service-specific transport instances sparingly for select, high priority services.

To change the transport protocol per service type, override the `reliable-transport` initialization parameter of a service definition that is located in an operational override file. The following example changes the transport protocol for the `DistributedCache` service:

```
<?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="26">
                  <param-name>reliable-transport</param-name>
                  <param-value system-
property="coherence.distributed.transport.reliable">tmbs</param-value>
               </init-param>
            </init-params>
         </service>
      </services>
   </cluster-config>
</coherence>
```

The `reliable-transport` initialization parameter can be set for the `DistributedCache`, `ReplicatedCache`, `OptimisticCache`, `Invocation`, `Proxy`, and `FederatedCache` 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 `reliable-transport` parameter for a service.

Each service also has a system property that sets the transport protocol, respectively:

`coherence.distributed.transport.reliable` (also used for the `FederatedCache` service)

`coherence.replicated.transport.reliable`

`coherence.optimistic.transport.reliable`

`coherence.invocation.transport.reliable`

`coherence.proxy.transport.reliable`

Enabling CRC Validation for TMB

TCP/IP Message Bus (TMB) includes a cyclic redundancy check (CRC) implementation that validates network messages that are sent on the wire. CRC is commonly used to detect and handle message corruption in networks.

The CRC implementation calculates a check value that is based on the message body and adds the value to the message header. An additional check value is then calculated based on the resulting message header and is likewise added to the message header before sending the message. When TMB receives a message, the check values are recalculated and validated against the values that were sent with the message. If CRC validation fails, indicating packet corruption, then the bus connection is migrated.

CRC is not enabled by default. Enabling CRC does impact performance, especially for cache operations that mutate data (approximately 5% performance impact for `put`, `invoke`, and similar operations).

To enable CRC validation, set the following system property on all cluster members:

```
-Dcom.oracle.common.internal.net.socketbus.SocketBusDriver.crc=true
```

11

Using the Service Guardian

You can configure the service guardian to detect and resolve deadlocked service threads. You can create custom failure policies as required.
This chapter includes the following sections:

- Overview of the Service Guardian
- Configuring the Service Guardian
- Issuing Manual Guardian Heartbeats
- Setting the Guardian Log Thread Dump Frequency
  If Coherence cache server logs are overwhelmed with too many service guardian thread dumps in a short duration of time, you can increase the interval between service guardian thread dumps.

Overview of the Service Guardian

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 configuration 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.
This section includes the following topics:

- Setting the Guardian Timeout
- Using the Timeout Value From the PriorityTask API
- Setting the Guardian Service Failure Policy

Setting the Guardian Timeout

This section includes the following topics:

- Overview of Setting the Guardian Timeout
- Setting the Guardian Timeout for All Threads
- Setting the Guardian Timeout Per Service Type
- Setting the Guardian Timeout Per Service Instance

Overview of 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.

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 `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:

```
coherence.distributed.guard.timeout
coherence.replicated.guard.timeout
coherence.optimistic.guard.timeout
coherence.invocation.guard.timeout
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 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 Managing Thread Execution.

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.

Setting the Guardian Service Failure Policy

This section includes the following topics:

- Overview of Setting the Guardian Service Failure Policy
- Setting the Guardian Failure Policy for All Threads
- Setting the Guardian Failure Policy Per Service Type
- Setting the Guardian Failure Policy Per Service Instance
- Enabling a Custom Guardian Failure Policy

Overview of 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 causes the local node to exit gracefully, halting the process only when the duration of the coherence.shutdown.timeout system property exceeds.
- 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.

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. 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 `GuardSupport` class provides heartbeat methods that applications can use to manually issue heartbeats to the service 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);
```

Setting the Guardian Log Thread Dump Frequency

If Coherence cache server logs are overwhelmed with too many service guardian thread dumps in a short duration of time, you can increase the interval between service guardian thread dumps.

Set the property `coherence.guardian.log.threaddump.interval` to a time duration. The time duration format is a number followed by a letter, `h` for hour, `m` for minute, `s` for seconds. The default time duration is 3 seconds.

You can set the interval up to a maximum of 3 hours. Specifying a value larger than the maximum results in the interval being set to the maximum duration.

Thread dumps are important in order to identify thread deadlock. However, when thread dumps are too frequent in a system or the systems thread dumps are very large, use this setting to tune an acceptable balance.

Part III
Using Caches

Learn about Coherence cache types, how to configure caches and backing maps, how to cache data sources, how to implement quorum policies, and how to extend a cache configuration file.
Part III contains the following chapters:

12

Introduction to Coherence Caches

Coherence offers multiple cache types that can be used depending on your application requirements.
This chapter includes the following sections:

Understanding Distributed Caches

A distributed, or partitioned, cache is a clustered, fault-tolerant cache that has linear scalability.Data is partitioned among all storage members 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 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:

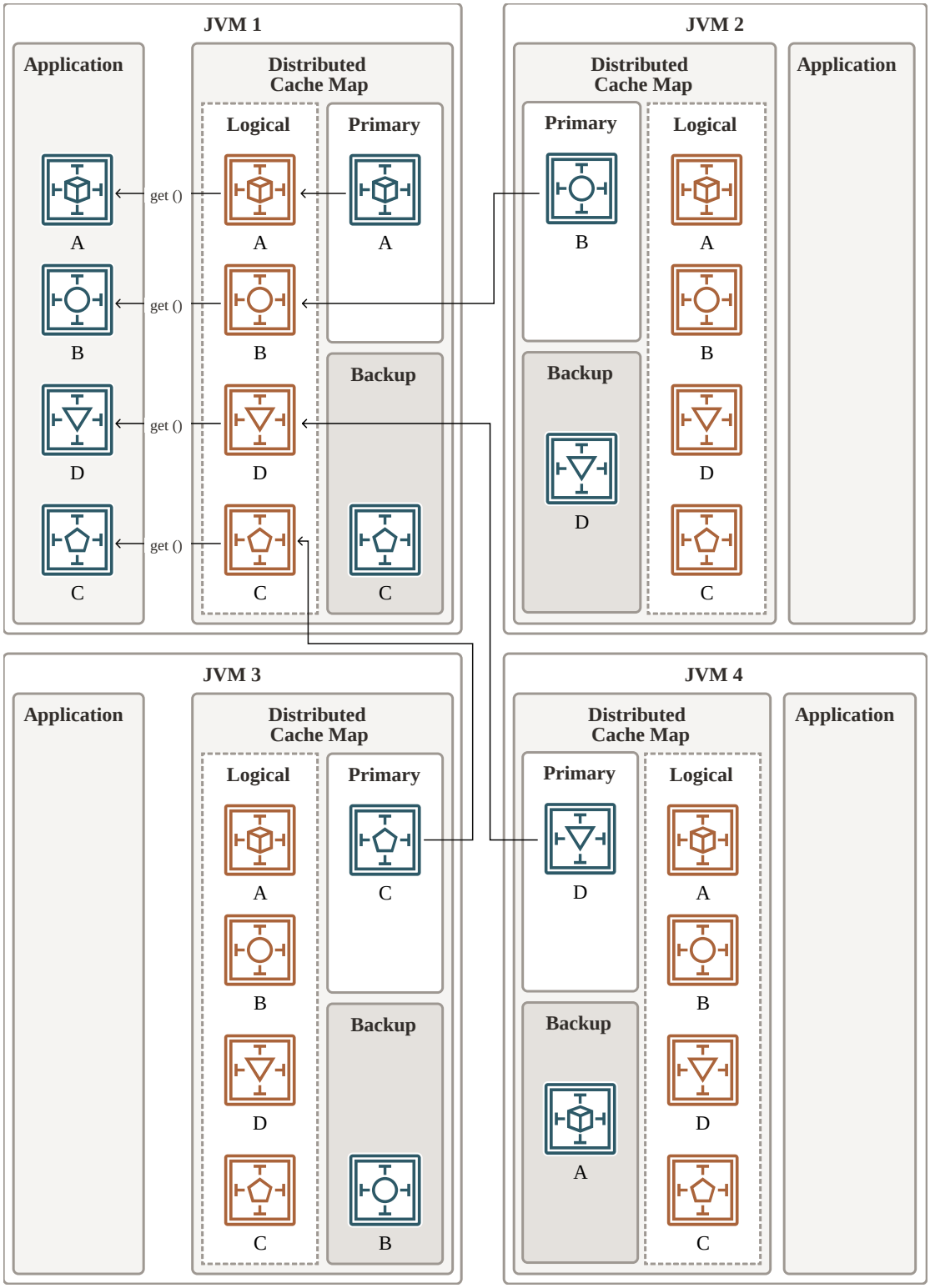
- **Partitioned:** The data in a distributed cache is spread out over all the servers in such a way that no two servers are responsible for the same piece of cached data. The size of the cache and the processing power associated with the management of the cache can grow linearly with the size of the cluster. Also, it means that read operations against data in the cache can be accomplished with a "single hop," in other words, involving at most one other server. Write operations are "single hop" if no backups are configured.
- **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 12-1 provides a conceptual view of a distributed cache during get operations.

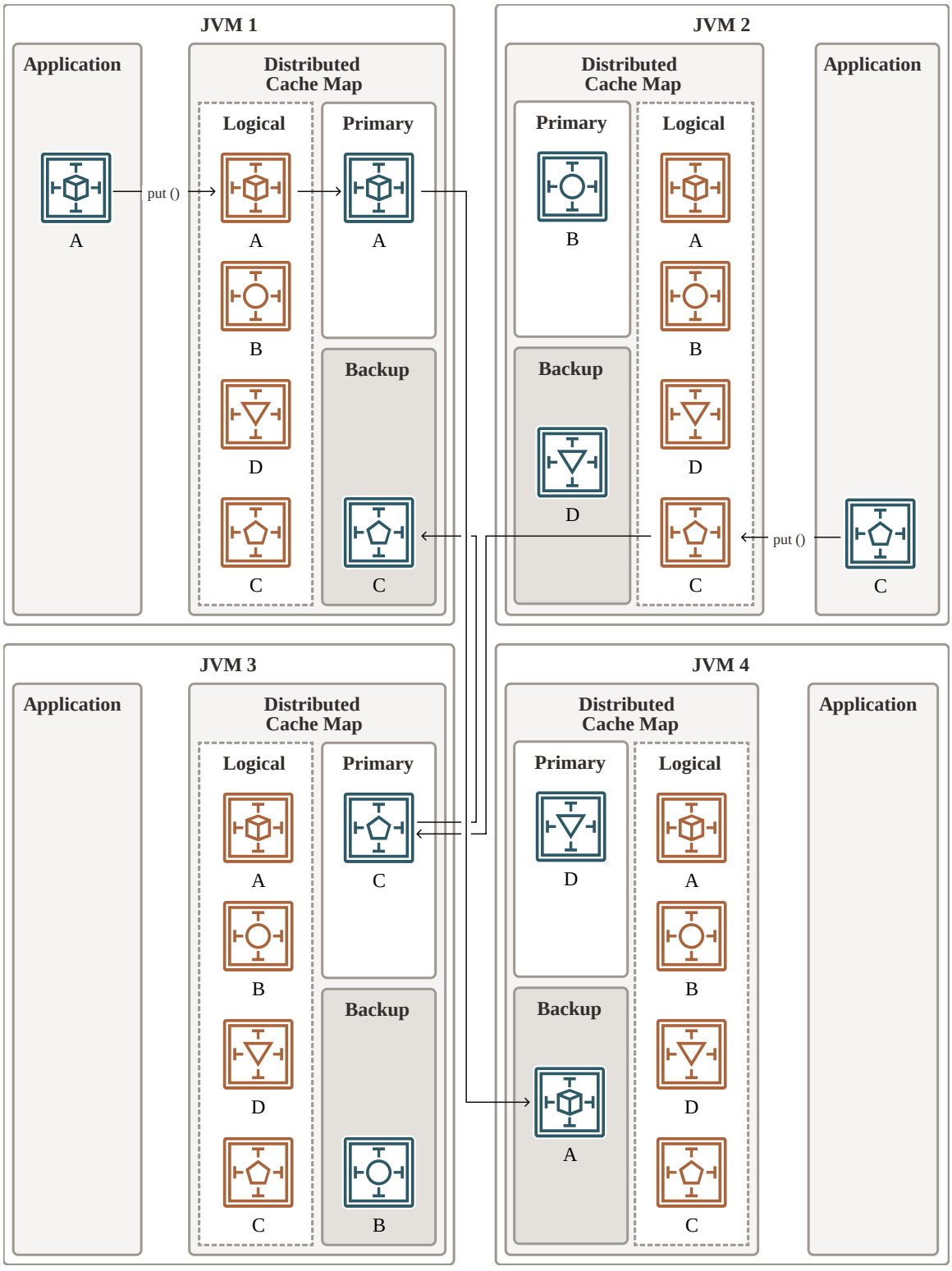
Figure 12-1 Get Operations in a Distributed Cache



Since each piece of data is managed by only one cluster node, read operations over the network are only "single hop" operations. This type of access is extremely scalable, since it can use point-to-point communication and thus take optimal advantage of a switched network.

Figure 12-2 provides a conceptual view of a distributed cache during put operations.

Figure 12-2 Put Operations in a distributed 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

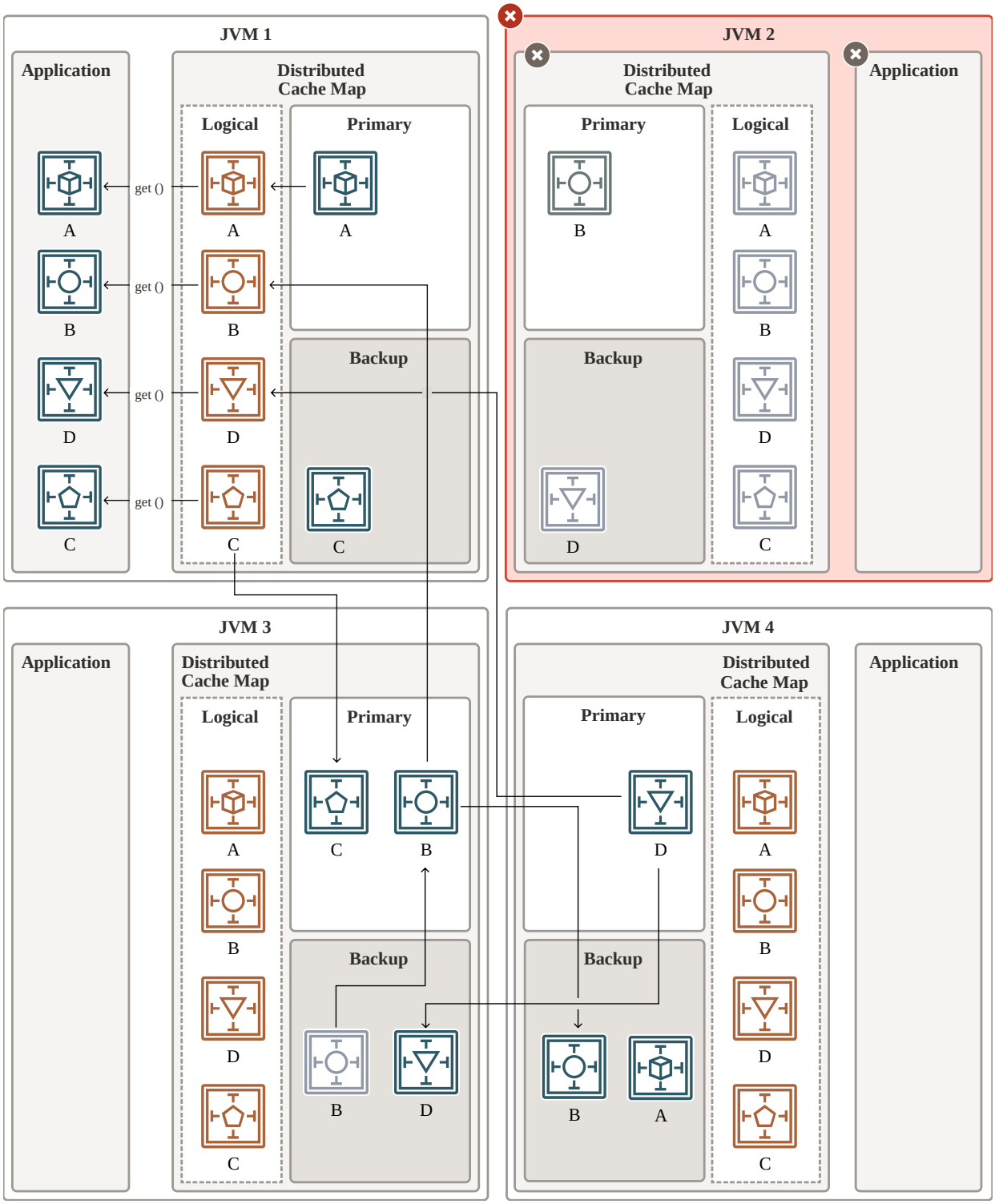
count setting is one.) If the cache data were not critical, which is to say that it could be re-loaded from disk, the backup count could be set to zero, which would allow some portion of the distributed cache data to be lost if a cluster node fails. If the cache were extremely critical, a higher backup 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 12-3 provides a conceptual view of a distributed cache during failover.

Figure 12-3 Failover in a Distributed Cache

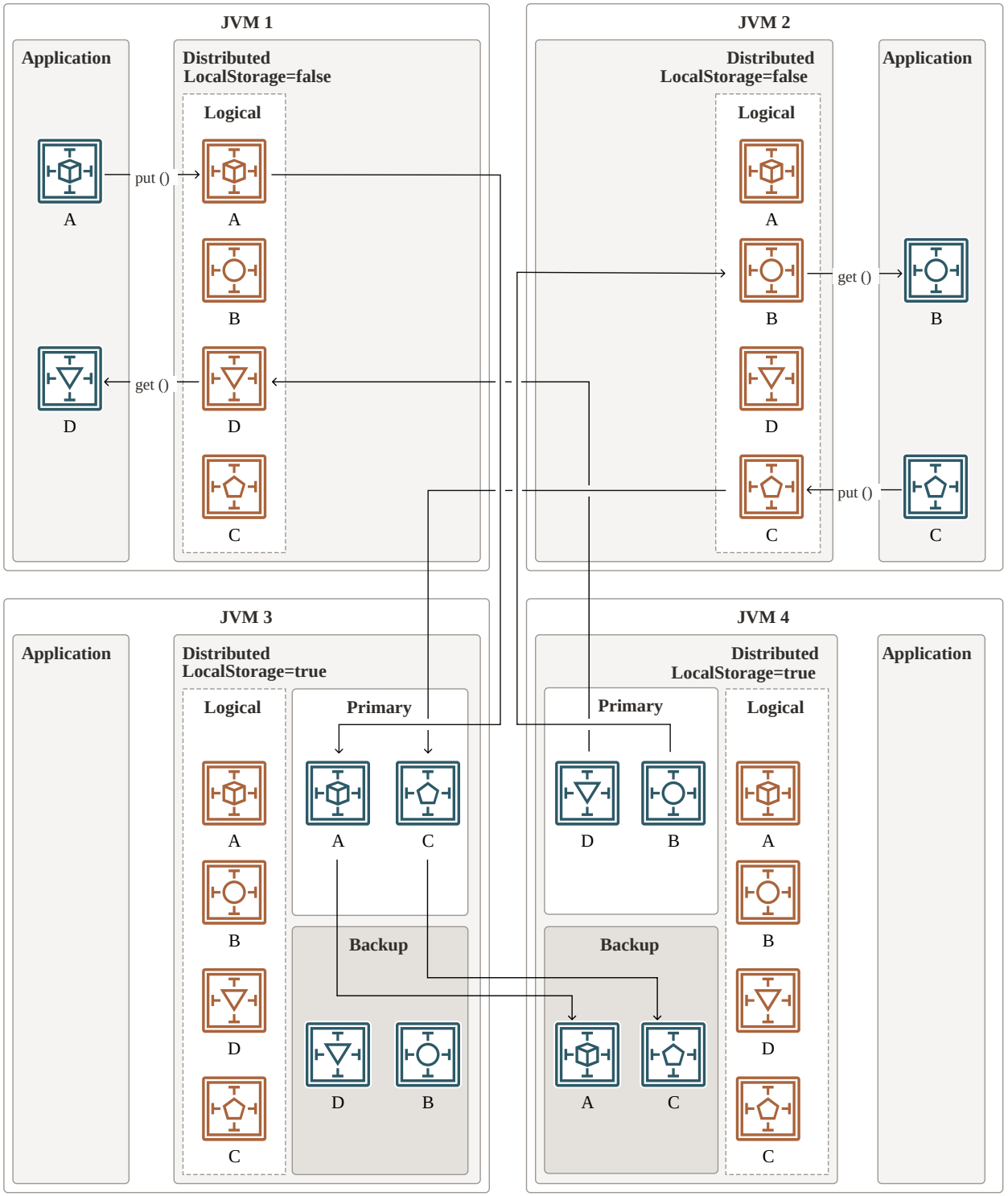


If there are multiple levels of backup, the first backup becomes responsible for the data; the second backup becomes the new first backup, and so on. Just as with the replicated cache service, lock information is also retained with server failure; the sole exception is when the locks for the failed cluster 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 12-4 provides a conceptual view of local storage in a distributed cache during get and put operations.

Figure 12-4 Local Storage in a Distributed Cache



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 elastic data. Different JVM versions are fine between storage enabled and disabled nodes, but all storage enabled nodes should use the same JVM version.
- 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.

Understanding Near Caches

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.

The objective of a near cache is to provide the best of both worlds between extreme read performance and extreme scalability of distributed/partitioned caches 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.

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.

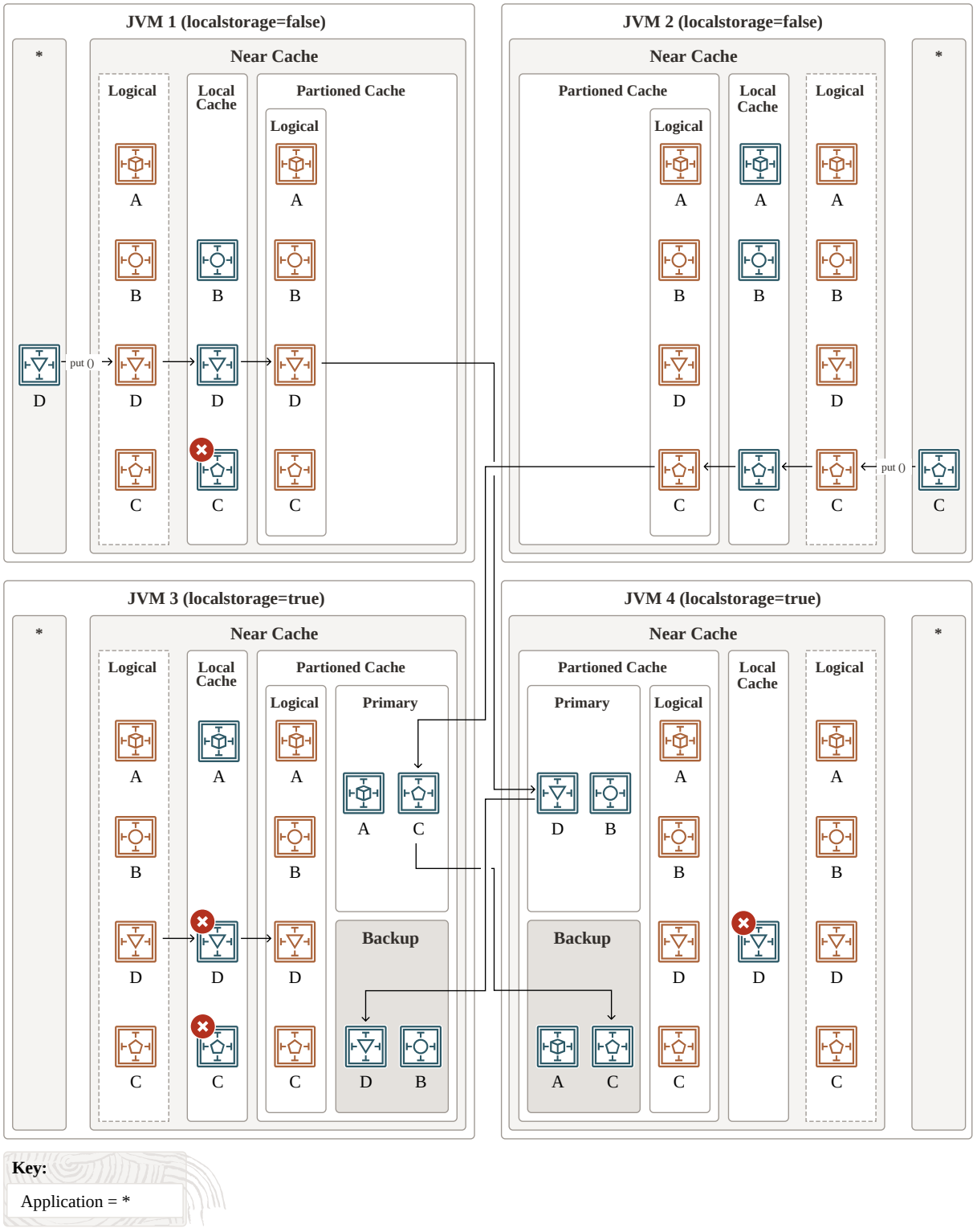
The typical deployment uses a local cache for the front cache. A local cache is a reasonable choice because it is thread safe, highly concurrent, size-limited, auto-expiring, and stores the data in object form. For the back cache, a partitioned cache is used.

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.

> **Note:**
>
> Because entries are stored in object form in the front of the near cache, the application must take care of synchronizing access by multiple threads in the same JVM. For example, if a thread mutates an entry that has been retrieved from the front of a near cache, the change is immediately visible to other threads in the same JVM.

Figure 12-5 provides a conceptual view of a near cache during put operations.

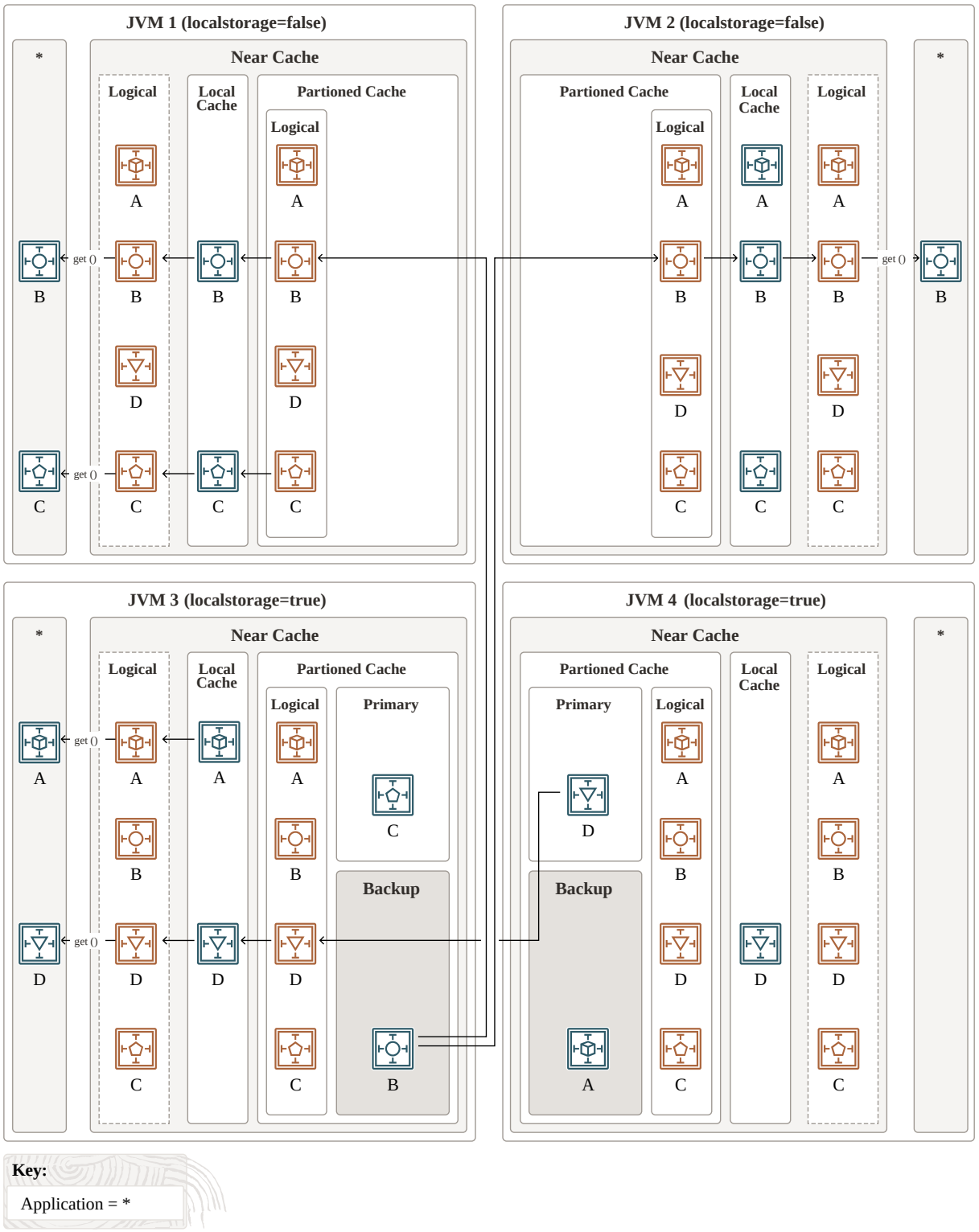
Figure 12-5 Put Operations in a Near Cache



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 12-6 provides a conceptual view of a near cache during get operations.

Figure 12-6 Get Operations in a Near Cache



Understanding View Caches

A view cache is a clustered, fault tolerant cache that provides a local in-memory materialized view of data stored in a distributed and partitioned cache.

The view cache provides extreme read performance while offering consistent writes leveraging from distributed/partitioned caches.

View caches are implemented using `ContinuousQueryCache`s, which have an internal map to store the contents of the view and a back `NamedCache` (optionally) to both gather the data from and listen to further updates. View caches define `ContinuousQueryCache` instances in the cache configuration. Two notable features are added to view caches:

1. Caches are populated as a part of activating the `ConfigurableCacheFactory` (typically via `DefaultCacheServer.startServerDaemon` or `ConfigurableCacheFactory.activate`).
2. Data is held serialized until accessed.

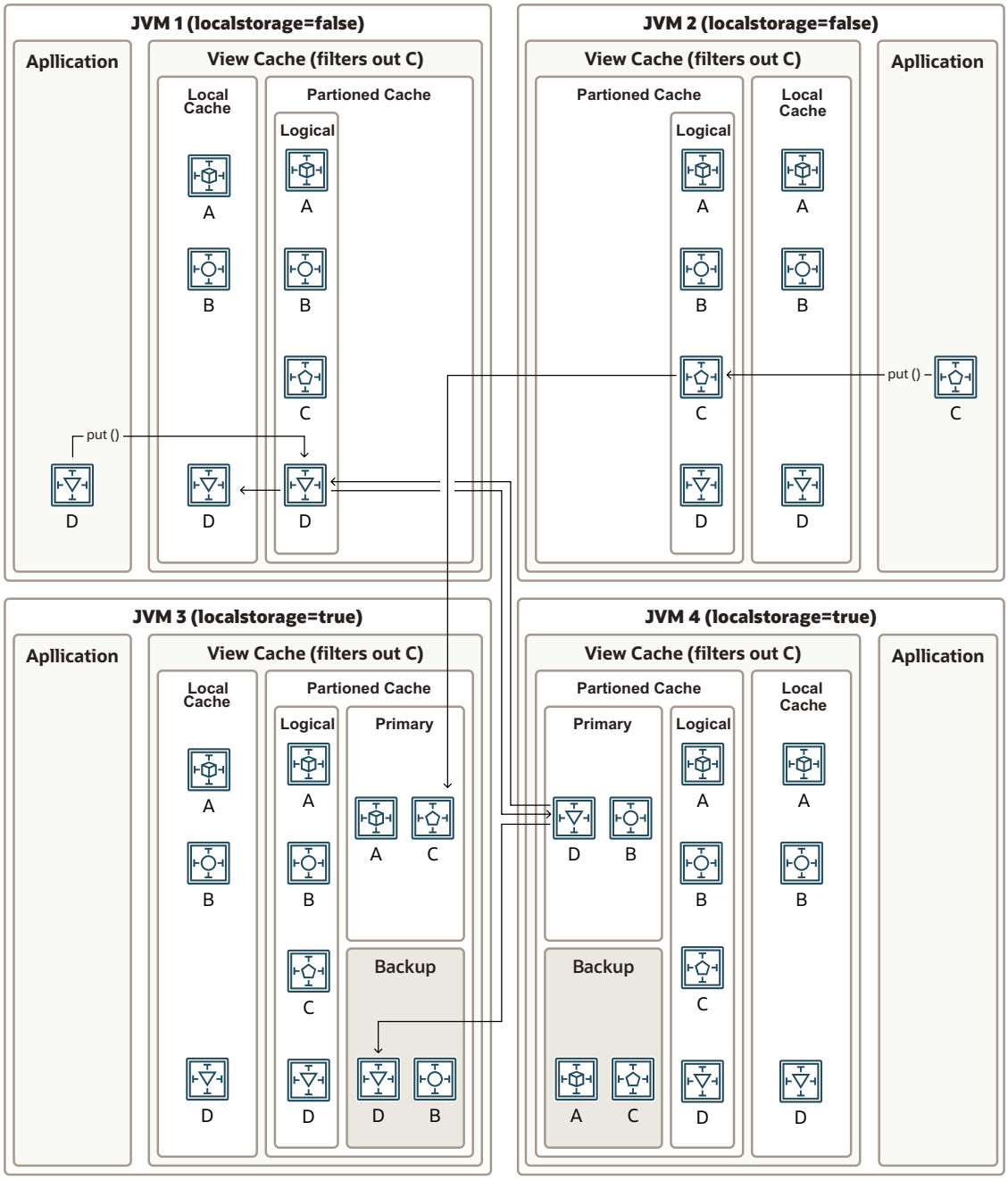
View caches have very high access speeds since the data is local to the view on each cluster node (JVM), which accesses the data from its own memory. This type of access is often referred to as zero latency access and is perfect for situations in which an application requires the highest possible speed in its data access.

See Understanding Distributed Caches, Using Continuous Query Caching for more information on the components that make up a view.

See Table 12-1 for more specifications of the view cache components.

Figure 12-7 provides a conceptual view of a view cache during put operations.

Figure 12-7 Put Operations in a View Cache



Understanding Local Caches

While it is not a clustered service, the local cache implementation is often used in combination with various clustered cache services as part of a near cache.

> **Note:**
>
> Applications that use a local cache as part of a near cache must make sure that keys are immutable. Keys that are mutable can cause threads to hang and deadlocks to occur.
>
> In particular, the near cache implementation uses key objects to synchronize the local cache (front map), and the key may also be cached in an internal map. Therefore, a key object passed to the `get()` method is used as a lock. If the key object is modified while a thread holds the lock, then the thread may fail to unlock the key. In addition, if there are other threads waiting for the locked key object to be freed, or if there are threads who attempt to acquire a lock to the modified key, then threads may hang and deadlocks can occur. For details on the use of keys, see the `java.util.Map` API documentation.

A local cache is completely contained within a particular cluster node. There are several attributes of the local cache that are worth noting:

- 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. See In-memory Cache with Expiring Entries and Controlling the Growth of a Local 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.

Understanding Remote Caches

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 in the Coherence cluster, for example, a Distributed/Partitioned Cache. See Overview of Coherence*Extend in *Developing Remote Clients for Oracle Coherence*.

Deprecated Cache Types

Replicated and Optimistic cache types are deprecated. View, Near, or Distributed/Partitioned Cache types should be used in their place.

The read and write performance and memory usage characteristics described in Table 12-1 can be used to determine which cache type to use. For documentation on Replicated and Optimistic cache types, refer to Introduction to Coherence Caches.

Summary of Cache Types

Compare the features of each cache type to select the right cache for your solution. 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 12-1 Summary of Cache Types and Characteristics

Characteristics	Partitioned Cache	Near Cache backed by partitioned cache	LocalCache not clustered	View Cache
Topology	Partitioned Cache	Local Caches + Partitioned Cache	Local Cache	Local View Cache + Partitioned Cache
Read Performance	Locally cached: instant **4** Remote: network speed **1**	Locally cached: instant **4** Remote: network speed **1**	Instant **4**	Instant **4**
Fault Tolerance	Configurable **3** Zero to Extremely High	Configurable **3** Zero to Extremely High	Zero	Configurable; Zero to Extremely High
Write Performance	Extremely fast **2**	Extremely fast **2**	Instant **4**	Extremely fast **2**
Memory Usage (Per JVM)	DataSize/ JVMs x Redundancy	LocalCache + [DataSize / JVMs]	DataSize	View Cache + [DataSize / JVMs x Redundancy]

Table 12-1 (Cont.) Summary of Cache Types and Characteristics

Characteristics	Partitioned Cache	Near Cache backed by partitioned cache	LocalCache not clustered	View Cache
Coherency	fully coherent	fully coherent **5**	not applicable (n/a)	fully coherent
Memory Usage (Total)	Redundancy x DataSize	[Redundancy x DataSize] + [JVMs x LocalCache]	n/a	[Redundancy x DataSize] + JVMs x View Cache **6**
Locking	fully transactional	fully transactional	fully transactional	fully transactional
Typical Uses	Read-write caches	Read-heavy caches w/ access affinity	Local data	Metadata

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 a few UDP unicast operations, depending on level of redundancy.
3. 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)
4. Limited by local CPU/memory performance, with negligible processing required (typically sub-millisecond performance).
5. Listener-based Near caches are coherent; expiry-based near caches are partially coherent for non-transactional reads and coherent for transactional access.
6. Size of view is dependent on the applied `Filter`. By default, a view does not filter entries.

13
Configuring Caches

You use the Coherence cache configuration deployment descriptor to define and configure caches for use by an application.
See Cache Configuration Elements for a complete reference of all the elements available in the cache configuration deployment descriptor. In addition, see Cache Configurations by Example for various sample cache configurations.

This chapter includes the following sections:

Overview of Configuring Caches

Caches are configured in a Coherence cache configuration deployment descriptor.By default, the first `coherence-cache-config.xml` deployment descriptor file that is found on the classpath is loaded. Coherence includes a sample `coherence-cache-config.xml` file in the `coherence.jar` library. 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.
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.

> **Note:**
>
> The following characters are reserved and cannot be used in cache names:
>
> - slash (`/`)
> - colon (`:`)
> - asterisk (`*`)
> - question mark (`?`)

This section includes the following topics:

- Using Exact Cache Mappings
- Using Name Pattern Cache Mappings

Using Exact Cache Mappings

Exact cache mappings map a specific cache name to a cache scheme definition. An application must provide the exact name as specified in the mapping to use a cache. Example 13-1 creates a single cache mapping that maps the cache name `example` to a distributed cache scheme definition with the scheme name `distributed`.

Example 13-1 Sample Exact 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 Name Pattern Cache Mappings

Name pattern cache mappings allow applications to use patterns when specifying a cache name. Patterns use the asterisk (*) wildcard. Name patterns alleviate an application from having to know the exact name of a cache. Example 13-2 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 name pattern `account-*` to the cache scheme definition with the scheme name `account-distributed`.

Example 13-2 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`.

As shown in Example 13-2, it is possible to have a cache name (for example `account-overdue`) that can be matched to multiple cache mappings. In such cases, if an exact cache mapping is defined, then it is always selected over any wildcard matches. Among multiple wildcard matches, the last matching wildcard mapping (based on the order in which they are defined in the file) is selected. Therefore, it is common to define less specific wildcard patterns earlier in the file that can be overridden by more specific wildcard patterns later in the file.

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, view, near, 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.

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.

This section includes the following topics:

- Defining Distributed Cache Schemes
- Defining Local Cache Schemes
- Defining Near Cache Schemes
- Defining View 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.

Example 13-3 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 13-3 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 Local Storage.

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.

This section includes the following topics:

- Sample Local Cache Definition
- Controlling the Growth of a Local Cache
- Configuring the Unit Calculator
- Specifying a Custom Eviction Policy

Sample Local Cache Definition

Example 13-4 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 clustering cache server distribution does not start if the only cache definition in the cache configuration file is a local cache.

Example 13-4 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 *Developing Remote Clients for Oracle Coherence*.

Controlling the Growth of a Local Cache

As shown in Example 13-4, 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. Entries that reach the expiry delay value are proactively evicted and are no longer accessible.

> **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.

When a cache entry expires, it is not immediately removed from the cache. Instead, it is removed the next time it is accessed after the expiration time. This means that there may be a delay between the expiration time and the actual eviction from the cache. However, when you get a cache entry, you will never receive an expired value. It is important to note that cached data may remain in the cache for a longer period than its expiration date.

When using a backing distributed (partitioned) cache (see In-memory Cache with Expiring Entries), there is a daemon called `EvictionTask` that periodically checks for evictions in the background. This daemon has an internal property called `EvictionDelay`, which is set to 250ms and specifies the minimum delay until the next eviction attempt. For additional information, see Capacity Planning.

Configuring the Unit Calculator

The unit calculator configured in a `<local-scheme>` is used by Coherence to calculate the size of an entry in a cache. There are two types of calculator built into Coherence:

- FIXED - This calculator gives every entry a size of one, so the cache units metrics for a cache using this calculator will be the same as the number of entries. This is the default calculator.
- BINARY - This calculator uses the size of the serialized binary key and value, plus some overhead for an entry to calculate the size. Caches using the binary calculator will have a cache units metric that is a representation of the actual amount of storage in bytes that the data in the cache is consuming.

The unit calculator is configured by setting the `<unit-calculator>` element inside the `<local-scheme>` element of a cache configuration file.

Example

```
<local-scheme>
  <scheme-name>local</scheme-name>
  <unit-calculator>BINARY</unit-calculator>
</local-scheme>
```

Configuring Unit Factor

The cache units metric is stored in a Java Integer, so for very large caches using a BINARY calculator, the cache units value in bytes may be larger than `Integer.MAX_VALUE`. In this case, you can configure the unit factor to apply a multiplier to the cache units metric. For example, if you use a BINARY unit calculator, you can use a unit factor of 1048576 to specify megabytes instead of bytes in the `<low-units>` and `<high-units>` settings.

You configure the unit factor in the `<unit-factor>` element of the `<local-scheme>` element.

The following example shows configuring the BINARY calculator with a unit factor of 1048576. The high and low units have an “M” suffix to denote they are mega-bytes. When viewing cache metrics of MBeans, the value for the cache units will now be mega-bytes instead of the default bytes.

```
<local-scheme>
  <scheme-name>local</scheme-name>
  <high-units>100M</high-units>
  <low-units>75M</low-units>
  <unit-calculator>BINARY</unit-calculator>
  <unit-factor>1048576</unit-factor>
</local-scheme>
```

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`.

Coherence's default eviction policy is very effective for most workloads; the majority of applications do not have to provide a custom policy. See local-scheme. 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 13-5 shows the implementation of a simple custom eviction policy:

Example 13-5 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 13-6 illustrates a Coherence cache configuration file with an eviction policy:

Example 13-6 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>20</high-units>
                  <unit-calculator>binary</unit-calculator>
               </local-scheme>
            </backing-map-scheme>
            <autostart>true</autostart>
         </distributed-scheme>
      </caching-schemes>
   </cache-config>
```

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.

This section includes the following topics:

- Sample Near Cache Definition
- Near Cache Invalidation Strategies

Sample Near Cache Definition

Example 13-7 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 13-7 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>
```

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 five different strategies of invalidating the front cache entries that have changed by other processes in the back cache.

> **Note:**
>
> When using an invalidation strategy of `all`, cache operations that modify a large number of entries (for example, a clear operation) can cause a flood of events that may saturate the network.

Table 13-1 describes the invalidation strategies.

Table 13-1 Near Cache Invalidation Strategies

Strategy Name	Description
`auto`	The default strategy if no strategy is specified. This strategy is identical to the `present` strategy.
`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.
`logical`	This strategy instructs a near cache to listen to all backing map events that are not synthetic deletes. A synthetic delete event could be emitted as a result of eviction or expiration. With this invalidation strategy, it is possible for the front map to contain cache entries that have been synthetically removed from the backing map. Any subsequent re-insertion of the entries to the backing map causes the corresponding entries in the front map to be invalidated.
`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. Note that the front map is reset if an extend client is disconnected with the proxy.

Defining View Cache Schemes

The `<view-scheme>` element is used to define a view caches. The `<view-scheme>` creates a `NamedCache` implementation that maintains a local in-memory cache and is backed by a clustered scheme. The clustered scheme could either be a federated, or a distributed scheme. By using the `<view-scheme>` element, you can harvest the benefits of having a local in-memory

store that is backed by a distributed scheme. The local store can be a full replica (all data) or a subset of the data in the distributed scheme. See view-scheme.

Example 13-8 defines a basic view cache that uses `view` as the scheme name and is mapped to the cache name `example`.

Example 13-8 Sample View 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>view</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>

   <caching-schemes>
      <view-scheme>
         <scheme-name>view</scheme-name>
         <back-scheme>
             <distributed-scheme>
                 <scheme-ref>partitioned-std</scheme-ref>
             </distributed-scheme>
         </back-scheme>
      </view-scheme>
   </caching-schemes>
</cache-config>
```

> **Note:**
>
> This example is a simple configuration of view. If a `<view-filter>` element is not defined, then the view cache will use an `AlwaysFilter`. If the `<view-filter>` element is defined, it uses the class-scheme mechanism outlined here.

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 13-9 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 13-9 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 13-9, 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 13-10 demonstrates multiple schemes reusing the same `LocalSizeLimited` base definition and overriding the `expiry-delay` property.

Example 13-10 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>

<distributed-scheme>
  <scheme-name>DistributedInMemoryCacheShortExpiry</scheme-name>
  <service-name>DistributedCacheShortExpiry</service-name>
  <backing-map-scheme>
    <local-scheme>
      <scheme-ref>LocalSizeLimited</scheme-ref>
      <expiry-delay>10m</expiry-delay>
    </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>
```

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 Cache Configuration Elements.

Many cache properties use default values unless a different value is explicitly given within the cache scheme definition. Clustered cache types 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 13-11 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 13-11 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-min>4</thread-count-min>
         <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 the use of parameter macros. Parameter macros are literal strings that are replaced with an actual value at runtime.Coherence includes predefined macros and also allows user-defined macros. This section includes the following topics:

- Using User-Defined Parameter Macros
- Using Predefined Parameter Macros

Using User-Defined Parameter Macros

User-defined parameter macros allow property values in a scheme to be replaced at runtime by values that are configured within cache mapping initialization parameters. User-defined parameter macros maximize the reuse of cache scheme definitions and can significantly reduce the size of a cache configuration file.

> **Note:**
>
> Parameter macros should not be used for service-scoped (shared by all caches in the same service) items, such as thread count, partition count, and service name. Parameter macros should only be used for cache-scoped items, such as expiry, high units, or cache stores to name a few.

To define a user-defined parameter macro, place a literal string within curly braces as the value of a property. A parameter macro can also include an optional default value by placing the value after the string preceded by a space. The form of a user-defined macro is as follows:

```
{user-defined-name default_value}
```

The following example creates a user-defined macro that is called `back-size-limit`. The macro is used for the `<high-units>` property of a backing map and allows the property value to be replaced at runtime. The macro specifies a default value of `500` for the `<high-units>` property.

```
<caching-schemes>
   <distributed-scheme>
      <scheme-name>distributed</scheme-name>
      <backing-map-scheme>
         <local-scheme>
            <high-units>{back-size-limit 500}</high-units>
         </local-scheme>
      </backing-map-scheme>
      <autostart>true</autostart>
   </distributed-scheme>
</caching-schemes>
```

At runtime, the `<high-units>` value can be replaced by using an initialization parameter that is defined within a cache mapping definition. The following example overrides the default value of 500 with 1000 by using an `<init-param>` element and setting the `<param-name>` element to `back-size-limit` and the `<param-value>` element to `1000`. See init-param.

```
<caching-scheme-mapping>
   <cache-mapping>
      <cache-name>example</cache-name>
      <scheme-name>distributed</scheme-name>
      <init-params>
         <init-param>
            <param-name>back-size-limit</param-name>
            <param-value>1000</param-value>
         </init-param>
      </init-params>
   </cache-mapping>
<caching-scheme-mapping>

<caching-schemes>
   <distributed-scheme>
      <scheme-name>distributed</scheme-name>
      <backing-map-scheme>
         <local-scheme>
            <high-units>{back-size-limit 500}</high-units>
         </local-scheme>
      </backing-map-scheme>
      <autostart>true</autostart>
   </distributed-scheme>
</caching-schemes>
```

The benefit of using user-defined parameter macros is that multiple cache mappings can use the same cache scheme and set different property values as required. The following example demonstrates two cache mappings that reuse the same cache scheme. However, the mappings result in caches with different values for the `<high-units>` element.

```
...
<caching-scheme-mapping>
   <cache-mapping>
      <cache-name>*</cache-name>
      <scheme-name>distributed</scheme-name>
   </cache-mapping>
   <cache-mapping>
      <cache-name>example</cache-name>
      <scheme-name>distributed</scheme-name>
      <init-params>
         <init-param>
            <param-name>back-size-limit</param-name>
            <param-value>1000</param-value>
         </init-param>
      </init-params>
   </cache-mapping>
<caching-scheme-mapping>

<caching-schemes>
   <distributed-scheme>
      <scheme-name>distributed</scheme-name>
      <backing-map-scheme>
         <local-scheme>
            <high-units>{back-size-limit 500}</high-units>
         </local-scheme>
      </backing-map-scheme>
      <autostart>true</autostart>
   </distributed-scheme>
</caching-schemes>
```

Using Predefined Parameter Macros

Coherence includes predefined parameter macros that minimize custom coding and enable the specification of commonly used attributes when configuring class constructor parameters. The macros must be entered within curly braces and are specific to either the `param-type` or `param-value` elements.

Table 13-2 describes the predefined parameter macros that may be specified.

Table 13-2 Predefined 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>`

`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>`

Table 13-2 (Cont.) Predefined Parameter Macros for Cache Configuration

<param-type>	<param-value>	Description

{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 13-2 (Cont.) Predefined 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-

Table 13-2 (Cont.) Predefined Parameter Macros for Cache Configuration

<param-type>	<param-value>	Description
		`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.

Using System Property Macros

The cache configuration deployment descriptor supports the use of system property macros.System property macros are literal strings that allow a portion of a value to be replaced with the value of a system property. System property macros can help reduce the size of the cache configuration file and can simplify runtime configuration.
To define a system property macro, place a literal string that represents a system property within curly braces and precede the curly braces with a dollar sign (`$`). A system property macro can also include an optional default value by placing the value after the string preceded by a space. The form of a system property macro is as follows:

```
${system.property default_value}
```

The following example is taken from the default cache configuration file and uses two system property macros: `${coherence.profile near}` and `${coherence.client direct}`. The macros are replaced at runtime with the values that are set for the respective system properties in order to use a specific cache scheme. If the system properties are not set, then the default values (`near-direct`) are used as the scheme name.

```
<caching-scheme-mapping>
   <cache-mapping>
      <cache-name>*</cache-name>
      <scheme-name>${coherence.profile near}-${coherence.client direct}
         </scheme-name>
   </cache-mapping>
</caching-scheme-mapping>
```

Setting the system properties at runtime changes the caching scheme that is used for the default cache. For example:

```
-Dcoherence.profile=thin
-Dcoherence.client=remote
```

Coherence 14c (14.1.2.0.0) introduces the following common Unix shell parameter expansion capabilities.

Support for nesting of system property macros to enable the defaulting value to be represented as a macro.

```
${system-property ${system-property-default default-value}}
```

In addition to the default delimiter of a space character, the colon character indicates a modifier for the macro parameter expansion.

Note that *word* referenced in each of the cases below is either a nested system property macro default or the default value.

- `${system-property:-word}`
  If the *system-property* is unset or null, the expansion of *word* is substituted. Otherwise, the value of the *system-property* is substituted.
- `${system-property:+word}`
  If the *system-property* is null or unset, nothing is substituted. Otherwise, the expansion of *word* is substituted.
- `${system-property:offset}`
  `${system-property:offset:length}`

  Perform a substring expansion of the *system-property's* value. Note that *offset* and *length* are integer values.

Substring expansion expands to up to `length` characters of the value of `system-property` starting at the character specified by `offset`.

If `length` is omitted, it expands to the substring of the value of `system-property` starting at the character specified by `offset` and extending to the end of the value.

If `offset` evaluates to a number less than zero, the value is used as an offset in characters from the end of the value of `system-property`.

If `length` evaluates to a number less than zero, it is interpreted as an offset in characters from the end of the value of `system-property`. Rather than a number of characters, the expansion is the characters between offset and that result.

> **Note:**
>
> A negative `offset` or `length` must be separated from the colon by at least one space to avoid being confused with the macro parameter default delimiter `:-`.

Examples illustrating substring expansion of a parameter:

```
Given property parameter of string value 01234567890abcdefgh.
 ${parameter:7} evaluates to 7890abcdefgh
 ${parameter:7:0} evaluates to empty string
 ${parameter:7:2} evaluates to 78
 ${parameter:7: -2} evaluates to 7890abcdef
 ${parameter: -7} evaluates to bcdefgh
 ${parameter: -7:0} evaluates to empty string
 ${parameter: -7:2} evaluates to bc
 ${parameter: -7: -2} evaluates to bcdef
```

Configuration level log messages identify misconfiguration of system property macro substring expressions such as an out of bounds error for an offset or an invalid integer value for an offset or length.

14

Implementing Storage and Backing Maps

Coherence uses backing maps to store data.You can choose from a variety of backing map implementations, which can be configured as required for an application.
This chapter includes the following sections:

Cache Layers

The Partitioned (Distributed) cache service in Coherence has three distinct layers that are used for data storage.

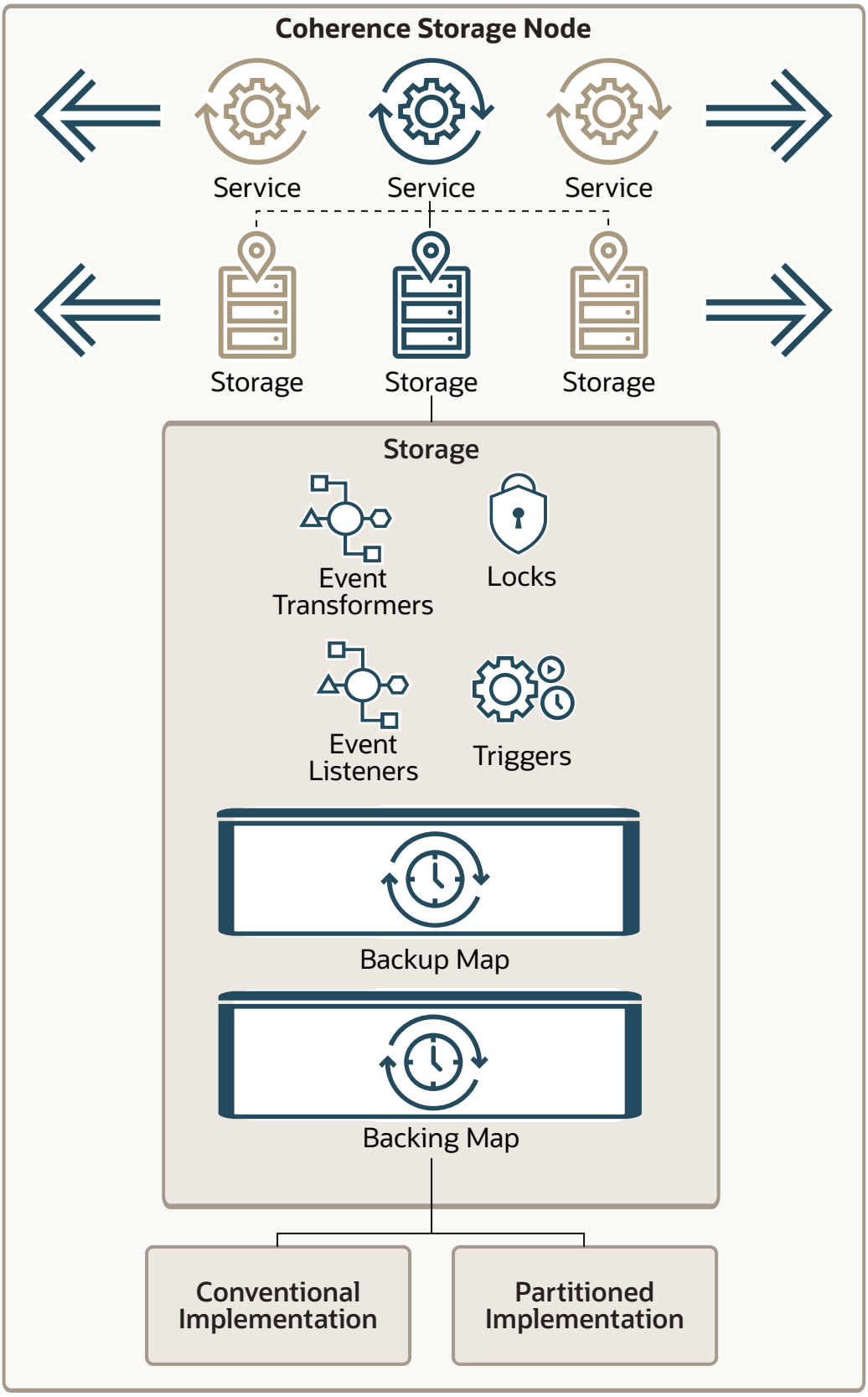
- **Client View** – The client view represents a virtual layer that provides access to the underlying partitioned data. Access to this tier is provided using the `NamedCache` interface. In this layer you can also create synthetic data structures such as `NearCache` or `ContinuousQueryCache`.
- **Storage 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 so on.
- **Backing Map** – The Backing Map is the server-side data structure that holds actual data.

Coherence allows users to configure out-of-the-box and custom backing map implementations. The only constraint for a Map implementation 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 14-1 shows a conceptual view of backing maps.

Figure 14-1 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.

> **Caution:**
>
> Be careful when using any backing map that does not store data on heap, especially if storing more data than can actually fit on heap. Certain cache operations (for example, unindexed queries) can potentially traverse a large number of entries that force the backing map to bring those entries onto the heap. Also, partition transfers (for example, restoring from backup or transferring partition ownership when a new member joins) force the backing map to bring lots of entries onto the heap. This can cause GC problems and potentially lead to `OutOfMemory` errors.

Coherence supports the following local storage implementations:

- **Safe HashMap:** This is the default lossless implementation. A lossless implementation is one, like the Java `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. See Capacity Planning. A local cache limits the size of the cache and automatically expires 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 backing store (such as 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 outside of the Java heap 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. 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`.
- **Journal:** This is a backing map implementation that stores data to either RAM, disk, or both RAM and disk. Journaling use the `com.tangosol.io.journal.JournalBinaryStore` class. See Using the Elastic Data Feature to Store Data.
- **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`.
- **Custom Map:** This is a backing map implementation that conforms to the `java.util.Map` interface.
  For example:

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

> **Note:**
>
> Irrespective of the backing map implementation you use, the map must be thread safe.

Operations

There are number of operation types performed against a backing map. The operations include:

- 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

The total amount of data placed into the data grid must not exceed some predetermined amount of memory. Applications can directly manage the total amount of data through application tier logic or can rely on automatic management using size- or expiry-based eviction. A backing map can store cache data in several ways depending on the backing map implementation (on or off heap, disk, and solid state). Keeping data in memory naturally provides dramatically smaller access and update latencies and is most commonly used. 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 the 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. The following example configures size constraints on the backing map:

```
<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. If no unit is specified in `<high-units>`, the default is byte. See <high-units>. You can also specify a `<unit-factor>`, as shown in the following example:

```
<backing-map-scheme>
  <local-scheme>
    <eviction-policy>LRU</eviction-policy>
    <high-units>100</high-units>
    <unit-calculator>BINARY</unit-calculator>
    <unit-factor>1000000</unit-factor>
  </local-scheme>
</backing-map-scheme>
```

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, which defaults to 80% of the `<high-units>`). If the value exceeds `Integer.MAX_VALUE`, then a unit factor is automatically used and the value for `<high-units>` and `<low-units>` are adjusted accordingly. 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 LRU and LFU).

The following backing map automatically evicts any entries that have not been updated for more than an hour. Entries that exceed one hour are not returned to a caller and are lazily removed from the cache when the next cache operation is performed or the next scheduled daemon `EvictionTask` is run. The `EvictionTask` daemon schedule depends on the cache expiry delay. The minimum interval is 250ms.

```
<backing-map-scheme>
  <local-scheme>
    <expiry-delay>1h</expiry-delay>
  </local-scheme>
</backing-map-scheme>
```

A backing map within a distributed scheme also supports sliding expiry. If enabled:

- Read operations extend the expiry of the accessed cache entries. The read operations include `get`, `getAll`, `invoke` and `invokeAll` without mutating the entries (for example, only `entry.getValue` in an entry processor).
- Any enlisted entries that are not mutated (for example, from interceptors or triggers) are also expiry extended.
- The backup (for expiry change) is done asynchronously if the operation is read access only. If a mutating operation is involved (for example, an eviction occurred during a `put` or a `putAll` operation), then the backup is done synchronously.

> **Note:**
>
> Sliding expiry is not performed for entries that are accessed based on query requests like `aggregate` and `query` operations.

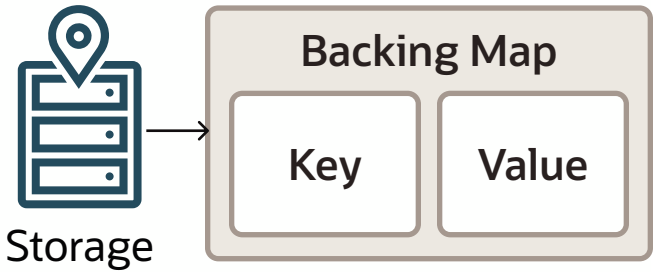
To enable sliding expiry, set the `<sliding-expiry>` element, within a `<backing-map-scheme>` element to `true` and ensure that the `<expiry-delay>` element is set to a value greater than zero. For example,

```
<distributed-scheme>
   <scheme-name>dist-expiry</scheme-name>
   <service-name>DistributedExpiry</service-name>
   <backing-map-scheme>
      <sliding-expiry>true</sliding-expiry>
      <local-scheme>
         <expiry-delay>3s</expiry-delay>
      </local-scheme>
   </backing-map-scheme>
</distributed-scheme>
```

Using Partitioned Backing Maps

Coherence provides a partitioned backing map implementation that differs from the default backing map implementation.The conventional backing map implementation stores entries for all partitions owned by the corresponding node. During partition transfer, it could also hold in-flight entries that, from the client perspective, are temporarily not owned by anyone. Figure 14-2 shows a conceptual view of the conventional backing map implementation.

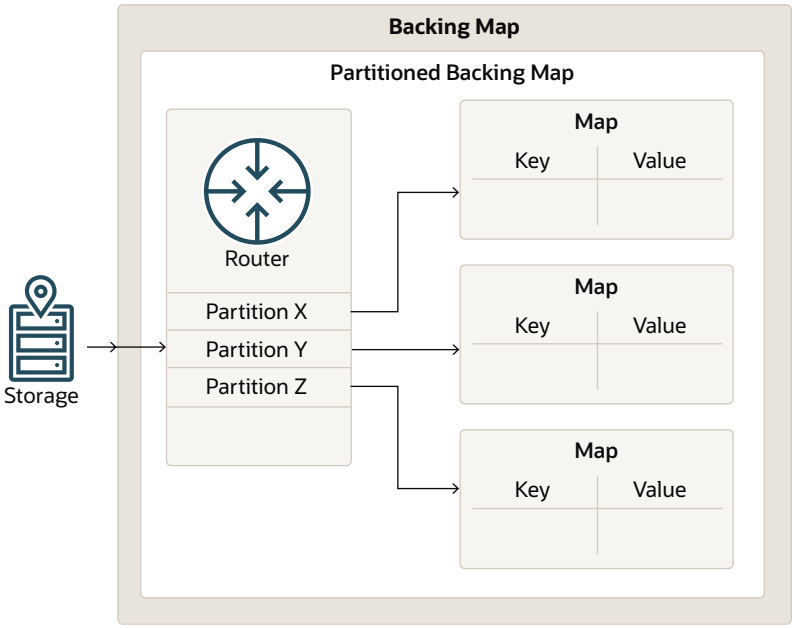
Figure 14-2 Conventional Backing Map Implementation



A partitioned backing map is a multiplexer of actual `Map` implementations, each of which contains only entries that belong to the same partition. Partitioned backing maps raise the storage limit (induced by the `java.util.Map` API) from 2G for a backing map to 2G for each partition. Partitioned backing maps are typically used whenever a solution may reach the 2G backing map limit, which is often possible when using the elastic data feature. See Using the Elastic Data Feature to Store Data.

Figure 14-3 shows a conceptual view of the partitioned backing map implementation.

Figure 14-3 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-file-manager>
      <initial-size>1MB</initial-size>
      <maximum-size>50MB</maximum-size>
    </nio-file-manager>
    <high-units>8192</high-units>
    <unit-calculator>BINARY</unit-calculator>
```

```
  </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).

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 can work 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.

This section includes the following topics:

- Journaling Overview
- Defining Journal Schemes
- Changing Journaling Behavior

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 can automatically accept the overflow from the RAM Journal, allowing for caches to expand far beyond the size of RAM.

> **Note:**
>
> Elastic data is ideal when performing key-based operations and typically not recommend for large filter-based operations. 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 General Guidelines in *Administering Oracle Coherence*.

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`).

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 ramjournal-scheme and flashjournal-scheme.

This section includes the following topics:

- Configuring a RAM Journal Backing Map
- Configuring a Flash Journal Backing Map
- Referencing a Journal Scheme
- Using Journal Expiry and Eviction
- 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.

```
<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 Journal Expiry and Eviction

The RAM and flash journal can be size-limited. They can restrict the number of entries to store and automatically evict entries when the journal becomes full. Furthermore, both the sizing of entries and the eviction policies can be customized. The following example defines expiry and eviction settings for a RAM journal:

```
<distributed-scheme>
   <scheme-name>distributed-journal</scheme-name>
   <service-name>DistributedCacheFlashJournal</service-name>
   <backing-map-scheme>
      <ramjournal-scheme>
         <eviction-policy>LFU</eviction-policy>
         <high-units>100</high-units>
         <low-units>80</low-units>
         <unit-calculator>Binary</unit-calculator>
         <expiry-delay>0</expiry-delay>
      </ramjournal-scheme>
   </backing-map-scheme>
   <autostart>true</autostart>
</distributed-scheme>
```

Using a Journal Scheme for Backup Storage

Journal schemes are used for backup storage as well as for backing maps. By default, Flash Journal is used as the 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 RAM 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-ram</scheme-name>
         </backup-storage>
         <backing-map-scheme>
            <ramjournal-scheme/>
         </backing-map-scheme>
      <autostart>true</autostart>
   </distributed-scheme>

   <ramjournal-scheme>
      <scheme-name>example-ram</scheme-name>
   </ramjournal-scheme>
</caching-schemes>
```

Enabling a Custom Map Implementation for a Journal Scheme

Journal schemes can be configured to use a custom backing map as required. Custom map implementations must extend the `CompactSerializationCache` 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 `MyCompactSerializationCache`.

```
<flashjournal-scheme>
   <scheme-name>example-flash</scheme-name>
   <class-name>package.MyCompactSerializationCache</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.

This section includes the following topics:

- 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 list summarizes the default characteristics of a RAM journal. See ramjournal-manager.

- Binary values are limited by default to 64KB (and a maximum of 4MB). A flash journal is automatically used if a binary value exceeds the configured limit.

- An individual buffer (a journal file) is limited by default to 2MB (and a maximum of 2GB). The maximum file size should not be changed.
- A journal is composed of up to 512 files. 511 files are usable files and one file is reserved for depleted states.
- The total memory used by the journal is limited to 1GB by default (and a maximum of 64GB). A flash journal is automatically used if the total memory of the journal exceeds the configured limit.

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 RAM journal 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 system-property="coherence.ramjournal.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 list summarizes the default characteristics of a flash journal. See flashjournal-manager.

- 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. 511 files are usable files and one file is reserved for depleted states. A journal is limited by default to 1TB, with a theoretical maximum of 2TB.
- A journal has a high journal size of 11GB by default. The high size determines when to start removing stale values from the journal. This is not a hard limit on the journal size, which can still grow to the maximum file count (512).
- Keys remain in memory in a compressed format. For values, only the unwritten data (being queued or asynchronously written) remains in memory. When sizing the heap, a reasonable estimate is to allow 50 bytes for each entry to hold key data (this is true for both RAM and Flash journals) and include additional space for the buffers (16MB). The entry size is increased if expiry or eviction is configured.
- A flash journal is automatically used as overflow when the capacity of the RAM journal is reached. The flash journal can be disabled by setting the maximum size of the flash journal to 0, which means journaling exclusively uses a RAM journal.

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 flash journal 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-size>100G</maximum-size>
            <directory>/coherence_storage</directory>
         </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 Asynchronous Backup

Distributed caches support both synchronous and asynchronous backup.With synchronous backup, clients are blocked until a backup operation completes. With asynchronous backup, clients continue to respond to requests during backup operations. Backups are performed synchronously unless asynchronous backup is explicitly enabled.
Asynchronous backup is typically used to increase client performance. However, applications that use asynchronous backup must handle the possible effects on data integrity. Specifically, cache operations may complete before backup operations complete (successfully or unsuccessfully) and backup operations may complete in any order. Consider using asynchronous backup if an application does not require backups (that is, data can be restored from a system of record if lost) but the application still wants to offer fast recovery in the event of a node failure.

> **Note:**
>
> The use of asynchronous backups together with rolling restarts requires the use of the `shutdown` method to perform an orderly shut down of cluster members instead of the `stop` method or `kill -9`. Otherwise, a member may shutdown before asynchronous backups are complete. The `shutdown` method guarantees that all updates are complete.

To enable asynchronous backup for a distributed cache, add an `<async-backup>` element, within a `<distributed-scheme>` element, that is set to `true`. For example:

```
<distributed-scheme>
   ...
   <async-backup>true</async-backup>
   ...
</distributed-scheme>
```

To enable asynchronous backup for all instances of the distributed cache service type, override the partitioned cache service's `async-backup` 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="27">
                  <param-name>async-backup</param-name>
                  <param-value
                     system-property="coherence.distributed.asyncbackup">
                     false
                  </param-value>
               </init-param>
            </init-params>
         </service>
      </services>
   </cluster-config>
</coherence>
```

The `coherence.distributed.asyncbackup` system property is used to enable asynchronous backup for all instances of the distributed cache service type instead of using the operational override file. For example:

```
-Dcoherence.distributed.asyncbackup=true
```

Using the Read Locator

Currently Coherence requests are serviced by the primary owner of the associated partition(s) (ignoring the client side caches `NearCache` and `ContinuousQueryCache`). Coherence 14c (14.1.2.0.0) introduces the read-locator feature. Read locator allows for certain requests to be targeted to non-primary partition owners (backups) to balance request load or reduce latency.

Currently only `NamedMap.get` and `NamedMap.getAll` (see Interface NamedMap) requests support this feature. If the application chooses to target a non-primary partition owner, then there is an implied tolerance for stale reads. This may be possible as the primary (or other backups) process future/in-flight changes while the targeted member that performed the read has not.

Coherence now provides an ability for applications to choose the appropriate `read-locator` for a cache or service by using the cache configuration, as shown below:

```
...
    <distributed-scheme>
      <scheme-name>example-distributed</scheme-name>
      <service-name>DistributedCache</service-name>
      <backing-map-scheme>
          <read-locator>closest</read-locator>
      </backing-map-scheme>
      <autostart>true</autostart>
    </distributed-scheme>
    ...
```

The following `read-locator` values are supported:

- `primary` - (default) target the request to the primary only.
- `closest` - find the 'closest' owner based on the member, machine, rack, or site information for each member in the partition's ownership chain (primary and backups).
- `random` - pick a random owner in the partition's ownership chain.
- `random-backup` - pick a random backup owner in the partition's ownership chain.
- `class-scheme` - provide your own implementation that receives the ownership chain and returns the member to the target.

For information about the client side caches `NearCache` and `ContinuousQueryCache`, see Understanding Near Caches and Using Continuous Query Caching.

Scheduling Backups

Coherence provides an ability for applications to favor the write throughput over the coherent backup copies (async-backup). This can result in acknowledged write requests being lost if they are not successfully backed up. Acknowledgement comes in the form of control being returned when using the synchronous API against a mutating method (put/invoke - see Interface ConcurrentMap and Interface InvocableMap), or receiving a notification of the completion of a write request through the asynchronous API.
Internally this still results in `n` backup messages being created for `n` write requests, which has a direct impact on write throughput. To improve the write throughput, Coherence 14c (14.1.2.0.0) introduces "Scheduled" (or periodic) backups. This feature allows the number of backup messages to be `<n`.

The current `async-backup` XML element (see Using Asynchronous Backup) has been augmented to accept more than a simple `true|false` value and now supports a time-based value. This allows applications to suggest a soft target of how long they are willing to tolerate stale backups. At runtime Coherence may decide to accelerate backup synchronicity, or increase the staleness based on the primary write throughput.

> **Note:**
>
> You must be careful when choosing the backup interval because there is a potential for losing updates in the event of losing a primary partition owner. All the updates waiting to be sent by that primary owner will not be reflected when the corresponding backup owner is restored and becomes the primary owner.

Example 14-1 Example Configuration

The following distributed scheme contains an example of setting the scheduled backup interval of ten seconds:

```
...
<distributed-scheme>
  <scheme-name>example-distributed</scheme-name>
  <service-name>DistributedCache</service-name>
  <autostart>true</autostart>
  <async-backup>10s</async-backup>
</distributed-scheme>
...
```

You can also use a default system property, which will take effect on all the distributed schemes used. For example:

```
-Dcoherence.distributed.asyncbackup=10s
```

Using Asynchronous Persistence

Asynchronous persistence mode allows the storage servers to persist data asynchronously, thus a mutating request is successful once the primary stores the data and (if there is a synchronous backup) once the backup receives the update.
This allows writes to not be blocked on latency of writing to the underlying device, however does introduce a potential to lose writes. This mode is primarily viable when data can be replenished and the persisted data provides an optimized means to recover data.

You can enable asynchronous persistence by specifying the `<persistence-mode>` of the `<persistence-environment>` as `active-async`.

There is an out-of-the-box persistence environment that specifies this mode and can be referred to by customer applications by either:

- Specifying the JVM argument:

  ```
  -Dcoherence.distributed.persistence.mode=active-async
  ```

  > **Note:**
  >
  > This is used by all distributed services.

- Referring to the `default-active-async` persistence environment in the cache configuration:
  See persistence.

```
<distributed-scheme>
         …….…….
         <persistence>
             <environment>default-active-async</environment>
         </persistence>
         …….…….
</distributed-scheme>
```

> **Note:**
>
> This allows you to selectively choose which services have persistence enabled.

You can also define a new `<persistence-environment>` element in the operational configuration file and specify the mode to be `active-async` to enable asynchronous persistence. See persistence-environment.

For example:

```
<persistence-environment id="async-environment">
         <persistence-mode>active-async</persistence-mode>
         <active-directory>/tmp/store-bdb-active</active-directory>
         <snapshot-directory>/tmp/store-bdb-snapshot</snapshot-directory>
         <trash-directory>/tmp/store-bdb-trash</trash-directory>
</persistence-environment>
```

To use this environment in the nominated distributed-schemes, it must be referenced in the cache configuration.

For example:

```
<distributed-scheme>
         …….…….
         <persistence>
             <environment>async-environment</environment>
         </persistence>
         …….…….
</distributed-scheme>
```

> **Note:**
>
> With asynchronous persistence, it is possible to have data loss if the cluster is shutdown before the persistent transaction is complete.

To ensure that there is no data loss during a controlled shutdown, customers can leverage the service suspend feature. A service is only considered suspended once all data is fully written, including asynchronous persistence tasks, entries in the write-behind queue of a readwrite-backing-map, and so on.

Using Persistent Backups

Coherence 14c (14.1.2.0.0) adds a new persistence mode that stores backup partitions on disk as additional copies of persisted primary ones.

This section includes the following topics:

- About Persistent Backups

About Persistent Backups

In active persistence mode, Coherence eagerly persists primary partitions to disk to permit recovery of data when a complete shutdown (voluntary or not) occurs.

In this mode and at runtime, every cache mutation is saved to disk in a synchronous manner. That is, every operation that waits for a cache update to complete before returning control to the client, also waits for persistence to complete.

While active persistence increases reliability, there are still conditions that could cause data to be lost. For example, when members do not save in the same location, and when some locations become inaccessible or corrupted.

To alleviate this issue, a new mode "active-backup", also persists the backup partitions to disk. This operation is always asynchronous so as to have as little impact on performance as possible.

In this mode, when primary partitions are not found during recovery, backup ones are looked for and used instead. The simplest case is that of a cluster where two storage-enabled members are running on two difference machines with each having its own storage. In this topology, they each divide almost equally into primary and backup partitions: member 1 will own primary partition P1, P2, and P3, and backup partitions B4, B5, B6, and B7. Member 2 will own P4, P5, P6, and P7 as well as B1, B2, and B3.

In the case, where both members are lost, but only one storage can be recovered, you can restart any member by using a single surviving storage and recover the entire set of data. In the active (non-backup) mode, only about half of the data set would be recovered in the same situation.

Configuring Active Persistence Mode

You can configure this mode using the persistence mode "active-backup" value in `coherence.distributed.persistence.mode`. For example:

```
-Dcoherence.distributed.persistence.mode=active-backup
```

Optionally, you can configure a backup location to store backup partitions using `<backup-directory/>` in the Coherence operational override file. Otherwise, they are stored in the path specified by `coherence.distributed.persistence.base.dir` (or its default) followed by '`/backup`'.

For example:

```
<persistence-environments>
   <persistence-environment id="active-backup-environment">
     <persistence-mode system-property="coherence.persistence.mode">active-
backup</persistence-mode>
       <active-directory system-property="coherence.persistence.active.dir">/
store-bdb-active</active-directory>
       <backup-directory system-property="coherence.persistence.backup.dir">/
store-bdb-backup</backup-directory>
       <snapshot-directory system-
property="coherence.persistence.snapshot.dir">/store-bdb-snapshot</snapshot-
```

```
directory>
       <trash-directory system-property="coherence.persistence.trash.dir">/
store-bdb-trash</trash-directory>
   </persistence-environment>
</persistence-environments>
```

Performance Considerations

While runtime performance impact is minimal, there are still performance considerations to account for. Since there are more persistence files created, start-up and recovery times can be affected, especially if the number of partitions configured is large. To minimize the impact on start-up/recovery times, you can use recovery quorum.

For example: in cache config:

```
<distributed-scheme>
    <scheme-name>partitioned</scheme-name>
    <backing-map-scheme>
        <partitioned>true</partitioned>
        <read-write-backing-map-scheme>
            <internal-cache-scheme>
                <local-scheme>
                </local-scheme>
            </internal-cache-scheme>
        </read-write-backing-map-scheme>
    </backing-map-scheme>
    <partitioned-quorum-policy-scheme>
       <recover-quorum>3</recover-quorum>
    </partitioned-quorum-policy-scheme>
    <autostart>true</autostart>
</distributed-scheme>
```

In the above example, you can set the recover-quorum value to the desired number of members in the cluster, which causes recovery to occur only when that number of members has joined. Instead of recovery taking place at the same time as distribution, it occurs only once and completes faster.

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. However, entries that change by more than 50% typically demonstrate little or no performance gain. In this case, the use of delta backup should only be used if no adverse effect on performance is observed.
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.

This section includes the following topics:

- Enabling Delta Backup

- Enabling a Custom Delta Backup Compressor

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="coherence.distributed.compressor">
                     standard</param-value>
               </init-param>
            </init-params>
         </service>
      </services>
   </cluster-config>
</coherence>
```

The `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:

```
-Dcoherence.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. 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>
```

Integrating Caffeine

Coherence 14c (14.1.2.0.0) adds a Caffeine backing map implementation, enabling you to use Caffeine wherever the standard Coherence local cache can be used: as a local cache, as a backing map for a partitioned cache, or as a front map for a near cache.
This section includes the following topics:

- About Caffeine
  Caffeine is a high performance, near optimal caching library. It improves upon Coherence's standard local cache by offering better read and write concurrency, as well as a higher hit rate.
- Using Caffeine
- Configuring Caffeine

About Caffeine

Caffeine is a high performance, near optimal caching library. It improves upon Coherence's standard local cache by offering better read and write concurrency, as well as a higher hit rate.

Caffeine implements an adaptive eviction policy that can achieve a significantly higher hit rate across a large variety of workloads. You can leverage this feature to either reduce latencies or maintain the same performance with smaller caches. This may allow for decreasing the operational costs due to requiring fewer resources for the same workload. For more information, see Caffeine

The adaptive nature of this policy, nicknamed W-TinyLFU, allows it to stay robustly performant despite changes in the runtime workload. For more information, see Adaptive Software Cache Management and TinyLFU: A Highly Efficient Cache Admission Policy. Those changes may be caused by variations in the external request pattern or differences caused by the application's evolution. This self-optimizing, O(1) algorithm avoids the need to manually analyze the application and tune the cache to a more optimal eviction policy.

The following table shows cache hit rates for Caffeine's W-TinyLFU compared to other commonly used cache eviction policies, for various types of workloads:

Table 14-1 Cache Hit Rates for Caffeine's W-TinyLFU vs. Other Common Cache Eviction Policies

Workload	W-TinyLFU	Hybrid	LRU	LFU
An analytical loop	**32.7%**	2.6%	1.0%	1.4%
Blockchain mining	32.3%	12.1%	**33.3%**	0.0%
OLTP	**40.2%**	15.4%	33.2%	9.6%
Search	**42.5%**	31.3%	12.0%	29.3%
Database	**44.8%**	37.0%	20.2%	39.1%

For an in-depth introduction to Caffeine, Oracle strongly recommends you to see the following articles:

- Design of a Modern Cache - Part I
- Design of a Modern Cache - Part II

Using Caffeine

Caffeine is integrated tightly into Coherence and is almost as easy to use as any of the built-in backing map implementations that Coherence provides. The only difference is that Caffeine requires you to add a dependency on Caffeine to your project's POM file, as it is defined as an optional dependency within Coherence POM.

To be able to use Caffeine, you need to add the following dependency to your POM file:

```
<dependency>
  <groupId>com.github.ben-manes.caffeine</groupId>
  <artifactId>caffeine</artifactId>
  <version>${caffeine.version}</version>
</dependency>
```

The supported Caffeine versions are `3.1.0` or higher.

Configuring Caffeine

After you have added the dependency, Caffeine is as easy to use as a standard local cache implementation.

Coherence provides `caffeine-scheme` configuration element, which can be used anywhere the `local-scheme` element is currently used: standalone, as a definition of a local cache scheme; within the `distributed-scheme` element as a `backing-map` for a partitioned cache; or within the `near-scheme` element as a `front-map`.

Local Cache

```
<caffeine-scheme>
   <scheme-name>caffeine-local-scheme</scheme-name>
</caffeine-scheme>
```

Distributed Cache

```
<distributed-scheme>
    <scheme-name>caffeine-distributed-scheme</scheme-name>
    <backing-map-scheme>
        <caffeine-scheme />
    </backing-map-scheme>
    <autostart>true</autostart>
</distributed-scheme>
```

Near Cache

```
<near-scheme>
    <scheme-name>caffeine-near-scheme</scheme-name>
    <front-scheme>
        <caffeine-scheme />
    </front-scheme>
    <back-scheme>
        <distributed-scheme>
            <scheme-ref>my-dist-scheme</scheme-ref>
        </distributed-scheme>
    </back-scheme>
</near-scheme>
```

You can configure each of the `caffeine-scheme` elements the same way `local-scheme` is configured, by specifying one or more of the following child elements:

Table 14-2 Child Elements Within the caffeine-scheme Elements

Configuration Element	Description
`scheme-name`	The name of this scheme, which you can reference elsewhere in the configuration file.
`scheme-ref`	The reference (by name) to a `caffeine-scheme` defined elsewhere in the configuration file.

Table 14-2 (Cont.) Child Elements Within the caffeine-scheme Elements

Configuration Element	Description
`class-name`	The name of the custom class that extends `com.oracle.coherence.caffeine.CaffeineCache`, allowing you to customize its behavior.
`scope-name`	The name of the scope.
`service-name`	The name of the service.
`init-params`	The arguments to pass to the `class-name` constructor.
`high-units`	The maximum amount of data the cache should be allowed to hold before the eviction occurs.
`unit-calculator`	The unit calculator to use, typically one of the following: • `BINARY`, which determines the number of "units" based on the number of bytes that the serialized form of cache keys and values consume. • `FIXED`, which simply uses the number of entries as "units".
`unit-factor`	Sometimes used in combination with a `BINARY` calculator to overcome the 2GB limit for "units". For example, specifying `1024` as a "unit factor" allows you to express `high-units` in kilobytes instead of in bytes.
`expiry-delay`	The amount of time from last update that the entries are kept in cache before being discarded.
`listener`	A `MapListener` to register with the cache.

All of the configuration elements above are optional, but you will typically want to set either `high-units` or `expiry-delay` (or both) to limit cache based on either size or time-to-live (TTL).

If neither is specified, the cache size is limited only by available memory, and you can specify the TTL explicitly by using the `NamedCache.put(key, value, ttl)` method or by calling `BinaryEntry.expire` within an entry processor.

There is nothing wrong if you do not want to limit the cache by either size or time; you may still benefit from using Caffeine in those situations, especially under high concurrent load, due to its lock-free implementation.

Finally, when using Caffeine as a backing map for a partitioned cache, you will likely want to configure `unit-calculator` to `BINARY`, so you can set the limits and observe cache size (through JMX or Metrics) in bytes instead of the number of entries in the cache.

15

Caching Data Sources

Coherence is commonly used to cache data sources which makes the cache a temporary system-of-record.
This chapter includes the following sections:

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.

This section includes the following topics:

Pluggable Cache Store

A cache store is an application-specific adapter used to connect a cache to an underlying data source. The cache store implementation accesses the data source by using a data access mechanism (for example: Toplink/EclipseLink, JPA, Hibernate, application-specific JDBC calls, another application, mainframe, another cache, and so on). The cache store 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 cache store implementation. See Creating a Cache Store Implementation.

Pre-defined Cache Store Implementations

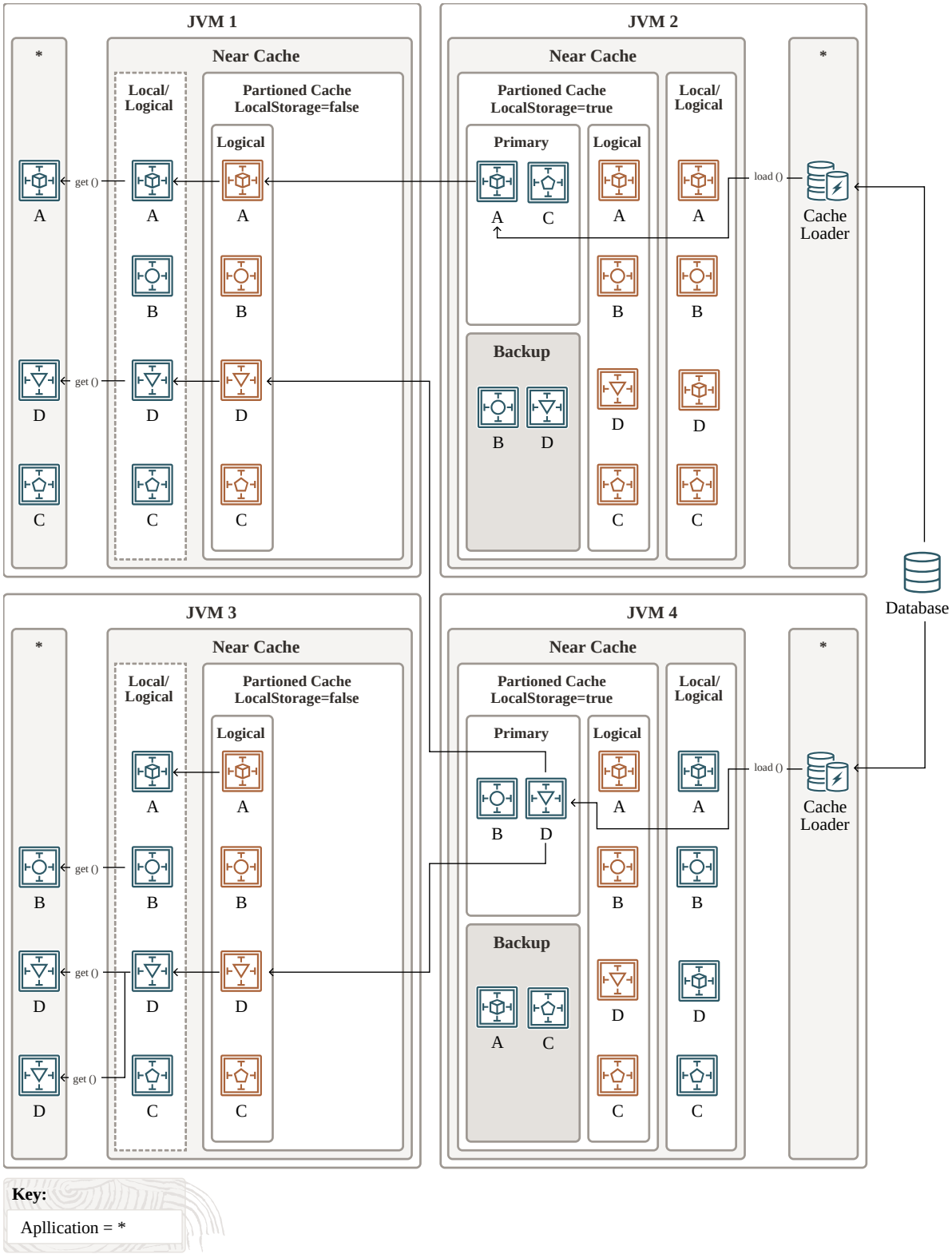
Coherence includes several JPA cache store implementations:

- Generic JPA – a cache store implementation that can be used with any JPA provider. See Using JPA with Coherence.
- EclipseLink – a cache store implementation optimized for the EclipseLink JPA provider. See Using JPA with Coherence.
- Hibernate – a cache store implementation for the Hibernate JPA provider. See Integrating Hibernate and Coherence.

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 asks 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 .

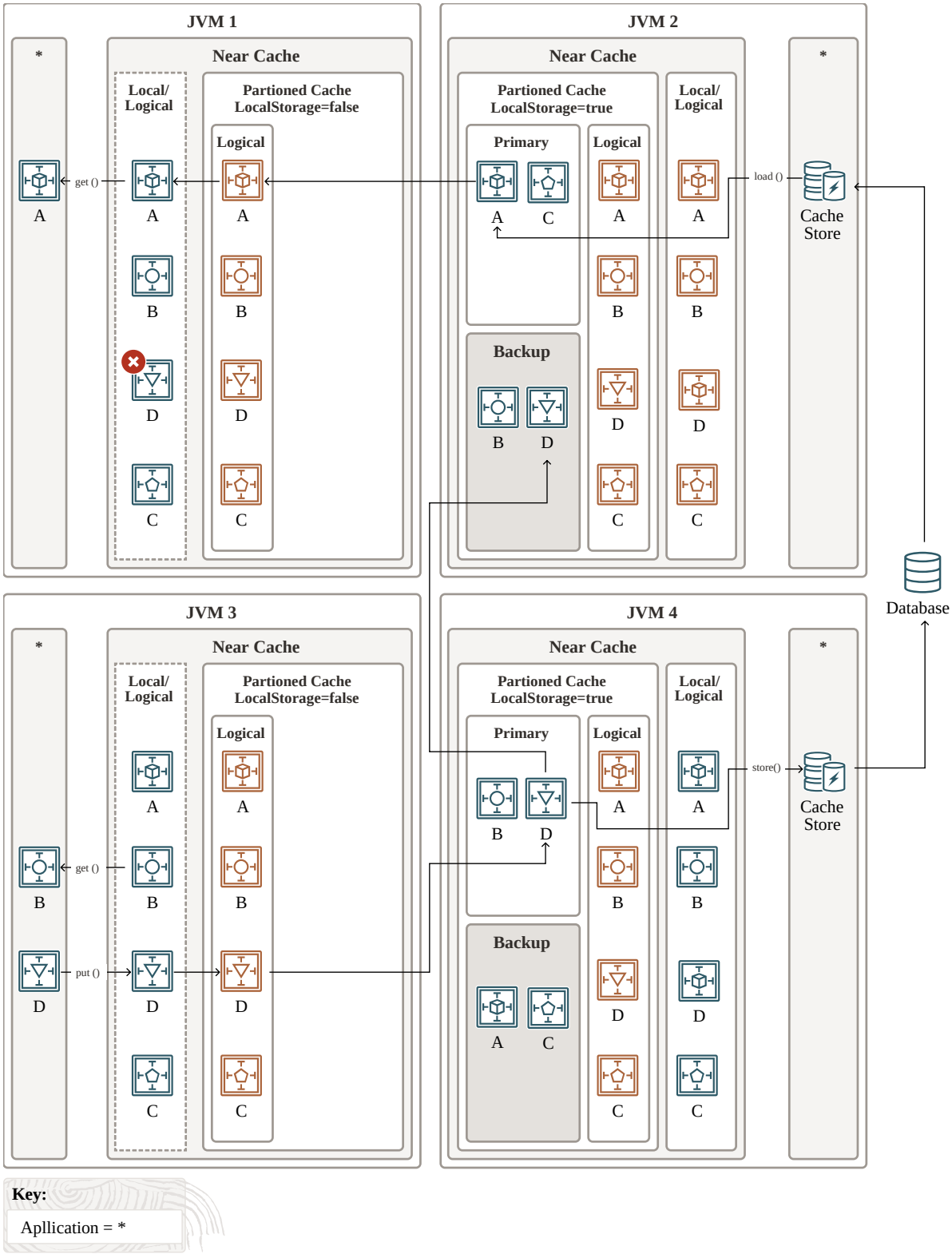
Figure 15-1 Read-Through Caching



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 cache store 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 .

Figure 15-2 Write-Through Caching



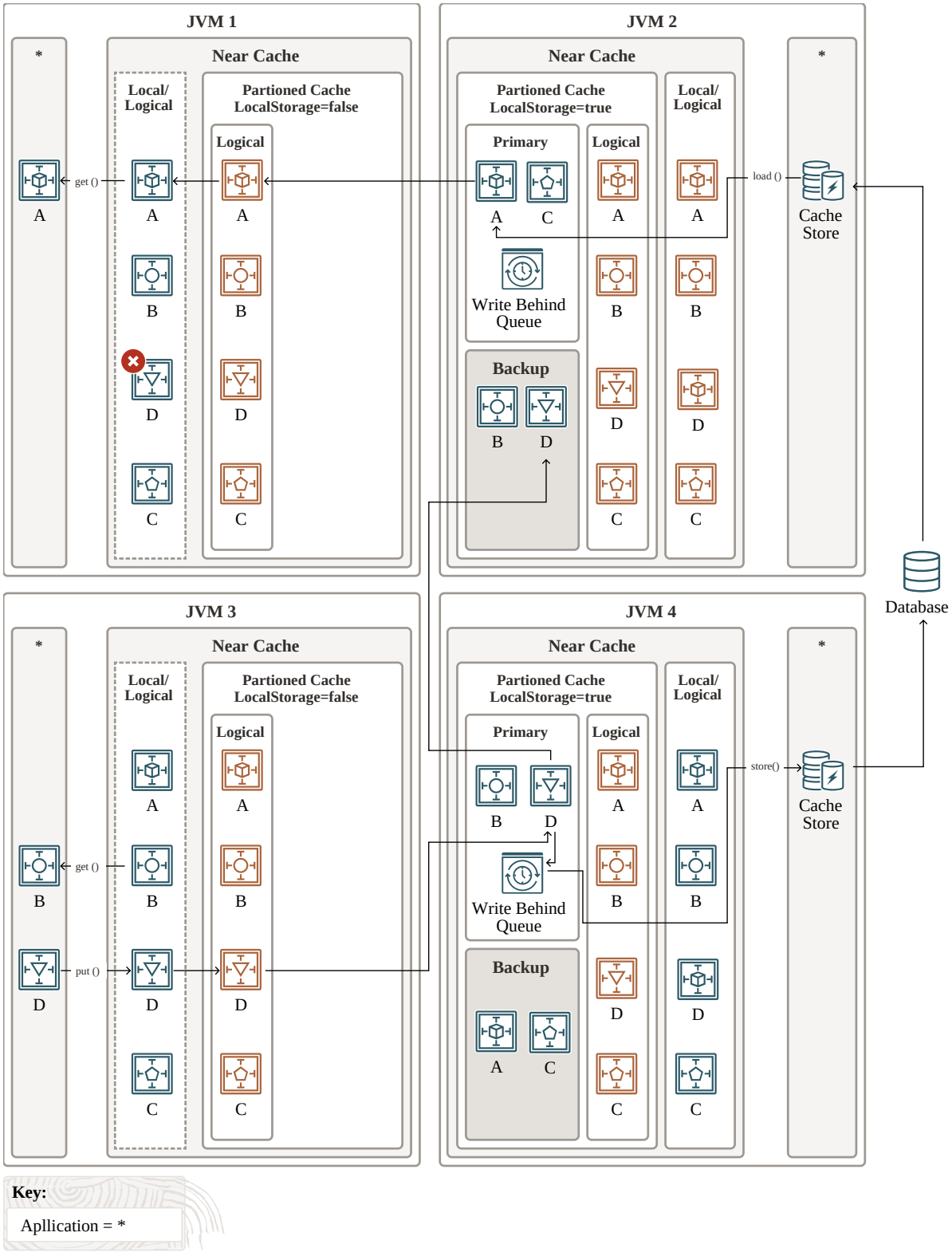
Write-Behind Caching

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 15-3 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. However, this does not result in the whole cache being refreshed before the response is returned. The refresh happens in the background.

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. See read-write-backing-map-scheme. Refresh-ahead assumes that you have also set an expiration time (`<expiry-delay>`) for entries in the cache.

Example 15-1 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 15-1 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>
```

Synchronizing Database Updates with HotCache

The Oracle Coherence GoldenGate HotCache (HotCache) integration allows database changes from sources that are external to an application to be propagated to objects in Coherence caches. The HotCache integration ensures that applications are not using potentially stale or out-of-date cached data. HotCache employs an efficient push model that processes only stale data. Low latency is assured because the data is pushed when the change occurs in the database. The HotCache integration requires installing both Coherence and GoldenGate. See Integrating with Oracle Coherence GoldenGate HotCache.

Non-Blocking Data Sources

Coherence provides a means of integrating with underlying data sources using a number of existing strategies, such as read-through, write-through, write-behind, and refresh-ahead. See Caching Data Sources .

Coherence provides the **`NonBlockingEntryStore`** interface for integrating with data sources that provide non-blocking APIs. See Interface NonBlockingEntryStore. This strategy is similar to write-behind, as it is asynchronous to the original mutation. However, it does not require a queue to defer the call to the store and immediately passes the intent to store to the implementer. The implementer can in turn immediately call the non-blocking API of the data source and on success or failure, a `Future` can pass that information to the provided

`StoreObserver` through `onNext` or `onError`, respectively. The primary methods of the `NonBlockingEntryStore` are highlighted below:

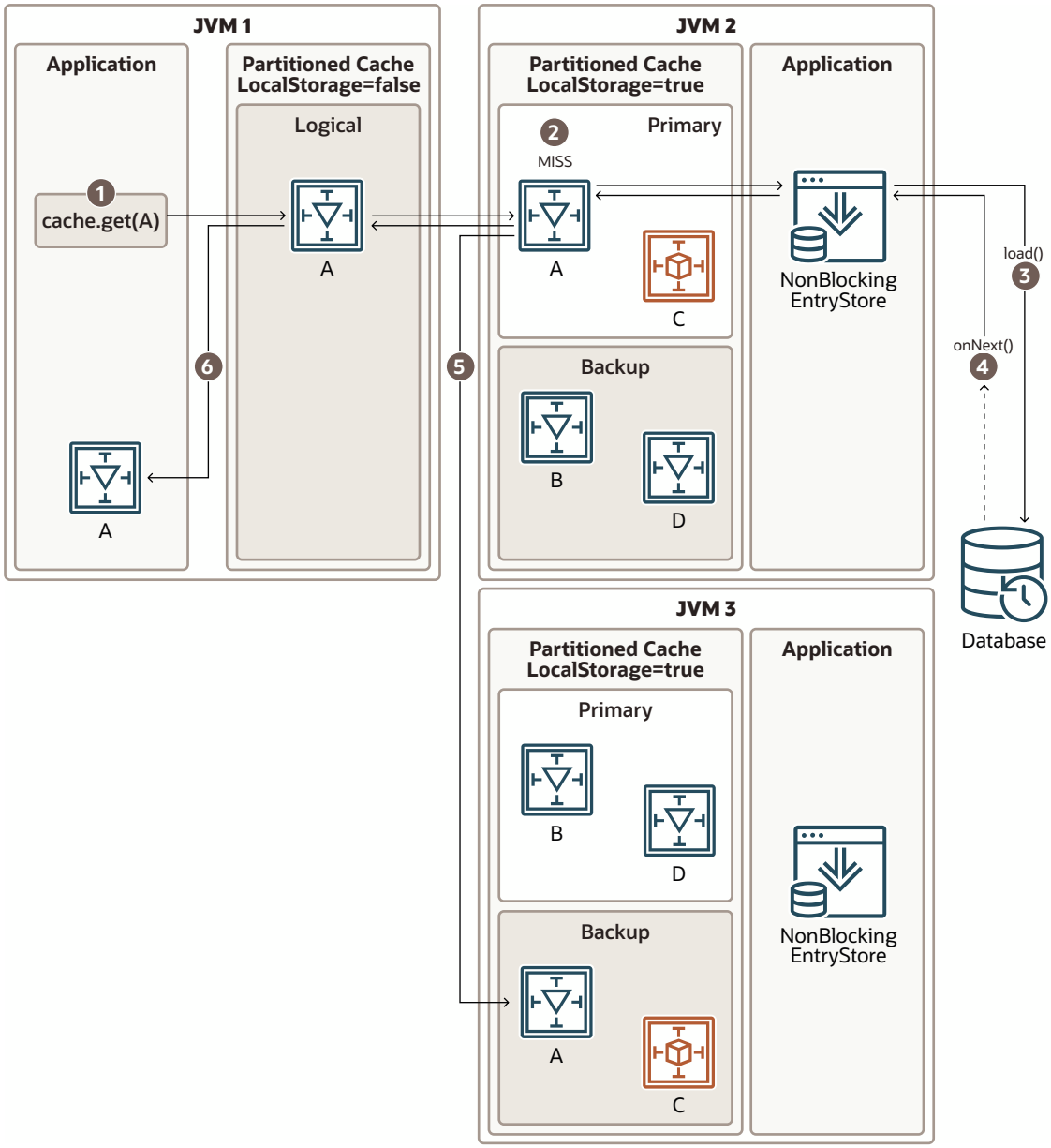
```
public interface NonBlockingEntryStore<K, V>
    {
    public void load(BinaryEntry<K, V> binEntry, StoreObserver<K, V>
observer);

    public void store(BinaryEntry<K, V> binEntry, StoreObserver<K, V>
observer);

    public void erase(BinaryEntry<K, V> binEntry);
    }
```

There are other similar methods in the CacheStore (see Interface CacheStore and BinaryEntryStore (see Interface BinaryEntryStore) interfaces. However, with the `NonBlockingEntryStore` interface, the calls are non-blocking and therefore, Coherence does not expect the operation to be completed when control is returned. To enable the implementer to notify Coherence of operation completion, the `StoreObserver` is provided and must be called upon success or failure. This helps Coherence process the result of the operation.

It is worth pointing out that similar to the write-behind strategy upon failure, and therefore restoration of the primary partitions, Coherence will call `NonBlockingEntryStore.store` for the entries for which it did not receive a success or error notification. This provides at least once semantics and enables implementers to call the non-blocking data source if found necessary.

The diagrams below illustrate the flow from the initial request to the invocation of the `NonBlockingEntryStore` on the storage enabled nodes:

Figure 15-4 For a `get()` Operation Inducing a Load

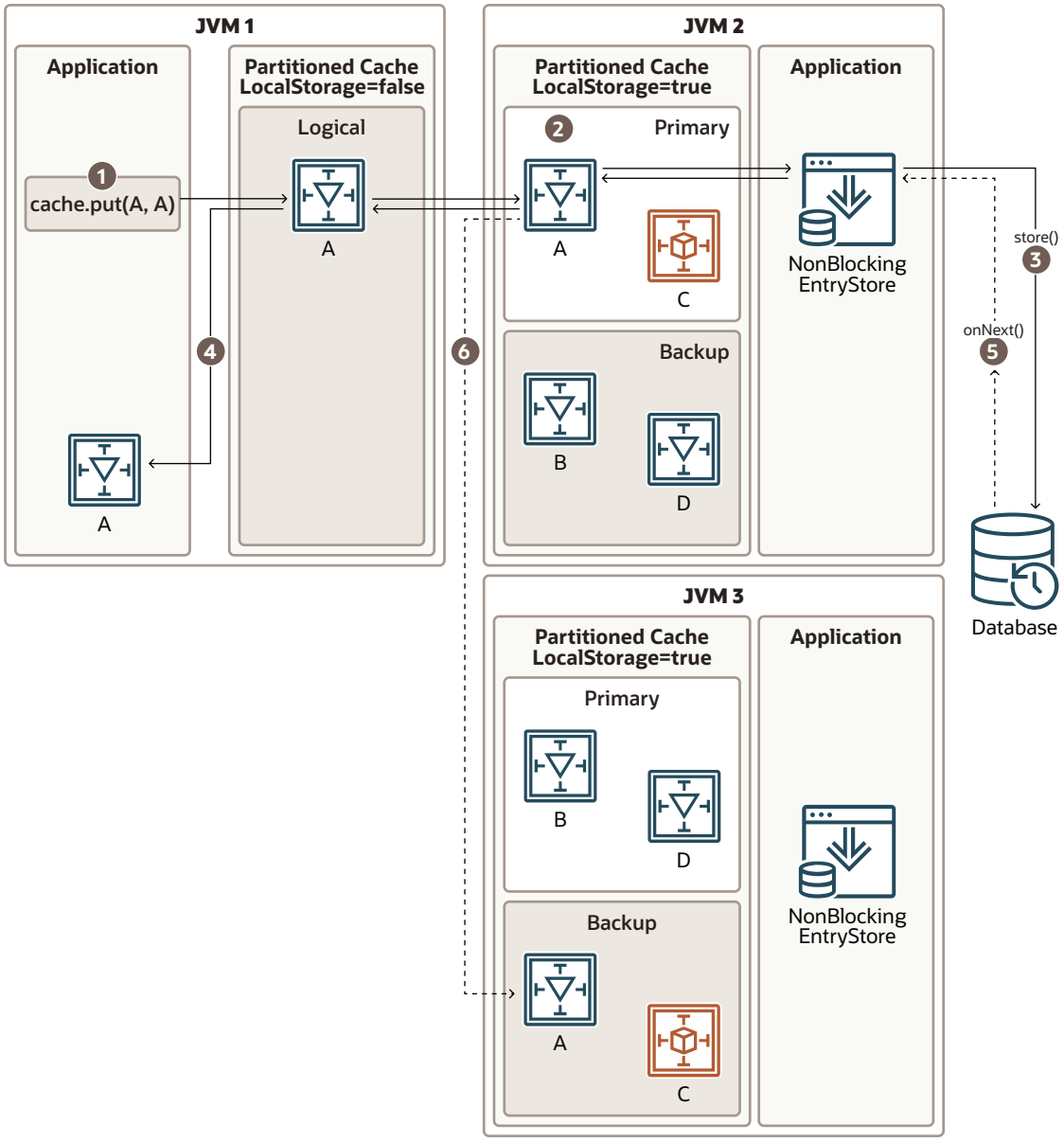


The description of the flow:

- 1 The application calls `get()` on entry `A` that is not yet in the cache.
- 2 A request goes to the storage member that owns the entry, in this instance **JVM2**. Entry ownership, and thus partition ownership, is determined algorithmically based on the raw (or binary) value of the key and the number of partitions that is configured in the associated partitioned service. Because the request has not been accessed yet or has expired, a miss takes place and the call is relayed to the configure entry store.

-  The `load()` operation for the entry store that implements `NonBlockingEntryStore` is called; custom logic is provided a BinaryEntry (see Interface BinaryEntryStore) with a null initial value and a StoreObserver (see Interface StoreObserver). The implementer performs the datastore operation(s) necessary to populate the cache entry.
-  When the operation on the underlying data source completes, the implementation will call either `observer.onNext` or `observer.onError` to determine whether the value was successfully loaded or not. The implementer will update the `BinaryEntry` by using `setValue` or `updateBinaryValue`, prior to calling `onNext`. This will enable Coherence to ensure that data is inserted in the primary partition owner (**JVM2**) and backed up accordingly.
-  The primary partition owner sends the value to another storage member in the cluster for backup purposes.
-  The entry value is sent back to the calling application where a transient reference is kept. Note that although the data source operation can be performed asynchronously and the call to `load()` does not need to wait for its completion to return, the `get()` invocation is synchronous from the caller's perspective.

Figure 15-5 For a put() Operation



The description of the flow:

- 1 The application calls `put()` on entry `A` with value `A`.
- 2 The entry is stored on the owning member.
- 3 Since the cache is configured with a `NonBlockingEntryStore`, the `store()` operation is called. `store()` is provided a `BinaryEntry` and a `StoreObserver`. The implementer performs the datastore operation(s) that are necessary to save the cache entry into a datastore.

-  At this point, the `store()` call of the `NonBlockingEntryStore` can return, and `put()` will then give control back to the calling application.
-  The datastore asynchronously performs the datastore operation(s) that are necessary to save the cache entry into a datastore, and then calls the `observer.onNext()` method for normal operations (or `observer.onError()` in case of a problem). If necessary (for example, the value of the `BinaryEntry` has been updated), the value is put back into the cache.
-  The value is then sent to the backup owning member for safekeeping.

`getAll()`

`getAll()` functions comparably to `get()`, except that `getAll()` processes a set of entries. This provides an opportunity for an implementer to optimize batch operations (multi-entry) against the datasource and thus, reduce the communication overhead with the datasource. After the associated entry is successfully written, the implementer must call `StoreObserver.onNext` to pass the relevant entry (or `onError()` if an error occurred when processing this particular entry).

> **Note:**
>
> Coherence expects all entries to be processed before concluding.

`putAll()`

`putAll()` also functions comparably to `put()`, except on a set of entries. The same expectation is in effect here: either all entries are processed using `onNext()`/`onError()`, or `onComplete()` can be used to interrupt the operation. The difference with `putAll()` is that the caller will not wait for completion and thus, any exception will not be thrown but printed out in the log.

From the application standpoint, the `remove()` operation functions in the same way as `CacheStore` or `BinaryEntryStore`.

Besides providing a natural way of integrating with non-blocking data stores, this model takes advantage of the benefits of such stores in terms of performance and scalability.

- About NonBlockingEntryStore
- Using Non-Blocking Data Sources
- Configuring NonBlockingEntryStore
- Implementing NonBlockingEntryStore

About NonBlockingEntryStore

Certain data source libraries have APIs that do not necessitate the caller to wait for the result to come back before doing something else. For example, making HTTP calls can lead to relatively long waits between the time a request to store data is sent and the response comes back. By implementing non-blocking APIs, the caller can immediately do other work without having to wait for the actual store operation to complete.

By implementing the `NonBlockingEntryStore` interface, the store implementer is able to use non-blocking APIs in a more natural way.

`NonBlockingEntryStore` is being provided in the context of pluggable data stores: in order to use it, an implementation class needs to be provided and configured. This class will either load, store, or remove data from the data source by way of a `ReadWriteBackingMap`. This backing map provides two elements:

- An internal map to cache the data.
- A data source access portion to interact with the data base.

The `NonBlockingEntryStore` interface is provided the `BinaryEntry` that represents the `load`, `store`, or `erase` operation. This provides an opportunity for implementers to avoid deserialization, if required. This is similar to `BinaryEntryStore`. Avoiding deserialization is possible if the raw binary is stored in the downstream system, or the binary can be navigated to extract relevant parts, as opposed to deserializing the entire key or value.

> **Note:**
>
> `getKey`, `getValue`, and `getOriginalValue` induce deserialization for the first call.

Using Non-Blocking Data Sources

This section provides a summary of the tasks required to use this feature. To get started, see the Non Blocking Entry Store Example in Cache Stores.

Configuring NonBlockingEntryStore

To specify a non-blocking cache store implementation, provide the implementation class name within the read-write-backing-map-scheme as shown below:

```
...
<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">
    <cache-mapping>
...
        <cache-name>myCache</cache-name>
        <scheme-name>distributed-rwbm-nonblocking</scheme-name>
...
    </cache-mapping>

    <distributed-scheme>
...
        <scheme-name>distributed-rwbm-nonblocking</scheme-name>
        <backing-map-scheme>
            <read-write-backing-map-scheme>

                <cachestore-scheme>
                    <class-scheme>
                        <class-name>com.company.NonBlockingStoreImpl</class-
name>
```

```
                    </class-scheme>
                </cachestore-scheme>

            </read-write-backing-map-scheme>
        </backing-map-scheme>
        <autostart>true</autostart>
...
    </distributed-scheme>

</cache-config>
```

Implementing NonBlockingEntryStore

After you configure, add a class implementing the `NonBlockingEntryStore` interface to the classpath of the storage enabled members. See the example code below.

With the class in place, the equivalency below is established:

- `get()` - invokes→ `load()`

> **Note:**
>
> If data is already in the cache, `load()` does not get called. Also, calling `get()` will wait for `onNext()`/`onError()` to complete the request before returning.

- `getAll()` - invokes→ `loadAll()`
- `put()` - invokes→ `store()`
- `putAll()` - invokes→ `storeAll()`
- `remove()` - invokes→ `erase()`
- `removeAll()` - invokes→ `eraseAll()`

The code snippet below shows the use of a reactive API to access a data source.

```
...
/**
 * An example NonBlockingEntryStore implementation
 */
public class ExampleNonBlockingEntryStore<K, V>
    {
    @Override
    public void load(BinaryEntry<K, V> binEntry, StoreObserver<K, V> observer)
        {
        K key = binEntry.getKey();

        Flux.from(getConnection())
                .flatMap(connection -> connection.createStatement(LOAD_STMT)
                        .bind("$1", key)
                        .execute())
                .flatMap(result ->
                         result.map((row, meta) ->
                                 {
                                 returnnew Student(
```

```
                                        (String) row.get("name"),
                                        (String) row.get("address"));
                                }
                        ))
                .collectList()
                .doOnNext(s ->
                        {
                        binEntry.setValue((V) s.get(0));
                        observer.onNext(binEntry);
                        })
                .doOnError(t ->
                        {
                        if (t instanceof IndexOutOfBoundsException)
                            {
                            CacheFactory.log("Could not find row for key:
" + key);
                            }
                        else
                            {
                            CacheFactory.log("Error: " + t);
                            }
                        observer.onError(binEntry, new Exception(t));
                        })
                .subscribe();
        }
...
    @Override
    public void store(BinaryEntry<K, V> binEntry, StoreObserver<K, V>
observer)
        {
        K       key      = binEntry.getKey();
        Student oStudent = (Student) binEntry.getValue();

        Flux.from(getConnection())
                .flatMap(connection -> connection.createStatement(STORE_STMT)
                        .bind("$1", key)
                        .bind("$2", oStudent.getName())
                        .bind("$3", oStudent.getAddress())
                        .execute())
                .flatMap(Result::getRowsUpdated)
                .doOnNext((s) ->
                        {
                        CacheFactory.log("store done, rows updated: " + s);
                        observer.onNext(binEntry);
                        })
                .doOnError(t -> new Exception(t))
                .subscribe();
        }
...
    privatestaticfinal String STORE_STMT = "INSERT INTO student VALUES
($1, $2, $3) ON conflict (id) DO UPDATE SET name=$2, address=$2";
    privatestaticfinal String LOAD_STMT = "SELECT NAME, ADDRESS FROM student
WHERE id=$1";
```

Ensure that you use best practices when implementing an entry store for your data sources. See Implementation Considerations.

Selecting a Cache Strategy

Compare and contrast the different data source caching strategies that Coherence supports. This section includes the following topics:

- 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 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 which means 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 Cache Store Implementation

Coherence provides several cache store interfaces that can be used depending on how a cache uses a data source.To create a cache store, implement one of the following interfaces:

- `CacheLoader` – read-only caches
- `CacheStore` – read/write caches

- `BinaryEntryStore` – read/write for binary entry objects.

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)`. The `CacheStore` interface adds the methods `store(Object key, Object value)`, `storeAll(Map mapEntries)`, `erase(Object key)`, and `eraseAll(Collection colKeys)`. The `BinaryEntryStore` interface provides the same methods as the other interfaces, but it works directly on binary objects.

See Sample Cache Store Implementation and Sample Controllable Cache Store Implementation.

Plugging in a Cache Store Implementation

To plug in a cache store implementation, specify the implementation class name within a `distributed-scheme`, `backing-map-scheme`, `cachestore-scheme`, or `read-write-backing-map-scheme`.
The `read-write-backing-map-scheme` configures the `ReadWriteBackingMap` implementation. This backing map is composed of two key elements: an internal map that actually caches the data (`internal-cache-scheme`), and a cache store implementation that interacts with the database (`cachestore-scheme`). See read-write-backing-map-scheme and take note of the `write-batch-factor`, `refresh-ahead-factor`, `write-requeue-threshold`, and `rollback-cachestore-failures` elements.

Example 15-2 illustrates a cache configuration that specifies a cache store implementation. The `<init-params>` element contains an ordered list of parameters that is passed into the constructor. The `{cache-name}` configuration macro is used to pass the cache name into the implementation, allowing it to be mapped to a database table. See Using Parameter Macros.

Example 15-2 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.example.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>
```

Using Federated Caching with Cache Stores

By default, data that is loaded into a cache store is not federated to federation participants. Care should be taken when enabling federated caching because any read-through requests on the local participant is federated to the remote participant and results in an update to the data source on the remote site. This may be especially problematic for applications that are timing sensitive. In addition, applications that are read heavy can potentially cause unnecessary writes to the data source on the remote site.

To federate entries that are loaded into a cache when using read-through caching, set the `<federated-loading>` element to `true` in the cache store definition. For example:

```
<cachestore-scheme>
   <class-scheme>
      <class-name>com.example.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>
   <federated-loading>true</federated-loading>
</cachestore-scheme>
```

Sample Cache Store Implementation

Before creating a cache store implementation, review a very basic sample implementation of the `com.tangosol.net.cache.CacheStore` interface.

The implementation in Example 15-3 uses a single database connection 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. See Cache of a Database for 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. See Pre-Loading a Cache.

Example 15-3 Implementation of the CacheStore Interface

```
package com.tangosol.examples.coherence;
```

```
import com.tangosol.net.AbstractCacheStore;

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 CacheStore implementation.
*/
public class DBCacheStore
        extends AbstractCacheStore
    {
    // ----- 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);
        }
    }

/**
* 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 Cache Store Implementation

You can implement a controllable cache store that allows an application to 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 15-4 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 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` interface can be used to indicate that an object added to the cache should not be stored in the database. See Cache of a Database.

Example 15-4 provides a listing of the `Main.java` interface.

Example 15-4 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();
            }
        }

    }
```

Implementation Considerations

Consider best practices when creating a cache store implementation. This section includes the following topics:

- Idempotency
- Write-Through Limitations
- Cache Queries
- Re-entrant Calls
- Cache Server Classpath
- CacheStore Collection Operations
- Connection Pools

Idempotency

All 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 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 operations across multiple cache store instances. In other words, if two cache entries are updated, triggering calls to cache store implementations that are 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 a cache store implementation to load any missing (or potentially missing) data. Therefore, applications that query cache store-backed caches should ensure that all necessary data required for the queries has been pre-loaded. See Querying Data in a Cache. 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 cache store pattern to disable circular updates back to the database. See Sample Controllable Cache Store Implementation. The cache store 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 View cache (a different cache service) and reading it in every cache store implementation 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 cache store 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 cache store.

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).

16

Serialization Paged Cache

Coherence supports caching large amounts of binary data off-heap.
This chapter includes the following sections:

Understanding Serialization Paged Cache

Coherence provides 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 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.

Serialization paged caches are configured within the `<external-scheme>` and `<paged-external-scheme>` element in the cache configuration file. See external-scheme and paged-external-scheme.

Optimizing a Partitioned Cache Service

Coherence provides an optimization for the partitioned cache service which takes advantage of the fact that 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 external-scheme and paged-external-scheme.

Configuring for High Availability

Serialization Paged Cache includes support 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 paged-external-scheme.

Configuring Load Balancing and Failover

When using a serialization paged cache 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 external-scheme and paged-external-scheme.

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.)

17

Using Quorum

Coherence provides quorum policies that control when specific service actions are allowed in a cluster in order to ensure that a cluster is adequately provisioned.
This chapter includes the following sections:

Overview of Using Quorum

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 ensures a cluster always contains a configured number of cluster members.
This section includes the following topics:

Overview of the Cluster Quorum Policy

The cluster quorum policy defines quorums for the Cluster service that mandates when a cluster can terminate 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. These quorums can be specified generically across all cluster members or

constrained to members that have a specific role in the cluster, such as client or server members. See member-identity.

- **Timeout Site Quorum** – The timeout site quorum mandates the minimum number of sites that must remain in the cluster when the cluster service is terminating suspect members.
- **Timeout Machine Quorum** – The timeout machine quorum mandates the minimum number of cluster machines that must remain in the cluster when the cluster service is terminating suspect members.
- **Timeout Survivor Quorum** – 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.

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 these quorums, 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 machines or 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

Cluster quorums are configured in an operational override file within the `<cluster-quorum-policy>` element. Each quorum can optionally use a `role` attribute. The following example demonstrates configuring the quorums together and ensures that `5` cluster members, `2` sites, and `3` machines are always kept in the cluster while removing suspect members. In addition, the quorums are specific to members with the `server` role.

```
<cluster-config>
   <member-identity>
      <role-name>server</role-name>
   </member-identity>
   <cluster-quorum-policy>
      <timeout-site-quorum role="server">2</timeout-site-quorum>
      <timeout-machine-quorum role="server">3</timeout-machine-quorum>
      <timeout-survivor-quorum role="server">5</timeout-survivor-quorum>
   </cluster-quorum-policy>
</cluster-config>
```

Using the Partitioned Cache Quorums

The partitioned cache quorum policy ensures that a partitioned cache service contains a configured number of members to perform service operations.
This section includes the following topics:

- Overview of the Partitioned Cache Quorum Policy
- Configuring the Partitioned Cache Quorum Policy

Overview of the Partitioned Cache Quorum Policy

The partitioned cache quorum policy defines 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.
- **Recover Quorum** – This quorum mandates the minimum number of storage-enabled members of a partitioned cache service that must be present to recover orphaned partitions from the persistent storage, or assign empty partitions if the persistent storage is unavailable or lost. The members must be defined in a host-address list that can be referenced as part of the quorum definition.

> **Note:**
>
> The Recover Quorum supports a value of zero. A value of zero enables the dynamic recovery policy, which ensures availability of all persisted state and is based on the last good cluster membership information to determine how many members must be present for the recovery. See Using Quorum for Persistence Recovery in *Administering Oracle Coherence*.

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>
      <recover-quorum>2</recover-quorum>
      <recovery-hosts>persistence-host-list</recovery-hosts>
   </partitioned-quorum-policy-scheme>
   <autostart>true</autostart>
</distributed-scheme>
```

The `<partitioned-quorum-policy-scheme>` element does not support scheme references. However, you can reference the scheme as `<distributed-scheme>`, as shown in the following 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">
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>*</cache-name>
      <scheme-name>dist-example</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>
  <caching-schemes>
    <distributed-scheme>
      <scheme-name>dist-example</scheme-name>
      <scheme-ref>partitioned-cache-with-quorum</scheme-ref>
       <service-name>DistExample</service-name>
       <backing-map-scheme>
         <local-scheme>
           <high-units>100M</high-units>
         </local-scheme>
       </backing-map-scheme>
       <autostart>true</autostart>
    </distributed-scheme>
    <distributed-scheme>
       <scheme-name>partitioned-cache-with-quorum</scheme-name>
       <service-name>PartitionedCacheWithQuorum</service-name>
       <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>
    </distributed-scheme>
  </caching-schemes>
</cache-config>
```

Using the Proxy Quorum

The proxy quorum policy ensures a cluster always contains enough proxies to service client requests.
This section includes the following topics:

Overview of the Proxy Quorum Policy

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>
```

Using 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.
This section includes the following topics:

- Enabling Custom Action Policies
- Enabling the Custom Failover Access Policy

Enabling Custom Action Policies

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>
```

Enabling the Custom Failover Access Policy

Coherence provides a pre-defined custom action policy that moderates client request load during a failover event in order to allow cache servers adequate opportunity to re-establish partition backups. Use this policy in situations where a heavy load of high-latency requests may prevent, or significantly delay, cache servers from successfully acquiring exclusive access to partitions needing to be transferred or backed up.

To enable the custom failover access policy, add a `<class-name>` element within a `<partition-quorum-policy-scheme>` element that contains the fully qualified name of the failover access policy (`com.tangosol.net.partition.FailoverAccessPolicy`). The policy accepts the following parameters:

- `cThresholdMillis` – Specifies the delay before the policy should start holding requests (after becoming endangered). The default value is `5000` milliseconds.
- `cLimitMillis` – Specifies the delay before the policy makes a maximal effort to hold requests (after becoming endangered). The default values is `60000` milliseconds.
- `cMaxDelayMillis` – Specifies the maximum amount of time to hold a request. The default value is `5000` milliseconds.

The following example enables the custom failover access policy and sets each of the parameters:

```
<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>com.tangosol.net.partition.FailoverAccessPolicy/class-name>
      <init-params>
         <init-param>
            <param-name>cThresholdMillis</param-name>
            <param-value>7000</param-value>
         </init-param>
         <init-param>
            <param-name>cLimitMillis</param-name>
            <param-value>30000</param-value>
         </init-param>
         <init-param>
            <param-name>cMaxDelayMillis</param-name>
            <param-value>2000</param-value>
         </init-param>
      </init-params>
   </partitioned-quorum-policy-scheme>
   <autostart>true</autostart>
</distributed-scheme>
```

18

Cache Configurations by Example

You can learn how to configure Coherence caches by reviewing a series of sample cache scheme definitions that can be used or modified as required.See Configuring Caches for detailed instructions on how to configure caches. The samples build upon one another and often use a `<scheme-ref>` element to reuse other samples as nested schemes. See Using Scheme Inheritance. Lastly, the 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 chapter contains the following sections:

- Local Caches (accessible from a single JVM)
- Clustered Caches (accessible from multiple JVMs)

Local Caches (accessible from a single JVM)

Use the local cache scheme samples to learn how to configure caches that are accessible by a single JVM.Local caches are used as building blocks for clustered caches. See Clustered Caches (accessible from multiple JVMs).
This section contains the following topics:

- In-memory Cache
- Size Limited In-memory Cache
- In-memory Cache with Expiring Entries
- In-memory Cache with Disk Based Overflow
- Cache on Disk
- Size Limited Cache on Disk
- Persistent Cache on Disk
- Cache of a Database

In-memory Cache

Example 18-1 uses a local-scheme element to define an in-memory cache. The cache stores as much as the JVM heap allows.

Example 18-1 Configuration for a Local, In-memory Cache

```
<local-scheme>
   <scheme-name>SampleMemoryScheme</scheme-name>
</local-scheme>
```

Size Limited In-memory Cache

Adding a `<high-units>` subelement within the `<local-scheme>` element 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. See local-scheme.

Example 18-2 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 within the `<local-scheme>` element 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. See local-scheme.

Example 18-3 Configuration for an In-memory Cache with Expiring Entries

```
<local-scheme>
   <scheme-name>SampleMemoryExpirationScheme</scheme-name>
   <expiry-delay>5m</expiry-delay>
</local-scheme>
```

In-memory Cache with Disk Based Overflow

Example 18-4 uses an overflow-scheme element 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 to implement the front and back of the cache.

Example 18-4 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 on Disk

Example 18-5 uses an external-scheme element to define an on-disk cache.

> **Note:**
>
> This example uses the bdb-store-manager (Berkeley Database) for its on disk storage implementation. See external-scheme for additional external storage options.

Example 18-5 Configuration to Define a Cache on Disk

```
<external-scheme>
  <scheme-name>SampleDiskScheme</scheme-name>
  <bdb-store-manager/>
</external-scheme>
```

Size Limited Cache on Disk

Adding a `<high-units>` subelement to an external-scheme element 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. See paged-external-scheme for an alternate size-limited external caching approach.

Example 18-6 Configuration for a Size Limited Cache on Disk

```
<external-scheme>
  <scheme-name>SampleDiskLimitedScheme</scheme-name>
  <bdb-store-manager/>
  <high-units>1000000</high-units>
</external-scheme>
```

Persistent Cache on Disk

Example 18-7 uses an external-scheme element to define 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 in non-clustered caches when using either of the bdb-store-manager storage managers. See Persistence (long-term storage) . For clustered persistence, see Partitioned Cache of a Database.

The `{cache-name}` macro is used to specify the name of the file the data is stored in. See Using Parameter Macros.

Example 18-7 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>
```

Cache of a Database

Example 18-8 uses a read-write-backing-map-scheme element 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 Cache Store Implementation.

The `{cache-name}` macro is used to inform the cache store implementation of the name of the cache it backs. See Using Parameter Macros.

Example 18-8 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)

Use the clustered cache scheme samples to learn how to configure clustered caches that 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) . The cache service provides the capability to access local caches from other cluster nodes.
This section contains the following topics:

- Partitioned Cache
- Partitioned Cache with Overflow
- Partitioned Cache with Journal Storage
- Partitioned Cache of a Database
- Partitioned Cache with a Serializer
- Partitioned Cache with Persistence
- Near Cache
- View Cache

Partitioned Cache

Example 18-9 uses the distributed-scheme element to define a clustered cache in which cache storage is partitioned across all cluster nodes.

An In-memory Cache 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.

Example 18-9 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 can use a scheme reference to specify any of the other local cache samples. For instance, if it had used the In-memory Cache with Disk Based Overflow, 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 18-10 Configuration for a Partitioned Cache with Overflow

```
<distributed-scheme>
  <scheme-name>SamplePartitionedOverflowScheme</scheme-name>
  <backup-storage>
    <type>scheme</type>
    <scheme-name>SampleOverflowScheme</scheme-name>
  </backup-storage>
  <backing-map-scheme>
    <overflow-scheme>
      <scheme-ref>SampleOverflowScheme</scheme-ref>
    </overflow-scheme>
  </backing-map-scheme>
  </distributed-scheme>
```

Partitioned Cache with Journal Storage

Example 18-11 uses the backing-map-scheme element to define a partitioned cache that uses a RAM journal for the backing map. The RAM journal automatically delegates to a flash journal when the RAM journal exceeds a configured memory size. See Defining Journal Schemes.

Example 18-11 Configuration for a Partitioned Cache with RAM Journaling

```
<distributed-scheme>
   <scheme-name>SamplePartitionedJournalScheme</scheme-name>
      <backing-map-scheme>
         <ramjournal-scheme/>
      </backing-map-scheme>
   <autostart>true</autostart>
</distributed-scheme>
```

Example 18-12 defines a partitioned cache that directly uses a flash journal for the backing map.

Example 18-12 Configuration for a Partitioned Cache with Flash Journaling

```
<distributed-scheme>
   <scheme-name>SamplePartitionedJournalScheme</scheme-name>
      <backing-map-scheme>
         <flashjournal-scheme/>
      </backing-map-scheme>
   <autostart>true</autostart>
</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 18-13 reuses Cache of a Database to define database access.

Example 18-13 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>
      <internal-cache-scheme>
         <local-scheme>
            <scheme-ref>SampleMemoryScheme</scheme-ref>
         </local-scheme>
      </internal-cache-scheme>
    </read-write-backing-map-scheme>
  </backing-map-scheme>
</distributed-scheme>
```

Partitioned Cache with a Serializer

Example 18-14 uses the serializer element 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 POF User Type Configuration Elements.

Example 18-14 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 18-15 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 18-16 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>
   ...
```

Partitioned Cache with Persistence

Example 18-17 uses the `<persistence-environment>` element to define a partitioned cache that uses asynchronous persistence to allow the storage servers to persist data asynchronously. The asynchronous persistence ensures that writes are not blocked on latency of writing. See Using Asynchronous Persistence.

You can enable asynchronous persistence in cache configuration by specifying the persistence environment of the `<distributed-scheme>` element.

Example 18-17 Configuration for a Partitioned Cache with Persistence

```
<distributed-scheme>
  <scheme-name>SamplePartitionedCachePersistence</scheme-name>
  <persistence>
        <environment>async-environment</environment>
  </persistence>
</distributed-scheme>
```

Near Cache

Example 18-18 uses the near-scheme element 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.

The Size Limited In-memory Cache sample is reused to define the near (`<front-scheme>`) cache, while the Partitioned Cache sample is reused to define the back (`<back-scheme>`) cache.

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 18-18 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>
```

View Cache

A view cache is configured by using the view-scheme element in the coherence-cache-config file. The `<view-scheme>` element has only one required sub-element, which is the name of the scheme. If no other configuration elements are applied, the result lists a local view of all the entries of an existing cache. By default, all entries of the back cache will be visible within the view. However, it is possible to restrict the result set by providing a Coherence filter through the `<filter>` element.

Entries can have transformations applied prior to insertion into the view, by specifying a custom `ValueExtractor` with the `<view-scheme>` element. Applying transformation marks view as `read-only` and thus disallows mutations. Additionally, if the view is being transformed and values are not being stored then operations such as aggregations will not be possible and result in a runtime error.

A `MapListener` could be declared to allow interception of all map events including those generated while materializing the view.

The `<read-only>` element disables all mutations of the back cache through the view. However, modifications could still occur on the back cache directly.

The `<reconnect-interval>` specified in milliseconds, indicates the period in which re-synchronization with the underlying cache will be delayed in the case the connection is severed. During this time period, local content can be accessed without triggering re-synchronization of the local content. A value of zero means that the view cannot be used when not connected.

The `<back-scheme>` element refers to a distributed or remote scheme (if using Coherence*Extend).

Example 18-19 Configuration for a View Cache

```
<view-scheme>
     <scheme-name>view</scheme-name>
</view-scheme>
```

This example is a minimum configuration of a view cache. A view will be materialized on the default view cache service with no filtering, listeners, or transformation of entries. It is a complete local copy of an existing cache.

Example 18-20 Configuration for a View With a Custom Filter

```
<view-scheme>
   <scheme-name>view-customers-over-40</scheme-name>
   <back-scheme>
      <distributed-scheme>
         <scheme-ref>partitioned-std</scheme-ref>
      </distributed-scheme>
   </back-scheme>
   <view-filter>
```

```
      <class-scheme>
         <class-name>com.tangosol.util.filter.GreaterFilter</class-name>
         <init-params>
            <init-param>
               <param-type>java.lang.String</param-type>
               <param-value>age</param-value>
            </init-param>
            <init-param>
               <param-type>java.lang.Integer</param-type>
               <param-value>40</param-value>
            </init-param>
         </init-params>
      </class-scheme>
   </view-filter>
</view-scheme>
```

In this example, the view uses partitioned cache from the `partitioned-std` cache service and only include entries of customers that are over the age of 40.

19

Using Cache Configuration Override

Coherence 14c (14.1.2.0.0) introduces the cache configuration override, a feature similar to the Coherence Cluster override. You can specify the cache configuration override by using the `xml-override` attribute in the root element of cache configuration. The root element is referred to as the "base" cache configuration file whose elements you can now override by placing the override file in the classpath or a module path. If the specified XML override file is found on the classpath or a module path, Coherence loads the file at runtime and overrides the base configuration based on each module's specific requirements.
This chapter includes the following topics:

- Specifying the xml-override Attribute
- Use Cases for Overriding Coherence Cache Configuration
  The use cases in this section describe the different scenarios for overriding Coherence cache configuration.

Specifying the xml-override Attribute

Based on the `xml-override` attribute you specify, Coherence loads the `cache-config-override.xml` file and applies when it is found on the classpath or a module path of the application when you do not specify the `coherence.cacheconfig.override` system property. If you specify `coherence.cacheconfig.override`, Coherence loads the XML override pointed by this system property and applies it at runtime.
The following is a snippet of the cache config XML file with the `xml-override` attribute specified:

```
<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"
              xml-override="{coherence.cacheconfig.override cache-config-
override.xml}">
    ....
</cache-config>
```

Use Cases for Overriding Coherence Cache Configuration

The use cases in this section describe the different scenarios for overriding Coherence cache configuration.

- **Cache configuration override has `caching-scheme-mapping`/`cache-mapping` that already exists in the base cache configuration with the same `cache-name`**
  In this case, the `cache-mapping` element within `caching-scheme-mapping` is overridden from the override file, as shown in the following example:

- **Base XML**

```
<cache-mapping>
    <cache-name>my-cache-*</cache-name>
    <scheme-name>my-cache-scheme-parent</scheme-name>
</cache-mapping>
```

- **Override XML**

```
<cache-mapping>
    <cache-name>my-cache-*</cache-name>
    <scheme-name>my-cache-scheme-override</scheme-name>
</cache-mapping>
```

- **The effective `<caching-scheme-mapping>` element after the application of the XML override to the base XML**

```
<caching-scheme-mapping>
    <cache-mapping>
        <cache-name>my-cache-*</cache-name>
        <scheme-name>my-cache-scheme-override</scheme-name>
    </cache-mapping>
</caching-scheme-mapping>
```

- **Cache configuration override has `caching-scheme-mapping/cache-mapping` that does not exist in the base cache configuration**
  In this case, the new `<cache-mapping>` element from the override file prepends to the other elements within `caching-scheme-mapping`, as shown in the following example:

  - **Base XML**

```
<cache-mapping>
    <cache-name>my-cache-*</cache-name>
    <scheme-name>my-cache-scheme-parent</scheme-name>
</cache-mapping>
```

  - **Override XML**

```
<cache-mapping>
    <cache-name>my-cache-override</cache-name>
    <scheme-name>my-cache-scheme-override</scheme-name>
</cache-mapping>
```

  - **The effective `<caching-scheme-mapping>` element after the application of the XML override to the base XML**

```
<caching-scheme-mapping>
    <cache-mapping>
        <cache-name>my-cache-override</cache-name>
        <scheme-name>my-cache-scheme-override</scheme-name>
    </cache-mapping>
    <cache-mapping>
        <cache-name>my-cache-*</cache-name>
        <scheme-name>my-cache-scheme-parent</scheme-name>
```

```
    </cache-mapping>
</caching-scheme-mapping>
```

- **Cache configuration override has `caching-scheme` with the same `scheme-name` that exists in the base configuration file**
  Only a `caching-scheme` with the given `scheme-name` is merged with the relevant content from the override file, as shown in the following example:
  - **Base XML**

```
<caching-schemes>
   <distributed-scheme>
      <scheme-name>my-cache-scheme</scheme-name>
      <service-name>MyCache</service-name>
      <backing-map-scheme>
         <local-scheme>
            <unit-calculator>BINARY</unit-calculator>
         </local-scheme>
      </backing-map-scheme>
      <partitioned-quorum-policy-scheme>
         <write-quorum>3</write-quorum>
      </partitioned-quorum-policy-scheme>
      <autostart>true</autostart>
   </distributed-scheme>
   <distributed-scheme>
      <scheme-name>my-cache-scheme-two</scheme-name>
      <service-name>MyCache2</service-name>
   </distributed-scheme>
<caching-schemes>
```

  - **Override XML**

```
<caching-schemes>
    <distributed-scheme>
      <scheme-name>my-cache-scheme</scheme-name>
      <partitioned-quorum-policy-scheme>
         <write-quorum>1</write-quorum>
      </partitioned-quorum-policy-scheme>
    </distributed-scheme>
</caching-schemes>
```

  - **The effective `<caching-schemes>` element after the application of the XML override**

```
<caching-schemes>
   <distributed-scheme>
      <scheme-name>my-cache-scheme</scheme-name>
      <service-name>MyCache</service-name>
      <backing-map-scheme>
         <local-scheme>
            <unit-calculator>BINARY</unit-calculator>
         </local-scheme>
      </backing-map-scheme>
      <partitioned-quorum-policy-scheme>
         <write-quorum>1</write-quorum>
      </partitioned-quorum-policy-scheme>
```

```
      <autostart>true</autostart>
   </distributed-scheme>
   <distributed-scheme>
      <scheme-name>my-cache-scheme-two</scheme-name>
      <service-name>MyCache2</service-name>
   </distributed-scheme>
<caching-schemes>
```

- **Cache configuration override with `caching-schemes` without any `scheme-name`**
  All types of that `caching-scheme` is modified with the relevant content from the override file, as shown in the following example:
  - **Base XML**

```
<caching-schemes>
   <distributed-scheme>
      <scheme-name>my-cache-scheme</scheme-name>
      <service-name>MyCache</service-name>
      <backing-map-scheme>
         <local-scheme>
            <unit-calculator>BINARY</unit-calculator>
         </local-scheme>
      </backing-map-scheme>
      <partitioned-quorum-policy-scheme>
         <write-quorum>3</write-quorum>
      </partitioned-quorum-policy-scheme>
      <autostart>true</autostart>
   </distributed-scheme>
   <distributed-scheme>
      <scheme-name>my-cache-scheme-two</scheme-name>
      <service-name>MyCache2</service-name>
   </distributed-scheme>
<caching-schemes>
```

  - **Override XML**

```
<caching-schemes>
   <distributed-scheme>
      <partitioned-quorum-policy-scheme>
         <write-quorum>1</write-quorum>
      </partitioned-quorum-policy-scheme>
   </distributed-scheme>
</caching-schemes>
```

  - **The effective `<caching-schemes>` element after the application of the XML override**

```
<caching-schemes>
   <distributed-scheme>
      <scheme-name>my-cache-scheme</scheme-name>
      <service-name>MyCache</service-name>
      <backing-map-scheme>
         <local-scheme>
            <unit-calculator>BINARY</unit-calculator>
         </local-scheme>
      </backing-map-scheme>
```

```
      <partitioned-quorum-policy-scheme>
         <write-quorum>1</write-quorum>
      </partitioned-quorum-policy-scheme>
      <autostart>true</autostart>
   </distributed-scheme>
   <distributed-scheme>
      <scheme-name>my-cache-scheme-two</scheme-name>
      <service-name>MyCache2</service-name>
      <partitioned-quorum-policy-scheme>
         <write-quorum>1</write-quorum>
      </partitioned-quorum-policy-scheme>
   </distributed-scheme>
<caching-schemes>
```

Part IV
Using Topics

Learn how to configure and manage a Topic, how to publish data to a Topic and how to process data from a Topic Subscriber. Additionally, learn how to use a Topic Subscriber Group to achieve queue message processing functionality.
Part IV contains the following chapters:

20

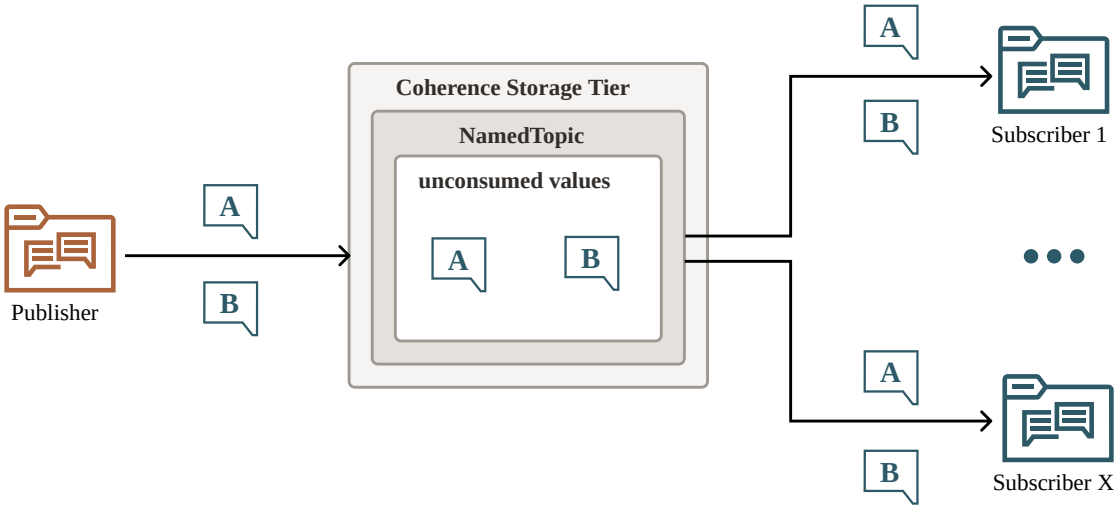
Introduction to Coherence Topics

Topics adds a publish and subscribe messaging functionality to Oracle Coherence. This chapter includes the following sections:

Topics Overview

The Topics API enables the building of data pipelines between loosely coupled producers and consumers.
One or more publishers publish a stream of values to a Topic. One to many subscribers consume the stream of values to a Topic. The Topic values are spread evenly across all Oracle Coherence data servers, enabling high throughput processing in a distributed and fault tolerant manner. Each direct subscriber to a Topic receives all values published to the Topic.

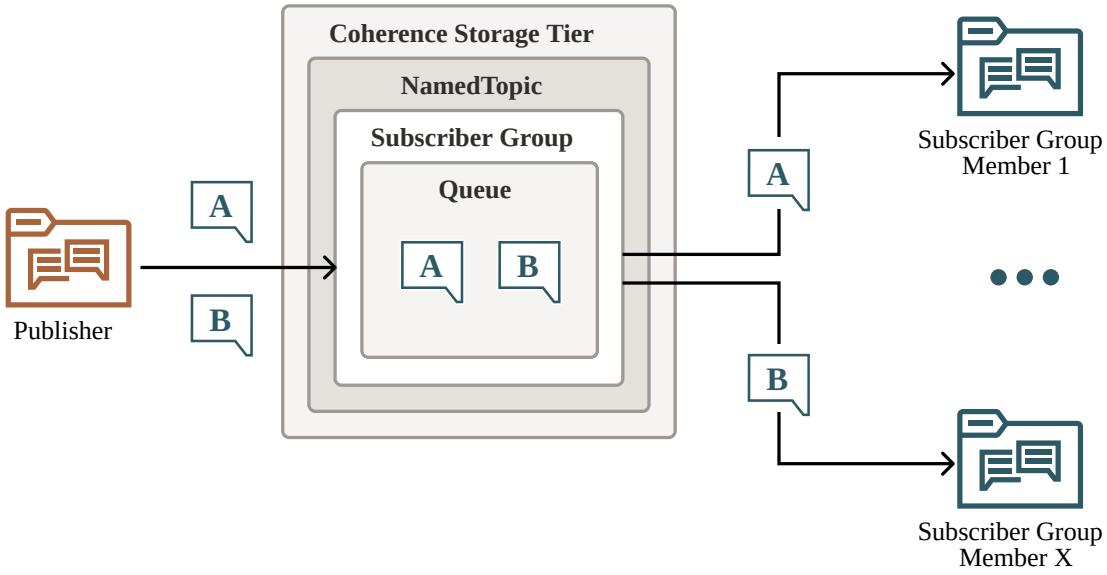
Figure 20-1 Topic Values



In publish and subscribe messaging, messages are delivered only to active subscribers of the Topic. Subscriber groups add queue processing capabilities to Topics. After a subscriber group is created on a Topic, all values sent to the Topic are accumulated for the subscriber group. One or more subscribers can subscribe to a subscriber group. For the stream of values accumulated by a subscriber group, each value is only delivered to one of its subscriber group members, thus enabling distributed, parallel processing of these values.

> **Note:**
>
> For examples of basic Topic operations, see Performing Basic Topic Publish and Subscribe Operations.

Figure 20-2 Topic Values in a Subscriber Group



About Channels

To scale publishers and subscribers while still guaranteeing message ordering, Coherence Topics introduces the concept of Channels. A Channel is similar in idea to how Coherence partitions data in distributed caches. To avoid confusion, the name partition is not reused. Channels are an important part of the operation of both publishers and subscribers.
A publisher is configured to control what ordering guarantees exist for the messages it publishes when they are received by subscribers. This is achieved by publishing messages to specific Channels. All messages published to a Channel will be received by subscribers in the order a publisher published them to that Channel. Messages published to a Channel by different publishers may be interleaved; there is no ordering guarantee across different publishers. Messages published to different Channels may be interleaved as they are received by subscribers, and may be received by different subscribers in a subscriber group.

The number of Channels that a Topic has allows publishers to scale better as they avoid contention that may occur with many publishers publishing to a single Channel. Message consumption can be scaled because multiple subscribers (in a group) will subscribe to different Channels, so scaling up receiving of messages, while maintaining order.

As new subscribers are created in the same subscriber group, the available Channels are allocated as fairly as possible among the subscribers. If there are more subscribers than Channels, then some subscribers will not be assigned a Channel and will not receive any messages. If a subscriber disconnects for some reason, such as failure or a heart-beat timeout, then its assigned Channels are reallocated to the remaining subscribers. It is possible to register listeners to be notified of Channel allocation changes.

The Channel count for a Topic is configurable, ideally a small prime (the default is 17). There are pros and cons with very small or very large Channel counts, depending on the application use case and what sort of scaling or ordering guarantees it requires.

About Position

Every element published to a Topic has a Position. A Position is an opaque data structure used by the underlying `NamedTopic` implementation to track the Position of an element in a channel and maintain ordering of messages.
Positions are then used by subscribers to track the elements that they have received, and when committing a Position to determine which preceding elements are also committed, and to then recover to the correct Position in the Topic when subscribers reconnect or recover from failure.

While a Position data structure is opaque, they are serializable, meaning that they can be stored into a separate data store by application code that wants to manually track message element processing. The combination of Channel and Position should be unique for each message element published and received.

21
Configuring Topics

You make configuration changes in the `coherence-cache-config.xml` file.
This section includes the following topics:

- Defining Topic Mappings
- Defining a Distributed Topic Scheme

Defining Topic Mappings

A Topic mapping maps a Topic name to a paged Topic scheme definition.The mappings provide a level of separation between applications and the underlying Topic definitions. The separation allows Topic implementations to be changed as required without having to change application code. Topic mappings have optional initialization parameters that are applied to the underlying paged Topic scheme definition.
Topic mappings are defined using a `<topic-mapping>` element within the `<topic-scheme-mapping>` node. Any number of Topic mappings can be created. The Topic mapping must include the Topic name and the scheme name to which the Topic name is mapped. See topic-mapping config element.

This section includes the following topics:

- Using Exact Topic Mappings
- Using Named Pattern Topic Mappings
- Using Subscriber Groups

Using Exact Topic Mappings

Exact Topic mapping maps a specific Topic name to a paged Topic scheme definition. An application must provide the exact name as specified in the mapping to use a Topic. The slash (`/`) and colon (`:`) are reserved characters and cannot be used in Topic names. Example 21-1 creates a single Topic mapping that maps the Topic name `exampleTopic` to a paged-topic-scheme definition with the scheme name `topic-scheme`.

Example 21-1 Sample Exact Topic 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">
   <topic-scheme-mapping>
      <topic-mapping>
         <topic-name>exampleTopic</topic-name>
         <scheme-name>topic-scheme</scheme-name>
      </topic-mapping>
   </topic-scheme-mapping>
   <caching-schemes>
      <paged-topic-scheme>
         <scheme-name>topic-scheme</scheme-name>
         <service-name>DistributedTopicService</service-name>
```

```
      </paged-topic-scheme>
   </caching-schemes>
</cache-config>
```

Using Named Pattern Topic Mappings

Name pattern Topic mappings allow applications to use patterns when specifying a Topic name. Patterns use the asterisk (*) wildcard. Name patterns alleviate an application from having to know the exact name of a Topic. The slash (`/`) and colon (`:`) are reserved characters and cannot be used in Topic names. Example 21-2 creates two Topic mappings. The first mapping uses the wildcard (*) to map any Topic name to a paged Topic scheme definition with the scheme name `basic-topic-scheme`. The second mapping maps the name pattern `account-*` to the paged Topic scheme definition with the scheme name `account-topic-scheme`.

Example 21-2 Sample Topic 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">
    <topic-scheme-mapping>
      <topic-mapping>
          <topic-name>*</topic-name>
          <scheme-name>basic-topic-scheme</scheme-name>
      </topic-mapping>
      <topic-mapping>
          <topic-name>account-*</topic-name>
          <scheme-name>account-topic-scheme</scheme-name>
          <service-name>accountDistributedTopicService</service-name>
      </topic-mapping>
    </topic-scheme-mapping>

    <caching-schemes>
      <paged-topic-scheme>
          <scheme-name>basic-topic-scheme</scheme-name>
      </paged-topic-scheme>
      <paged-topic-scheme>
          <scheme-name>account-topic-scheme</scheme-name>
      </paged-topic-scheme>
    </caching-schemes>
</cache-config>
```

For the first mapping, an application can use any name when creating a Topic and the name is mapped to the paged Topic scheme definition with the scheme name `basic-topic-scheme`. The second mapping requires an application to use a pattern when specifying a Topic name. In this case, an application must use the prefix `account-` before the name. For example, an application that specifies `account-overdue` as the Topic name uses the paged Topic scheme definition with the scheme name `account-topic-scheme`.

As shown in Example 21-2, it is possible to have a Topic name (for example `account-overdue`) that can be matched to multiple Topic mappings. In such cases, if an exact Topic mapping is defined, then it is always selected over any wildcard matches. Among multiple wildcard matches, the last matching wildcard mapping (based on the order in which they are defined in the file) is selected. Therefore, it is common to define less specific wildcard patterns earlier in the file that can be overridden by more specific wildcard patterns later in the file.

Note that there are different namespaces for Topic names and cache names.

Using Subscriber Groups

A Topic can have 0, 1, or more durable subscriber groups defined in the topic-mapping for the Topic. The subscriber group(s) are created along with the Topic and are ensured to exist before any data is published to the Topic. A subscriber group can have 0, 1, or more subscriber group members processing values from it. Creating multiple subscribers to a subscriber group enables parallel distributed processing of all the values collected by the durable subscriber group. When there are no subscribers, the subscriber group preserves the values for future subscriber(s) to consume each value.

> **Note:**
>
> A subscriber group does not have to be defined on a Topic's `topic-mapping` for a subscriber to be able to join its group. For more information about configured and dynamic subscriber groups, see Subscriber Groups for a NamedTopic.

Example 21-3 Sample Durable Subscriber Group

```
<?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">
   <topic-scheme-mapping>
      <topic-mapping>
         <topic-name>exampleTopic</topic-name>
         <scheme-name>topic-scheme</scheme-name>
         <subscriber-groups>
             <subscriber-group>
                 <name>durableSubscription</name>
             </subscriber-group>
         <subscriber-groups>
      </topic-mapping>
   </topic-scheme-mapping>
   <caching-schemes>
      <paged-topic-scheme>
         <scheme-name>topic-scheme</scheme-name>
         <service-name>DistributedTopicService</service-name>
      </paged-topic-scheme>
   </caching-schemes>
</cache-config>
```

Defining a Distributed Topic Scheme

Topic schemes are used to define the Topic services that are available to an application.Topic schemes provide a declarative mechanism that allows Topics to be defined independent of the applications that use them. This removes the responsibility of defining Topics from the application and allows Topics to change without having to change an application's code. Topic schemes also promote Topic definition reuse by allowing many applications to use the same Topic definition.
Topic schemes are defined within the `<caching-schemes>` element. A `<paged-topic-scheme>` scheme element and its properties are used to define a topic of that type.

Sample Distributed Topic Definition

The `<paged-topic-scheme>` element is used to define distributed Topics. A distributed Topic utilizes a distributed (partitioned) Topic service instance. Any number of distributed Topics can be defined in a cache configuration file. See paged-topic-scheme.

Example 21-4 defines a basic distributed Topic that uses `distributed-topic` as the scheme name and is mapped to the Topic name `example-topic`. The `<autostart>` element is set to `true` to start the service on a cache server node.

Example 21-4 Sample Distributed Topic 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">
   <topic-scheme-mapping>
     <topic-mapping>
       <topic-name>example-topic</topic-name>
       <scheme-name>distributed-topic</scheme-name>
       <value-type>java.lang.String</value-type>
     </topic-mapping>
   </topic-scheme-mapping>
   <caching-schemes>
     <paged-topic-scheme>
        <scheme-name>distributed-topic</scheme-name>
        <service-name>distributed-topic-service</service-name>
        <autostart>true</autostart>
     </paged-topic-scheme>
   </caching-schemes>
</cache-config>
```

> **Note:**
>
> The `<value-type>` subelement of `<topic-mapping>` is optional and needs to be specified only to enable enhanced generic type checking in processing the values of a Topic. For information about specifying a `ValueTypeAssertion` when getting a Topic, see Getting a Topic Instance.

This section includes the following topics:

- Size Limited Topic
- Topic with Expiring Values
- Storage Options for Topic Values
- Topic Values Serializer
- Persistent Topic

Size Limited Topic

Adding a `<high-units>` subelement to `<paged-topic-scheme>` element limits the storage size of the values retained for the Topic. The Topic is considered full if this storage limit is reached. Not exceeding this high water mark is managed by default using flow control. When subscriber(s) are lagging in processing outstanding values retained on the Topic, the publishers are throttled until there is space available. See Managing the Publisher Flow Control to Place Upper Bound on Topics Storage.

Topic with Expiring Values

Adding a `<expiry-delay>` subelement to `<paged-topic-scheme>` element limits the length of time that the published value lives on Topic, waiting to be received by a `Subscriber`.

Storage Options for Topic Values

The `<storage>` subelement allows specification of `on-heap`, `ramjournal` and `flashjournal` to store the values and metadata for a Topic. See Using the Elastic Data Feature to Store Data for details on `ramjournal` and `flashjournal` options.

Topic Values Serializer

The `<serializer>` subelement of `<paged-topic-scheme>` element enables specifying predefined serializers `pof` or `java`(default). See serializer.

Persistent Topic

If active persistence is configured for a Topic and the entire cluster is restarted, the outstanding unconsumed values on the Topic for subscriber(s) and subscriber group(s) are recovered during persistence recovery operations. The following subelements of `<paged-topic-scheme>` element configure whether a Topic is persistent or not. The optional `transient` subelement must be `false`, which is its default. The optional `persistence.environment` subelement references a pre-defined or custom persistence environment. For an example of a configuration, see Configuring a Persistent Topic .

22

Topic Configurations by Example

You can learn how to configure Coherence Topics by reviewing a series of sample Topic scheme definitions that can be used or modified as required. For detailed instructions on how to configure Topics, see Configuring Topics.
This chapter includes the following topics:

- Define a Durable Subscriber Group
- Configuring Value Expiry and Upper Bound on Storage Used for Values Retained for a Topic
  The following example illustrates how to parameterize a `paged-topic-scheme` element so that it can be configured via `topic-mapping` user-defined parameter macros.
- Configuring a Persistent Topic

Define a Durable Subscriber Group

The example below shows how to statically configure a durable subscriber-group `durable-subscription` for the `topicWithDurableSubscription` Topic. You can define none, one, or many subscriber groups for a Topic. All values published to `topicWithDurableSubscription` will be accumulated for this subscriber group.

```
<topic-mapping>
  <topic-name>topicWithDurableSubscription</topic-name>
  <scheme-name>topic-scheme</scheme-name>
  <value-type>java.lang.String</value-type>
  <subscriber-groups>
    <subscriber-group>
      <name>durable-subscription</name>
    </subscriber-group>
  </subscriber-groups>
</topic-mapping>
```

> **Note:**
>
> Since each value published to a Topic is retained until all subscriber(s)/subscriber group(s) have consumed the value, it is important to actively manage subscriber groups. If a statically configured subscriber group is no longer necessary, remove it from the `topic-mapping` configuration file. The life cycle of statically configured and dynamically created subscriber groups should be manage as described in **list subscriber groups** and **destroy subscriber group**.

Configuring Value Expiry and Upper Bound on Storage Used for Values Retained for a Topic

The following example illustrates how to parameterize a `paged-topic-scheme` element so that it can be configured via `topic-mapping` user-defined parameter macros.

See Using User-Defined Parameter Macros for details on how to use this feature with `cache-mapping`. The same usage model applies to `topic-mapping`.

In the example below, the `paged-topic-scheme.`*`configurable-topic-scheme`*`.expiry-delay` has a user-defined parameter macro default value of `0`; thus, the default expiry for the Topic values will be to never expire. The default `high-units` user-defined parameter macro value is `0B`; thus, there is no storage limit for `paged-type-scheme.`*`configurable-topic-scheme`* unless the referring `topic-mapping` explicitly defines `expiry-delay` and `high-units` parameters. For this example, the `topic-mapping.initial-params` override both `expiry-delay` and `high-units`.

Example 22-1 Configure paged-topic-scheme using User-defined Parameter Macros

```
<cache-config>
   <topic-mapping-schemes>
     <topic-mapping>
        <name>topic</name>
        <scheme>configurable-topic-scheme</scheme>
        <initial-params>
          <init-param>
            <param-name>high-units</param-name>
            <param-value>2G<param-value>
          </init-param>
          <init-param>
            <param-name>expiry-delay</param-name>
            <param-value>12h<param-value>
          </init-param>
        </initial-params>
     <topic-mapping>
   <topic-mapping-scheme>
   <caching-schemes>
     <paged-topic-scheme>
       <scheme-name>configurable-topic-scheme</scheme-name>
       <service-name>DistributedTopicService</service-name>
       <storage>on-heap</storage>
       <expiry-delay>{expiry-delay 0}</expiry-delay>
       <high-units>{high-units 0B}</high-units>
    </paged-topic-scheme>
   <caching-schemes>
</cache-config>
```

Configuring a Persistent Topic

If persistence is enabled on a Topic and the entire cluster is restarted, the outstanding unconsumed values on the Topic for subscribers, and subscriber group(s) are recovered during persistence recovery operations.
The following is a sample configuration that enables configuring persistent Topics.

Example 22-2 Configuring transient and persistent topic-mappings with same topic scheme

```
<cache-config>
   <topic-mapping-schemes>
     <topic-mapping>
        <name>persistent-topic-*</name>
        <scheme>persistent-topic-scheme</scheme>
     <topic-mapping>
     <topic-mapping>
        <name>transient-topic-*</name>
        <scheme>persistent-topic-scheme</scheme>
```

```
      <initial-params>
          <initial-param>
            <param-name>transient</param-name>
            <param-value>true</param-value>
          </initial-param>
        </initial-params>
    <topic-mapping>
  <topic-mapping-scheme>
  <caching-schemes>
    <paged-topic-scheme>
      <scheme-name>persistent-topic-scheme</scheme-name>
      <service-name>DistributedTopicService</service-name>
      <storage>on-heap</storage>
      <transient>{transient false}</transient>
      <persistence>
        <environment>default-active</environment>
      </persistence>
      <partitioned-quorum-policy-scheme>
      <recover-quorum>{recover-quorum 0}</recover-quorum>
      <partitioned-quorum-policy-scheme>
      <autostart>true</autostart>
      <expiry-delay>{expiry-delay 0}</expiry-delay>
      <high-units>{high-units 0B}</high-units>
    </paged-topic-scheme>
  <caching-schemes>
</cache-config>
```

In Example 2.3.3, all Topic names starting with prefix `persistent-topic-` are persistent Topics. All Topic names starting with prefix `transient-topic-` are transient Topics. If `paged-topic-scheme` subelement `transient` is `false` and a `persistence.environment` is defined, the Topic is persistent. When the `transient` subelement is set to `true`, the unconsumed values of the Topic is not recovered during persistence recovery operations when a cluster is completely restarted. This example uses one of the pre-defined persistence environments, `default-active`, for the value of `paged-topic-scheme.persistence.environment`. See Overview of the Pre-Defined Persistence Environment for details. Configuration of the `recover-quorum` subelement is discussed in Using Quorum for Persistence Recovery. If no value is provided for `recover-quorum` in a `topic-mapping.initial-param`, the default of 0 results in the predefined Using the Dynamic Recovery Quorum Policy being used.

23
Topic Management

Coherence provides various features and functionalities to help you manage and monitor Topics within a Coherence cluster. MBeans for Topics, Topic Subscribers, and Topic Subscriber Groups are at the core of these capabilities, with three reports available to view the data over time.
Management over REST provides endpoints for accessing and managing all aspects of Topics. The latest versions of the Coherence CLI and Coherence VisualVM Plug-in also support monitoring and management by using the command line and VisualVM, respectively.

Coherence metrics emit Topics, Topic Subscribers, and Topic Subscriber Groups related metrics. Four Grafana dashboards are available to view the Topics metrics over time.

This chapter includes the following topics:

- Topic MBeans
  When you cofigure Topics, the associated Topics MBeans become available to you.
- Topics Reports
  Three reports are available which provide detailed information on Topics, Topic Subscribers, and Topic Subscriber Groups.
- Management Over REST
  You are can view and manage Topics, Topic Subscribers, and Topic Subscriber Groups using the Management over REST API.
- Managing Topics Using the VisualVM Plug-In and CLI
  The Coherence VisualVM Plug-in and Coherence CLI are used to provide management and monitoring of Topics within a Coherence cluster.
- Grafana Dashboards for Topics
  When you enable Coherence metrics, these metrics can be scraped and viewed in four Grafana dashboards.
- Using Topics with Persistence
- Using Topics with Federation

Topic MBeans

When you cofigure Topics, the associated Topics MBeans become available to you.

The Topic MBeans are:

- `PagedTopic` MBean - Provides statistics for Topic services running in a cluster.
- `PagedTopicSubscriber` MBean - Provides statistics for Topic Subscribers running in a cluster.
- `PagedTopicSubscriberGroup` MBean - Provides statistics for Topic Subscriber Groups running in a cluster.

For more information about these MBeans, see the following topics in *Managing Oracle Coherence*:

- PagedTopic MBean

- PagedTopicSubscriber MBean
- PagedTopicSubscriberGroup MBean

Topics Reports

Three reports are available which provide detailed information on Topics, Topic Subscribers, and Topic Subscriber Groups.

The Topics reports are:

- Topic Report (`report-topics.xml`) - Provides detailed metrics for Topics defined within a cluster.
- Topic Subscribers Report (`report-subscribers.xml`- Provides detailed metrics for Topic Subscribers defined within a cluster.
- Topic Subscriber Groups Report (`report-subscriber-groups.xml`) - Provides detailed metrics for Topic Subscriber Groups defined within a cluster

For more information about these reports, see the following topics in *Managing Oracle Coherence*:

- Understanding the Topic Report
- Understanding the Topic Subscribers Report
- Understanding the Topic Subscriber Groups Report

Management Over REST

You are can view and manage Topics, Topic Subscribers, and Topic Subscriber Groups using the Management over REST API.

See *REST API for Managing Oracle Coherence*.

Managing Topics Using the VisualVM Plug-In and CLI

The Coherence VisualVM Plug-in and Coherence CLI are used to provide management and monitoring of Topics within a Coherence cluster.

See the following resources for more information on each tool:

- Coherence VisualVM Plug-in
- Coherence CLI
- CLI Command Reference

Grafana Dashboards for Topics

When you enable Coherence metrics, these metrics can be scraped and viewed in four Grafana dashboards.

The Grafana dashboards are:

- Topics Summary - Shows a summary of all Topics.
- Topic Details - Shows details about individual Topics.
- Topic Subscriber Details - Shows details about subscribers.

- Topic Subscriber Group Details - Shows details about subscriber groups.

These dashboards are available in the Coherence Operator GitHub repository.

For more information about configuring metrics, see Using Oracle Coherence Metrics in *Managing Oracle Coherence*.

Using Topics with Persistence

Coherence Topics will work with persistence enabled, both active and on-demand. When using on-demand persistence, there are some caveats. If the recover from snapshot functionality is used to recover the state of a topic from persistent storage, the data includes the state associated with subscribers and subscriber groups. If there are active subscribers connected to and subscribing from a topic when a recovery occurs, it can corrupt the subscribers internal state, causing them to fail or to return incorrect data. Recovery from snapshot should only be used when it can be guaranteed that there are no subscribers currently subscribing from topics.

Using Topics with Federation

Coherence Topics cannot be configured to use Federation and hence a topic's content and meta-data cannot be federated to another Coherence cluster. Federation in Coherence is asynchronous. Therefore, federating a topic would break any delivery and ordering guarantees that a topic provides.

Part V
Performing Data Grid Operations

Learn how to interact with Coherence caches, serialize data using POF, query cache data, process and aggregate cache data, handle events, and perform transactions.
Part V contains the following chapters:

- Introduction to Coherence Programming
- Performing Basic Cache Operations
- Performing Basic Topic Publish and Subscribe Operations
- Using Portable Object Format
- Querying Data in a Cache
- Using Continuous Query Caching
- Processing Data In a Cache
- Using Map Events
- Controlling Map Operations with Triggers
- Using Live Events
- Using Coherence Query Language
- Performing Transactions
- Working with Partitions
- Managing Thread Execution
- Constraints on Re-Entrant Calls
- Using Timeout with Cache Operations
- Using the Repository API
- Implementing Concurrency in a Distributed Environment
  The Coherence Concurrent module provides distributed implementations of the concurrency primitives from the `java.util.concurrent` package such as executors, atomics, locks, semaphores, and latches.
- Using Contexts and Dependency Injection
  Coherence provides support for Contexts and Dependency Injection (CDI) within the Coherence cluster members. This feature enables you to inject Coherence-managed resources, such as `NamedMap`, `NamedCache`, and `Session` instances into CDI managed beans; to inject CDI beans into Coherence-managed resources, such as event interceptors and cache stores; and to handle Coherence server-side events using the CDI observer methods.

24

Introduction to Coherence Programming

Read through a list of essential Coherence Java APIs and learn about Coherence support for generics, lambdas, streams, and default methods.

For details about using the Coherence C# and C++ API, see Extend Client APIs in *Developing Remote Clients for Oracle Coherence*.

This chapter includes the following sections:

Overview of the Coherence API

The Coherence API provides many programming features that allow you to interact with a cache and process cached data. In this release, the Coherence API provides a means to publish and subscribe values to a topic.For a detailed reference of the Coherence API, see *Java API Reference for Oracle Coherence*.

Understanding the Session Interface and the CacheFactory Class

The `Session` interface and the `CacheFactory` class are the main entry point for Coherence applications. Both APIs are used to manage `NamedCache` instances and are most often used for creating, releasing, and destroying cache instances. See Getting a Cache Instance.

Understanding the NamedCache Interface

A `NamedCache` is a `Map` implementation that holds resources that are shared among members of a cluster. The resources are expected to be managed in memory and are typically composed of data that are often stored persistently in a database or data that have been assembled or calculated at some significant cost.

The `NamedCache` interface provides many methods for performing cache operations and extends many different Map operations that add features for querying cache, processing cache entries, registering listeners for cache events, and controlling concurrent access to cache entries.

The `AsyncNamedCache` interface is an asynchronous version of the `NamedCache` Interface. The interface makes use of the `CompleteableFuture` API from the Java `java.util.concurrent` package to asynchronously perform cache operations.

See Performing Basic Cache Operations.

API for Queries

The `QueryMap` interface provides the ability to query a cache using various filters that operate on values that are extracted from cache entries. The interface also includes the ability to add and remove indexes. Indexes are used to correlate values stored in the cache to their corresponding keys and can dramatically increase performance. See Querying Data in a Cache.

API for Cache Processing

The `InvocableMap` interface provides the ability to invoke both entry-targeted processing and aggregating operations on a cache. The operations against the cache contents are executed by (and thus within the localized context of) a cache. This is particularly useful in a distributed environment, because it enables the processing to be moved to the location at which the entries-to-be-processed are being managed, thus providing efficiency by localization of processing.

An entry processor is an agent that operates against the entry objects within a cache. Coherence includes many predefined entry processors that can be used to perform many common operations. The processors are defined in the `com.tangosol.util.processor` package.

An entry aggregator represents processing that can be directed to occur against some subset of the entries resulting in an aggregated result. Common examples of aggregation include functions such as minimum, maximum, sum and average. However, the concept of aggregation applies to any process that must evaluate a group of entries to come up with a single answer. Aggregation is explicitly capable of being run in parallel in a distributed environment.

See Processing Data In a Cache.

API for Events

Coherence provides two programming models for processing events. The `ObservableMap` interface enables an application to receive events when the contents of a cache changes. To register interest in change events, an application adds a `Listener` implementation to the cache that receives events that include information about the event type (inserted, updated, deleted), the key of the modified entry, and the old and new values of the entry. See Using Map Events.

The live event model uses event interceptors to listen for specific events that occur within the cluster. The interceptors are created specific to the event types and can be chained together. Event types are defined for partitioned cache and server events, lifecycle events, and federated caching events. See Using Live Events.

API for Transactions

Coherence offers various transaction options that provide different transaction guarantees. The `ConcurrentMap` API is locking API that provides explicit locking. Entry processors also provide a lock-free programming model that minimizes contention and latency and improves system throughput, without compromising the fault-tolerance of data operations. Lastly the Transaction Framework API is a connection-based API that provides atomic transaction guarantees across partitions and caches even with a client failure. See Performing Transactions.

Serialization in Coherence

Coherence caches 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 different 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 efficient in both space

and time and is the recommended serialization option in Coherence. See Using Portable Object Format .

- `java.io.Serializable` – The simplest, but slowest option.
- `java.io.Externalizable` – This option requires developers to implement serialization manually, but can provide significant performance benefits. As compared to `java.io.Serializable`, serialized data sizes can be minimized by a factor of two or more. Smaller data sizes is 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 option 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`.

> **Note:**
>
> 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 allowed). Objects that implement their own serialization routines are not affected.

Support for Generics

The Coherence programming API makes use of Java generics to provide compile and runtime type checking together with compile type-inference.If you are new to Java Generics, see Generics Lesson in *Java Tutorials*. Many Coherence interfaces and public classes have associated types. Generics provides cleaner code, less casting, and better type safety. Type checking is not required but it is a recommended best practice.
In addition to Java Generics support, Coherence allows types to be explicitly configured when creating a `NamedCache` instance or when defining a cache in the cache configuration file. See Using NamedMap Type Checking.

> **Note:**
>
> The examples in this book may not be written to use generics. However, it is expected that the use of generics will be a standard approach when programming with the Coherence API.

25

Performing Basic Cache Operations

You can use the Coherence APIs to perform basic cache operations.
This chapter includes the following sections:

Overview of the NamedMap API

The `com.tangosol.net.NamedMap<K, V>` interface is the primary interface used by applications to get and interact with cache instances. The `NamedMap<K, V>` interface extends other interfaces, which each provide additional cache capabilities that are unique to Coherence and used to perform data grid operations.
The extended interfaces include:

- `java.util.Map<K, V>` – This interface provides basic `Map` methods such as `get()`, `put()`, and `remove()`.
- `com.tangosol.net.cache.CacheMap<K, V>` – This interface provides methods for getting a collection of keys (as a `Map`) that are in the cache and for putting objects in the cache. This interface also supports adding an expiry value when putting an entry in a cache.
- `com.tangosol.util.QueryMap<K, V>` – This interface provides methods for querying the cache. See Querying Data in a Cache.
- `com.tangosol.util.InvocableMap<K, V>` – This interface provides methods for server-side processing of cache data. See Processing Data In a Cache.
- `com.tangosol.util.ObservableMap<K, V>` – This interface provides methods for listening to cache events. See Using Map Events.
- `com.tangosol.util.ConcurrentMap<K, V>` – This interface provides methods for concurrent access such as `lock()` and `unlock()`. See Performing Transactions.

> **Note:**
>
> If you want to issue a `put` with expiry, which is not available with `NamedMap`, you can use the `com.tangosol.net.NamedCache<K, V>` interface, which extends `NamedMap`. You can retrieve a `NamedCache` instance by using the Session API as shown in the examples provided in the subsequent sections.

Getting a Cache Instance

To get a reference to a `NamedMap` instance, applications can use the `Session` API and the `CacheFactory` API.The `Session` API is the preferred approach for getting a cache because it offers a concise set of methods that allows for the use and injection of Coherence sessions that are non-static. In contrast, the `CacheFactory` API exposes many static methods that require knowledge of internal Coherence concepts and services. Lastly, the `Session` API allows for a more efficient lifecycle and integration with other frameworks, especially frameworks that use injection.
The following example creates a session using a default session provider then gets a reference to a `NamedMap` instance using the `Session.getCache` method. The name of the cache is included as a parameter.

```
import com.tangosol.net.*;
...
Session session = Session.create();
NamedMap<Object, Object> map = session.getMap("MyCache");
```

The `CoherenceSession` class implements the `Session` interface and allows applications to use the `new` operator. For example:

```
import com.tangosol.net.*;
...
Session session = new CoherenceSession();
NamedMap<Object, Object> map = session.getMap("MyCache");
```

The following example gets a reference to a `NamedCache` instance using the `CacheFactory.getCache` method and includes the name of the cache as a parameter:

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

> **Note:**
>
> Although you can use CacheFactory to get `NamedCache` instances, Oracle recommends you to use the Session API to get `NamedMap` and `NamedCache` instances.

Both the `Session` API and `CacheFactory` API start the underlying cache service if necessary. The cache instance is created using a cache scheme that is defined in the cache configuration

file (`coherence-cache-config.xml` by default). The cache scheme is mapped to the name `MyCache`. See Configuring Caches.

A `NamedMap` instance can store keys and values of any type. However, applications must ensure type safety when interacting with cache entries. Applications can create `NamedMap` instances for specific types and Coherence also offers API-level type checking. See Using NamedMap Type Checking.

Requirements for Cached Objects

Cache keys and values must be serializable (for example, `java.io.Serializable` or Coherence Portable Object Format serialization). 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. See Using Portable Object Format .

Performing Cache Put Operations

Basic cache `put` operations are performed using the `put` method as defined by the `Map` interface.The `put` method adds an entry to the cache and returns the previous value for the specified key. For example:

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

// obtain a session
Session session = ...

NamedMap<Object, Object> map = session.getMap("MyCache");
map.put(key, value);"
```

The `putAll` method is used to add multiple entries to a cache in a single bulk load operation and requires the entries to be in a `Map` type data structure. See Pre-Loading a Cache.

Performing Cache Get Operations

Basic cache get operations are performed using the `get` method as defined by the `Map` interface.The mapped value is returned for the specified key. For example:

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

// obtain a session
Session session = ...

NamedMap<Object, Object> map = session.getMap("MyCache");
map.put(key, value);
System.out.println(map.get(key));
```

Performing Cache Remove Operations

Basic cache remove operations are performed using the `remove` method as defined by the `Map` interface.The mapping for the specified key is removed from the cache and the previous value is returned. For example:

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

// obtain a session
Session session = ...

NamedMap<Object, Object> map = session.getMap("MyCache");
map.put(key, value);
System.out.println(map.get(key));
map.remove(key)
```

Using Default Map Operations

The `java.util.Map` interface includes default methods for performing operations such as `putIfAbsent`, `replaceAll`, and `merge`, to name a few. Coherence overrides many `Map` method implementations to ensure these operations perform well in a distributed environment.The methods have been re-implemented to use entry processors and to take advantage of lambda expressions. The methods are available when using the `NamedCache` interface. For example:

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

NamedCache<Object, Object> cache = CacheFactory.getCache("hello-example");
cache.putIfAbsent(key, value);
System.out.println(cache.get(key));
```

Lambda expressions can also be used to perform any required processing on the entries. For example,

```
map.replaceAll((key, value) ->
   {
   value.setLastName(value.getLastName().toUpperCase());
   return value;
   });
```

Pre-Loading a Cache

Pre-Loading a cache is a common scenario used to populate a cache before an application uses the data.
This section includes the following topics:

- Bulk Loading Data Into a Cache
- Performing Distributed Bulk Loading
- A Distributed Bulk Loading Example

Bulk Loading Data Into a Cache

The `put` method can be used to bulk load data into a cache. However; 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 (as defined in the `java.util.Map` interface) which adds more unnecessary overhead.

```
public static void bulkLoad(NamedMap map, 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)
        {...}
    }
```

Loading the cache can be made much more efficient by using the `ConcurrentMap.putAll` method instead. For example:

```
public static void bulkLoad(NamedMap map, 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)
    {...}
}
```

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.

```
protected Set getStorageMembers(NamedMap map)

        {
        return ((PartitionedService) map.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.

```
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`.

```
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);
            }
        }
```

Clearing Caches

The contents of a map can be cleared using either the `clear` or `truncate` methods that are defined by the `NamedMap` interface.The `clear` method can result in significant memory and CPU overhead and is generally not recommended to clear a distributed cache. As an alternative, the `truncate` method can be used. For example:

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

// obtain a session
Session session = ...

NamedMap<Object, Object> map = session.getMap("MyCache");
map.put(key, value);
System.out.println(map.get(key));

map.truncate();
```

The `truncate` method makes better use of memory and CPU resources and is often the best option when clearing large caches and caches that use listeners. The `truncate` method is also ideal for clearing a near cache as it clears both the front map and back map. The removal of

entries caused by the `truncate` method are not observable by listeners, triggers, and interceptors. However, a `CacheLifecycleEvent` event is raised to notify all subscribers of the execution of this operation.

Releasing Caches

Applications that no longer require a cache should use the `NamedMap.release` method to release local resources associated with the specified instance of the cache.Releasing a cache makes it no longer usable but does no affect the contents of a cache or other references to the cache across the cluster. For example:

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

// obtain a session
Session session = ...

NamedMap<Object, Object> map = session.getMap("MyCache");
map.put(key, value);
System.out.println(map.get(key));

map.release();
```

Destroying Caches

NamedMaps or NamedCaches that are created using a `Session` or a `CacheFactory` class can be destroyed using the `NamedMap.destroy` method. The `destroy` method destroys the specified cache across the entire cluster. References to the cache are invalidated; the cached data is cleared; and, all resources are released. The `destroy` method is often preferred over the `NamedMap.clear()` method, which can be both a memory and CPU intensive task in a distributed environment. For example:

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

// obtain a session
Session session = ...

NamedMap<Object, Object> map = session.getMap("MyCache");
map.put(key, value);
System.out.println(map.get(key));

map.destroy();
```

Closing Sessions

Applications should close a session when the session is no longer required.When a session is closed, all the resources that are associated with the session are also closed and references to those resources are no longer valid. An illegal state exception is thrown if an application attempts use a closed session or its resources.

Use the `close` method to close a session. For example:

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

try(Session session = Session.create()) {
   NamedMap<Object, Object> map = session.getMap("MyCache");
   map.put(key, value);
   System.out.println(map.get(key));
   session.close();
}
catch (Exception e){
   ...
}
```

Performing NamedMap Operations Asynchronously

The `com.tangosol.net.AsyncNamedMap<K, V>` interface allows cache operations to be completed in parallel.The interface makes use of the Java `CompletableFuture` class, which provides completion and/or exception callbacks, chaining multiple asynchronous calls to execute one after another, and waiting for all the calls executing in parallel to complete. Performing `NamedMap` operations asynchronously can improve throughput and result in more responsive user interfaces. The Coherence examples provide additional examples of performing asynchronous cache operations. See Coherence Asynchronous Features Example in *Installing Oracle Coherence*.

To perform asynchronous cache operations, you must use the `CompletableFuture` class and implement the `Future` interface. For example:

```
import com.tangosol.net.AsyncNamedCache;
import com.tangosol.net.CacheFactory;
import com.tangosol.net.NamedCache;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Future;

public class HelloWorld {

    public static void main(String[] args) throws ExecutionException,
       InterruptedException {

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

       NamedMap<Object, Object> map = CacheFactory.getCache("MyCache");
       AsyncNamedMap<Object, Object> as = map.async();

       Future future = as.put(key, value);
       future.get();

       CompletableFuture cf = as.get(key);

       System.out.println(cf.get());

       CompletableFuture cfremove = as.remove(key);

       System.out.print("Removing key/value: " + cfremove.get() + "\n" );
       System.out.println("The key/value is: " + future.get());
```

```
    }
}
```

Using NamedMap Type Checking

Coherence includes the ability to request strongly-typed `NamedMap` instances when using either the `Session` or the `CacheFactory` API by using explicit types.By default, `NamedMap<Object, Object>` instances return. This is the most flexible mechanism to create and use `NamedMap` instances; however, it is the responsibility of the application to ensure the expected key and value types when interacting with cache instances.
The following example stores objects of any type for use by an application:

```
NamedMap<Object,Object> map = session.getMap("MyCache");
```

or,

```
NamedCache<Object,Object> cache = CacheFactory.getCache("MyCache");
```

The `getCache` or `getMap` method can be used to request a `NamedCache` or a `NamedMap` instance of a specific type, including if necessary, without type checking. The `TypeAssertion` interface is used with the `getCache` or `NamedCache` method to assert the correctness of the type of keys and values used with a `NamedCache` or `NamedMap` instance. The method can be used to assert that a cache should use raw types. For example:

```
NamedMap<Object, Object> map = session.getMap("MyCache",
TypeAssertion.withRawTypes());
```

Likewise, when using `CacheFactory`,

```
NamedCache<Object, Object> cache = CacheFactory.getCache("MyCache",
TypeAssertion.withRawTypes());
```

For stronger type safety, you can create a cache and explicitly assert the key and value types to be used by the cache. For example, to create a cache and assert that keys and values must be of type `String`, an application can use:

```
NamedMap<String, String> map = session.getMap("MyCache",
TypeAssertion.withTypes(String.class, String.class));
```

A `NamedMap` instance is not required to adhere to the asserted type and may choose to disregard it; however, a warning message is emitted at compile-time. For example:

```
NamedMap map = session.getCache("MyCache",
TypeAssertion.withTypes(String.class, String.class));
```

Likewise, an application may choose to assert that raw types be used and a `NamedMap` instance may use specific types. However, in both cases, this could lead to errors if types are left unchecked. For example:

```
NamedMap<String, String> map = session.getCache("MyCache",
TypeAssertion.withRawTypes());
```

For the strongest type safety, specific types can also be declared as part of a cache definition in the cache configuration file. A runtime error occurs if an application attempts to use types that are different than those configured as part of the cache definition. The following examples configures a cache that only supports keys and values that are of type `String`.

```
<cache-mapping>
   <cache-name>MyCache</cache-name>
   <scheme-name>distributed</scheme-name>
   <key-type>String</key-type>
   <value-type>String</value-type>
</cache-mapping>
```

Lastly, an application can choose to disable explicit type checking. If type checking is disabled, then the application is responsible for ensuring type safety.

```
NamedCache<Object, Object> map = session.getMap("MyCache",
TypeAssertion.withoutTypeChecking());
```

26

Performing Basic Topic Publish and Subscribe Operations

You can use the Coherence APIs to perform basic topic publish and subscribe operations.This chapter includes the following sections:

Overview of Topics API

The `com.tangosol.net.NamedTopic<V>` interface is the initial interface used by applications to publish and subscribe to the topic. It has factory methods to create a `Publisher` and a `Subscriber`. It also has methods to manage subscriber groups.
The `com.tangosol.net.Pubisher<V>` interface has an asynchronous `publish` method that publishes a value to a topic. The method returns `CompletableFuture<Publisher.Status>` to enable tracking when the send completes or if an exception is thrown. The returned `Publisher.Status` contains details of the published message

The `com.tangosol.net.Subscriber<V>` interface has a receive method that returns a `CompleteFuture<Element<V>>`. A a Subscriber either subscribes to a `subscriber group` (using the `Subscriber.inGroup` (`subscriberGroupName`) option or subscribes directly to a topic as an anonymous subscriber.

Publisher Creation Options

The following table lists the Publisher creation options to be used with `NamedTopic.createPublisher(Publisher.Option... option)`.

Publisher Option	Description
`OnFailure.Stop`	`Default`. If an individual `Publisher.send(V)` invocation fails, stop any further publishing and close the `Publisher`.
`OnFailure.Continue`	If an individual `Publisher.send(V)` invocation fails, skip that value and continue to publish other values.
`FailOnFull.enabled()`	When the storage size of the unprocessed values on the topic exceeds a configured `high-units` value, the `CompletableFuture` returned from the `Publisher.send(V)` invocation should complete exceptionally. Overrides the default to block completing until the operation completes after space becomes available on the topic.
`OrderBy.thread()`	`Default`. Ensures that all `values sent` from the same thread are stored sequentially.
`OrderBy.none()`	Enforces no specific ordering between `sent values` allowing for the greatest level of parallelism.
`OrderBy.id(int)`	Ensures ordering of `sent values` across all threads which share the same ID.
`OrderBy.value(ToIntFunction)`	Computes the unit-of-order based on the application of this method on `sent value`.
`OrderBy.roundRobin()`	Publishes messages to each channel in the topic sequentially, looping back around to channel zero, after “channel count” messages have been published.

Subscriber Creation Options

The following table lists the Subscriber creation options to be used with `NamedTopic.createSubscriber(Subscriber.Option... option)`.

Subscriber Option	Description
`Name.of(String)`	Specifies a subscriber group name to join as a member. For each value delivered to a subscriber group, only one member of the group receives the value.

Subscriber Option	Description
`Filtered.by(Filter)`	Only `Subscriber.receive()` values matching this `Filter`. Only one `Filter` per subscription group.
`Convert.using(Function)`	Converts topic value using provided `Function` prior to `Subscriber.receive()`. Only one `Converter` per subscription group.
`CompleteOnEmpty.enabled()`	When no values left to `Subscriber.receive()`, returned `CompletableFuture.get()` returns `null` element. By default, returned `CompletableFuture.get()` blocks until next topic value is available to return.
`ChannelOwnershipListeners.of(ChannelOwnershipListener)`	Adds a `ChannelOwnershipListener` to receive events when the channels assigned to the subscriber change.

The `com.tangosol.net.Subscriber<V>` interface has a receive method that returns a `CompleteFuture<Element<V>>`. A Subscriber either subscribes to a `subscriber group` (using `Subscriber.Name.of` (`subscriberGroupName`) or directly subscribes to a topic.

Getting a Topic Instance

To get a reference to a `NamedTopic` instance, applications can use the `Session` API or the `CacheFactory` API.The `Session` API is the preferred approach for getting a topic because it offers a concise set of methods that allows for the use and injection of Coherence sessions that are non-static. In contrast, the `CacheFactory` API exposes many static methods that require knowledge of internal Coherence concepts and services. Lastly, the `Session` API allows for a more efficient lifecycle and integration with other frameworks, especially frameworks that use injection.
The following examples demonstrates creating a session using a default session provider and then get a reference to a `NamedTopic` instance using the `Session.getTopic` method. The name of the topic is included as a parameter.

```
import com.tangosol.net.*;
...
Session session = Session.create();
NamedTopic<String> Topic =
session.getTopic("topic",ValueTypeAssertion.withType(String.class));
```

Or

```
ConfigurableCacheFactory ccf   = CacheFactory.getConfigurableCacheFactory();
NamedTopic<String> topic       =
                 ccf.ensureTopic("topic",
ValueTypeAssertion.withType(String.class));
```

To bypass generics type checking, omit optional `<value-type>` definition in topic-mapping configuration and omit `ValueTypeAssertion` option as a parameter for `Session.getTopic` or `ConfigurableCacheFactory.ensureTopic` methods.

Using NamedTopic Type Checking

Coherence includes the ability to request strongly-typed `NamedTopic` instances when using either the `Session` or `CacheFactory` API by using explicit types.By default, `NamedTopic<Object>` instances return. This is the most flexible mechanism to create and use `NamedTopic` instances; however, it is the responsibility of the application to ensure the expected value types when interacting with topic instances.
The following example stores objects of any type for use by an application:

```
import com.tangosol.net.*;
...
Session session = Session.create();
NamedTopic<Object> topic = session.getTopic(“MyTopic”);
```

or,

```
ConfigurableCacheFactory ccf   = CacheFactory.getConfigurableCacheFactory();
NamedTopic<Object> topic       = ccf.ensureTopic(“MyTopic”);
```

The `getTopic` method can be used to request a `NamedTopic` instance of a specific type, including if necessary, without type checking. The `ValueTypeAssertion` interface is used with the `getTopic` method to assert the correctness of the type of values used with a `NamedTopic` instance. The method can be used to assert that a topic should use raw types. For example:

```
NamedTopic<Object> topic = session.getTopic("MyTopic",
 ValueTypeAssertion.withRawTypes());
```

likewise when using `CacheFactory`,

```
NamedTopic<String> topic       =
                 ccf.ensureTopic(“MyTopic”,
ValueTypeAssertion.withRawTypes());
```

For stronger type safety, you can create a topic and explicitly assert the value types to be used by the topic. For example, to create a topic and assert that values must be of type `String`, an application can use:

```
NamedTopic<String> topic = session.getTopic("MyTopic",
 ValueTypeAssertion.withTypes(String.class);
```

A `NamedTopic` instance is not required to adhere to the asserted type and may choose to disregard it; however, a warning message is emitted at compile-time. For example:

```
NamedTopic topic = session.getTopic("MyTopic",
 ValueTypeAssertion.withTypes(String.class);
```

Likewise, an application may choose to assert that raw types be used and a name cache instance may use specific types. However, in both cases, this could lead to errors if types are left unchecked. For example:

```
NamedTopic<String> topic = session.getTopic("MyTopic",
 ValueTypeAssertion.withRawTypes());
```

For the strongest type safety, specific types can also be declared as part of a topic definition in the cache configuration file. A runtime error occurs if an application attempts to use types that are different than those configured as part of the topic definition. The following examples configures a topic that only support values that are of type `String`.

```
<topic-mapping>
   <topic-name>MyTopic</topic-name>
   <scheme-name>distributed-topic</scheme-name>
   <value-type>String</value-type>
</topic-mapping>
```

Lastly, an application can choose to disable explicit type checking. If type checking is disabled, then the application is responsible for ensuring type safety.

```
NamedTopic<Object> topic = session.getTopic("MyTopic",
   ValueTypeAssertion.withoutTypeChecking());
```

Publishing to a Topic

An active `Publisher`, one that has not been closed, asynchronously sends a value to a topic using the send method and returns a `CompletableFuture`. For example, an asynchronous call is turned into a synchronous call by calling join on the `CompletableFuture`.
See `CompletableFuture` for the numerous options available for working with the completion of many outstanding asynchronous operations.

```
NamedTopic<String> topic    = ….;
Publisher<String> publisher = topic.createPublisher();
CompletableFuture future    = publisher.send(“someValue”);

// Completes when asynchronous send completes.
// Throws an exception if send operation completed exceptionally.
future.join();
```

- Requirements for Topic Values
- Awaiting Completion of Outstanding Publisher Sends
- Releasing Publisher Resources

Requirements for Topic Values

Topic values must be serializable (for example, `java.io.Serializable` or Coherence Portable Object Format serialization). See Using Portable Object Format .

Awaiting Completion of Outstanding Publisher Sends

If a publisher sends a number of values to a topic without waiting for each send to complete, a `Publisher.flush().join()` invocation synchronously waits for all pending `Publisher.send()` to complete. Note that even when one or more of the outstanding operations complete exceptionally, this invocation always completes successfully. Use the `CompletableFuture` methods for detecting completing exceptionally when it is necessary to detect when a value is not published successfully to the topic.

Releasing Publisher Resources

One releases publisher resources by explicitly invoking `Publisher.close()` or by creating the publisher with a `try-with-resources` statement as shown in try-with-resource. `Publisher.close()` is a blocking operation and does not return until the close is complete. Once the Publisher is closed, it is no longer active (see `Publisher.isActive()`), all future `send` operations on that `Publisher` fail with an `IllegalStateException` stating the Publisher is no longer active. However, all outstanding `CompletableFutures` for outstanding `send` operations occurring prior to the `close` call are allowed to complete. Just as `Publisher.flush.join()`, the close may complete successfully or exceptionally, and no exceptions are thrown from the close for any outstanding `CompletableFutures` that complete exceptionally.

Subscribing Directly to a Topic

An active subscriber receives all values delivered to a topic. A subscriber is active from its creation until it is closed.

```
NamedTopic<String> topic      = ….;
Subscriber<String> subscriber = topic.createSubscriber();

CompletableFuture<Element<String>> future = receive();
String received  = future.get().getValue();
```

Releasing Subscriber Resources

A subscriber can be closed either by explicitly invoking the `Subscriber.close()` operation or by creating the subscriber with a `try-with-resources` statement.For details, see try-with-resource.
Subscriber close is a blocking operation and does not return until the operation is completed. After `close` is called on a `Subscriber`, all future `receive` calls on the inactive `Subscriber` fail with an `IllegalStateException`. Most of the outstanding `CompletableFuture`(s) from `receive` calls will complete exceptionally, while some of them may complete.

Subscribing to a Subscriber Group

A topic value either is retained until it is received by a subscriber group member or expires. By default, there is no expiry for topic values.

See configuration `<expiry-delay>` subelement of <paged-topic-scheme>.

```
NamedTopic<String> topic      = ….;
Subscriber<String> subscriber =
         topic.createSubscriber(Subscriber.Name.of("subscriberGroup"));
CompletableFuture<Element<String>> future = receive();
String value  = future.get().getValue();
```

Subscriber Groups for a NamedTopic

Subscriber group can be created in two ways.

There are:

- Statically configure durable subscriber group(s) on a `<topic-mapping>` element in the `coherence-cache-config` file. See example durable subscription group configuration.

- Dynamically create a subscriber group when a Subscriber is created with the `Subscriber.Name.of(aSubscriberGroupName)` option to join a subscriber group, and the `subscriber group` does not exist yet on the topic.

The method `NamedTopic.getSubscriberGroups` returns both configured and dynamically created subscriber groups.

```
NamedTopic<String> topic            = ….;
Set<String>        subscriberGroups = topic.getSubscriberGroups();
```

Destroying a Topic's Subscriber Group

A subscriber group life span is independent of its subscriber group members. The code fragment below stops accumulating values for a subscriber group and releases all unconsumed values.
All existing subscriber group members will be impacted by this operation and all outstanding asynchronous receive operations for the subscriber group members are cancelled.

```
NamedTopic<String> topic            = ….;
topic.destroySubscriberGroup("subscriberGroup");
```

Committing Messages

When using subscriber groups, messages are durable and will be delivered with an "at least once" guarantee. This means that a message will be redelivered if the subscriber disconnects for any reason or the subscriber's channels are re-allocated to another subscriber. To indicate a message has been processed and should not be redelivered, the subscriber must commit the message. A subscriber does not have to commit every message received. Committing a message automatically commits any messages received by the subscriber prior to the committed message. Messages can be committed synchronously or asynchronously.
To synchronously commit a message, use the `commit()` method on the `Element` provided by the `CompletableFuture` that is returned by the subscriber's `receive()` method.

For example, the code below creates a subscriber for the topic named "test-topic" in the subscriber group named "test". A message is received, processed, and then committed. The returned `Subscriber.CommitResult` can be used to determine information about the commit operation, such as success or failure.

```
Subscriber<String> subscriber = session.createSubscriber("test-topic",
Subscriber.inGroup("test"));
CompletableFuture<Subscriber.Element<String>> future = subscriber.receive();
Subscriber.Element<String> element = future.get();
String message = element.getValue();
// process message...
Subscriber.CommitResult result = element.commit();
```

To commit a message asynchronously use the element's `commitAsync()` method.

For example, the code below calls the element's `commitAsync()` method. The `CompletableFuture` returned will be completed when the commit operation is completed.

```
CompletableFuture<Subscriber.CommitResult> future = element.commitAsync();
```

Messages can also be committed using just a Channel and a Position, using the Subscriber's commit methods.

For example, in the code shown below a Subscriber receives an element, and the message's Channel and Position obtained from the element. After processing, the message is committed using the Channel and Position.

```
Subscriber<String> subscriber = session.createSubscriber("test-topic",
Subscriber.inGroup("test"));
CompletableFuture<Subscriber.Element<String>> future = subscriber.receive();
Subscriber.Element<String> element = future.get();

int channel = element.getChannel();
Position position = element.getPosition();

String message = element.getValue();

// process message...

Subscriber.CommitResult result = subscriber.commit(channel, position);
```

Managing the Resources Used by a Topic

Both Publisher and Subscriber have resources associated with them and the resources must be closed when no longer needed.

Both `Publisher` and `Subscriber` can be used with `try-with-resource` pattern. This usage pattern ensures `close` is called for both of these topic resources. See Publisher auto closed resource details here. See Subscriber auto closed resource details here.

Assume subscriber group `durableSubscriber` is statically configured for topic `aTopic` as shown in Example 21-3.

Example 26-1 try-with-resource for Publisher

```
Session    session = Session.create();
NamedTopic topic   = session.createTopic("aTopic",
                                         ValueTypeAssertion.withType(String.class));
try (Publisher<String> publisher = topic.createPublisher(…)) {
   int MAX = …;
   for (int i = 0; I < MAX; i++) {
       publisher.send("value" + i);
   }

   // wait for all outstanding asynchronous operations to complete before publisher is
implicitly closed by exiting try-with-resource block.
   publisher.flush().join();
}
```

Example 26-2 Subscriber processing with try-with-resource

```
//Process outstanding values on a subscriber group
try (Subscriber<String> subscriber =
         topic.createSubscriber(Subscriber.Name.of("durableSubscriber")),
                                Subscriber.CompleteOnEmpy.enabled())) {
    while (true) {
        // asynchronous receive from direct subscriber to topic.
        CompletableFuture<Element<String>> result= subscriber.receive();
        Element<String>                    topicElement = result.get();
        if (topicElement == null) {
            // no more elements available
            break;
```

```
        } else {
          String value = topicElement.getValue();
            // process value received from subscriber.
            ...
        }
// auto close subscriber and its resources. Outstanding CompletableFutures for
// receive will mostly complete exceptionally, though some may complete.
// a subscriber group accumulates future delivered values even if there
// are no active subscriber group members.
}
```

`NamedTopic.destroy()` releases all the topics resources and destroys the instance. All outstanding `Publishers` and `Subscribers` are closed. Outstanding operations on `Publishers` and `Subscribers` are completed exceptionally. `Subscriber groups` are destroyed and all topic storage is released, including persistent storage.

Understanding Topics Flow Control

Given that topics `Publisher.send` and `Subscriber.receive` methods provide an asynchronous (non-blocking) way of submitting data exchange requests, there is a default flow control mechanism to efficiently manage these requests.

When the automated flow control detects too many outstanding send and/or receive operations, running out of storage on a topic for outstanding values to be consumed, the outstanding `CompletableFuture`(s) for the async `send` and/or `receive` may be throttled to allow the system to catch up with all the outstanding requests. Both `Publisher` and `Subscriber` class provide a way for the application to get a `FlowControl` instance that allows it to opt-out of automatic flow control and manually govern the rate of the request flow. See the `com.tangosol.net.FlowControl` Javadoc for more details.

Managing the Publisher Flow Control to Place Upper Bound on Topics Storage

In a data processing pipeline where the published values to the topic are significantly outpacing the subscribers' abilities to process those values, one can either allow for an unbounded storage for the unconsumed values on the topic or place a maximum storage size constraint for the retained values on the topic.

By default, there are no storage constraints on the storage size for the values being retained by a topic. If a storage limit is configured for the topic as illustrated here, there are several options available to restrict the topic's storage usage.

Once a storage limit is configured for a topic, the default behavior is for the publisher(s) to a topic to be throttled, blocked, when the next send would exceed the topic's storage limit. The intention is that by blocking the publisher(s), it gives the subscriber(s) time to catch up on processing, while limiting the amount of storage used by the topic. Once the subscriber(s) catch up on processing and those values on the topic have been consumed by all subscriber(s) and subscriber group(s), then the publisher(s) send(s) would be allowed to resume.

There are Publisher creation time options to change the default behavior. The `Publisher.FailOnFull.enabled()` option indicates that a publisher `send` should not block, but rather complete exceptionally. The default `Publisher` option `Publisher.OnFailure.Stop` results in the Publisher auto closing when this exception occurs. This default preserves the

order of published values from the publisher since the publisher is closed on first exceptional case that the value could not be delivered.

Example 26-3 FailOnFull publishing

```
Session    session = Session.create();
NamedTopic topic   = session.createTopic("aTopic",
                                         ValueTypeAssertion.withType(String.class));

try (Publisher<String> publisher =
topic.createPublisher(Publisher.FailOnFull.enabled())) {
   int MAX   = …;
   int cSent = 0;
   for (int i = 0; I < MAX; i++) {
       try {
           // synchronous send that throws exception if publisher is full.
           publisher.send("value" + i).join();
        cSent++;
      } catch (Exception e) {
        // when full throws CompletionException with a cause of IllegalStateException:
the topic is at capacity.
        break;
      }
   }
   if (cSent != MAX) {
      // all values were not published to topic.
   }
}
```

If there is no need to preserve this order, the Publisher can be created with option `Publisher.OnFailure.Continue` and only the sends that occur when the topic is full, will fail.

To override the default blocking on sending to a full topic, one can use the following pattern to exempt a thread from flow control pause. This technique disables flow control and eliminates the storage limit on the topic for this `NonBlocking` thread.

Example 26-4 Exempt thread from flow control pause when reaching topic storage limit

```
import com.tangosol.net.*;
import com.oracle.coherence.common.base.NonBlocking;

Session    session = Session.create():
NamedTopic topic   = Session.createTopic("aTopic",
                                         ValueTypeAssertion.withType(String.class));
try (NonBlocking       nb        = NonBlocking();
     Publisher<String> publisher = topic.createPublisher(…)) {
   int MAX = …;
   for (int i = 0; I < MAX; i++) {
       publisher.send("value" + i);
   }
}
```

27
Using Portable Object Format

You can use Portable Object Format (POF) to serialize Java objects for use in Coherence.

For information on how to work with POF when building .NET and C++ extend clients, see Building Integration Objects for .NET Clients and Building Integration Objects for C++ Clients in *Developing Remote Clients for Oracle Coherence.*.

This chapter includes the following sections:

- Overview of POF Serialization
- About Portable Types
  Portable Types provide a way to add support for POF serialization to your classes by using annotations and without the requirement to implement serialization code by hand.
- Using the Advanced POF Serialization Options
- Serializing Keys Using POF
- Using Protobuf With POF
  Coherence supports serializing protocol buffers (protobuf) messages to a POF stream. This allows protobuf messages to be used as cache values or as fields inside other POF serialized classes.

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 because data must be moved around the network. POF is a language agnostic binary format. POF is efficient in both space and time and is a cornerstone technology in Coherence. See 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 is restricted to only Java objects.

POF 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 to allow for extracting values without deserializing the whole object. See Using POF Extractors and POF Updaters.

About Portable Types

Portable Types provide a way to add support for POF serialization to your classes by using annotations and without the requirement to implement serialization code by hand.

POF is the preferred serialization format for writing pure Java applications, for the following reasons:

- It is significantly faster than other supported serialization formats, such as Java serialization and `ExternalizableLite`.
- It is significantly more compact that other supported serialization formats, allowing you to store more data in a cluster of a given size, and to move less data over the wire.
- It supports seamless evolution of data classes, allowing you to upgrade various parts of the application (both storage members and clients) independently of one another, without the risk of losing data in the process.

Over the years, although POF remained largely unchanged, POF Reflection was introduced in Coherence 3.5 (2009), allowing you to extract individual attributes from the POF stream through `PofNavigator`. See Using POF Extractors and POF Updaters.

The implementation of POF annotations is dependent on Java reflection which impacts the performance benefits of POF. Because of these limitations, legacy POF Annotations have been deprecated in this release. This feature is now replaced with Portable Types.

This section includes the following topics:

Features and Benefits of Portable Types

Unlike POF annotations, Portable Types:

- Provide a way to add support for POF serialization to your classes by using annotations and without the need to implement serialization code by hand, just as POF Annotations did.
- Implement serialization code at compile-time using byte code instrumentation, and do not rely on Java reflection at runtime. This non-dependency on Java makes them just as fast, but less error-prone, as manually implemented serialization code.
- Support, but do not require explicit registration through the POF configuration file because all the metadata required for POF type registration, such as type identifier, and the serializer class to use, are already available in the `@PortableType` annotation.
- Fully support class evolution.

In fact, Portable Types provide a better and more complete evolution support than the implementation of the `Evolvable` interface by hand.

One of the limitations of the `Evolvable` interface is that it only supports the evolution of the leaf classes in the class hierarchy. Portable Types do not have this limitation, and allow you to not only evolve any class in the hierarchy, but also to evolve the class hierarchy itself, by adding new classes to any level of the class hierarchy.

Understanding Usage Basics

There are only two basic requirements for Portable Types:

- The class must be annotated with the `@PortableType` annotation
- The fields that should be serialized must be annotated with `@Portable` or one of the related annotations (`@PortableDate`, `@PortableArray`, `@PortableSet`, `@PortableList`, or `@PortableMap`)

```
@PortableType(id = 1)
public class Pet
    {
    @Portable
    protected String name;

    // constructors, accessors, etc.
    }

@PortableType(id = 2)
public class Dog extends Pet
    {
    @Portable
    private String breed;

    // constructors, accessors, etc.
    }
```

> **Note:**
>
> Non-annotated fields are considered transient.

Additional attribute-level annotations allow you to control certain serialization behaviors that are specific to the type of the attribute.

For example, `@PortableDate` allows you to control whether you want to serialize date, time, or both when serializing `java.util.Date` instances (by using the `mode` property), and whether time zone information should be included (by using the `includeTimezone` property).

If you are using Java 8 (or later) `java.time` classes, you can derive this information from the class itself by using the `@Portable` annotation instead. For example, `LocalTime` will be serialized as time only, with no time zone information, while the `OffsetDateTime` will be serialized as both date and time, with time zone information.

Similarly, when serializing arrays, collections and maps, POF allows you to use uniform encoding, where the element type (or key and/or value type, in case of maps) is written into the

POF stream only once, instead of once for each element of the collection, resulting in a more compact serialized form.

```
public class MyClass
    {
    @PortableArray(elementClass = String.class)
    private String[] m_stringArray;

    @PortableSet(elementClass = String.class, clazz = LinkedHashSet.class)
    private Set<String> m_setOfStrings;

    @PortableList(elementClass = String.class)
    private List<String> m_listOfStrings;

    @PortableMap(keyClass = Integer.class, valueClass = String.class, clazz =
TreeMap.class)
    private Map<Integer, String> m_uniformMap;
    }
```

As you can see from the examples above, these annotations also allow you to specify the concrete class that should be created during deserialization for a given attribute. If you do not specify the `clazz` property, `HashSet` will be used as the default set type, `ArrayList` as the default list type, and `HashMap` as the default map type.

Class Versioning and Evolution

Coherence is a distributed system, and there is no guarantee that every cluster member, and every client process that connects to the cluster, will have the same version of each and every class. In fact, it is certainly true for systems that use rolling upgrades to avoid any downtime.

It is also neither safe nor practical to upgrade the cluster and all the clients at the same time. Therefore, being able to tolerate different versions of the same class across cluster members and clients is not only desirable, but a necessity for many.

The issue is that when a process that has an older version of the class reads serialized data created from the newer version of the same class, it may encounter some attributes that it knows nothing about. Ideally, it should be able to ignore them and read the attributes it needs and knows about, instead of crashing, but that only solves part of the problem. If the process ignores the unknown attributes completely, when it writes the same data back by serializing an older version of the class that is only aware of some attributes, the process will lose the data it previously received but knows nothing about.

Obviously, this is not a desirable scenario for a system that is intended for long-term data storage. POF supports class evolution in a way that ensures that no data is lost, regardless of how many versions of the same class are present across the various cluster and client processes, and regardless of which of those processes read or write the data. The support for class evolution has been in POF from the very beginning, through the `Evolvable` interface, but Portable Types remove some of the limitations and make the whole process significantly simpler.

Both the class annotation (`@PortableType`) and the attribute annotations (`@Portable` and related annotations) provide a way to specify versioning information that is necessary for class evolution.

At the class level, whenever you modify a class by introducing a new attribute, you should increment the version property of the `@PortableType` annotation.

At the same time, you should specify the `since` attribute that matches the new class version number for any new class attribute. For example, to add the `age` attribute to the `Pet` class, and the `color` attribute to the `Dog` class, change the code provided earlier (see Understanding Usage Basics):

```
@PortableType(id = 1, version = 1)
public class Pet
    {
    @Portable
    protected String name;
    @Portable(since = 1)
    protected int age;
    // constructors, accessors, etc.
    }

@PortableType(id = 2, version = 1)
public class Dog extends Pet
    {
    @Portable
    private String breed;
    @Portable(since = 1)
    private Color color;
    // constructors, accessors, etc.
    }

@PortableType(id = 3, version = 1)
public enum Color
    {
    BRINDLE,
    BLACK,
    WHITE,
    BROWN,
    GRAY
    }
```

Notice that both `version` and `since` properties are zero-based, which allows you to omit them completely in the initial implementation. It also means that for the first subsequent revision, they should be set to `1`.

These are just the defaults. You can certainly set the class and attribute version explicitly to any value even for the initial implementation, if required. The only thing that matters is that you increment the version and set the `since` property to the latest version number whenever you make changes to the class in future.

For example, if in future, you want to add the `height` and `weight` attributes to the `Pet` class, you should simply increment the `version` to `2` and set the `since` property for the new attributes accordingly, as shown below:

```
@PortableType(id = 1, version = 2)
public class Pet
    {
    @Portable
    protected String name;

    @Portable(since = 1)
```

```
    protected int age;

    @Portable(since = 2)
    protected int height;

    @Portable(since = 2)
    protected int weight;

    // constructors, accessors, etc.
    }
```

> **Note:**
>
> Class evolution allows you to add attributes to the new version of the class, but you should never remove the existing attributes because removing them will break the serialization across the class versions.
>
> However, you can remove or deprecate the attribute *accessors* from the class, but you should leave the field itself as is to preserve the backwards compatibility of the serialized form.
>
> Similarly, you should avoid renaming the fields because the default serialization order of fields is determined based on the alphabetical order of field names within a given class version (all fields with the same `since` value).

Using POF Extractors

When using Portable Types, the recommended way to create `ValueExtractors` is to use the `Extractors.fromPof` factory method. The general usage is:

```
Extractors.fromPof(rootClass, propertyPath)
```

If you have a `Pet` class and want to extract the name, you can use:

```
ValueExtractor<Person, String> nameExtractor = Extractors.fromPof(Pet.class,
"name");
```

If you have a `Person` class with `Address` property, you can create an extractor for a nested `` `city` `` property like this:

```
ValueExtractor<Person, String> cityExtractor =
Extractors.fromPof(Person.class, "address.city");
```

Instrumenting the Classes at Compile-Time

Annotating the classes is the first step in the implementation of Portable Types, but that alone is not sufficient. To implement the necessary serialization logic, the classes also need to be instrumented at compile time.

> **Note:**
>
> To use the plug-ins, you need to deploy them to your local artifactory. For more information, see Populating the Maven Repository Manager in *Developing Applications Using Continuous Integration*.

The following options are available for instrumentation:

- Using the Maven POF Plug-In
- Using the Gradle POF Plug-In

Using the Maven POF Plug-In

You can complete this task by using the `pof-maven-plugin` plug-in. You should configure this plug-in in the `pom.xml` file, as shown below:

```
<plugin>
  <groupId>com.oracle.coherence</groupId>
  <artifactId>pof-maven-plugin</artifactId>
  <version>14.1.2-0-0</version>
  <executions>
    <execution>
      <id>instrument</id>
      <goals>
        <goal>instrument</goal>
      </goals>
    </execution>
    <execution>
      <id>instrument-tests</id>
      <goals>
        <goal>instrument-tests</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

The above configuration will discover and instrument all project classes annotated with the `@PortableType` annotation, including test classes. If you do not need to instrument test classes, you can omit the `instrument-tests` execution from the plug-in configuration.

The `pof-maven-plugin` uses the `Schema` support to define the type system that contains all reachable portable types. This type system includes not only project classes that need to be instrumented, but also all portable types that exist in project dependencies. This is necessary because those dependent types may be used as attributes within the project classes, and therefore, need to be serialized appropriately.

In some cases, it may be necessary to expand the type system with the types that are not annotated with the `@PortableType` annotation, and are not discovered automatically. This is typically the case when some of your portable types have 'enum' values, or existing classes that implement the `PortableObject` interface explicitly as attributes.

You can add those types to the schema by creating a `META-INF/schema.xml` file and specifying them explicitly. For example, if you assume that the `Color` class from the earlier code examples (see Class Versioning and Evolution is of 'enum' type, then you will need to

create the following `META-INF/schema.xml` file to register it and allow `pof-maven-plugin` plug-in to instrument the `Dog` class correctly:

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

<schema xmlns="http://xmlns.oracle.com/coherence/schema"
       xmlns:java="http://xmlns.oracle.com/coherence/schema/java"
       external="true">

  <type name="Color">
    <java:type name="petstore.Color"/>
  </type>

</schema>
```

After all these bits and pieces are in place, you can simply run the build as usual:

```
$ mvn clean install
```

You can verify whether the classes are instrumented successfully by checking the Maven output log. You should see something similar to the following:

```
[INFO] --- pof-maven-plugin:21.12:instrument (instrument) @ petstore ---
[INFO] Running PortableTypeGenerator for classes in /projects/petstore/target/classes
[INFO] Instrumenting type petstore.Pet
[INFO] Instrumenting type petstore.Dog
```

After you have successfully instrumented the classes, they are ready to be registered and used.

Creating a POF Index

After the classes are instrumented, the Coherence Maven POF Plug-In will create a class index under `META-INF/pof.idx`. This will allow Coherence to discover POF instrumented classes automatically and add those classes to the `PofContext` during startup. The creation of the class index is enabled by default. You can disable this behavior using the configuration property `indexPofClasses` and set it to `false`.

> **Note:**
>
> Coherence also can load POF instrumented classes without the class index. However, this potentially may be an expensive operation. Therefore, this feature is disabled by default, but can be activated by setting the system-property `coherence.pof.classpath.scanning.enabled` to `true`.

Table 27-1 Maven Plug-In Configuration Properties

Configuration Property	User Property	Description	Default Value
`debug`	`pof.debug`	Add debug statements to serialization code.	`false`
`skip`	`pof.skip`	Whether to skip execution of the Maven POF Plug-In.	`false`

Table 27-1 (Cont.) Maven Plug-In Configuration Properties

Configuration Property	User Property	Description	Default Value
instrumentPofClasses	pof.instrument	Whether to execute POF instrumentation. This is enabled by default but may be disabled in situations where POF classes shall not be instrumented but POF indices may still be needed. This feature may be useful in testing scenarios where a Maven build instruments all relevant classes but you may still wish to create separate indices for specific packages/classes only.	true
indexPofClasses	pof.index	Whether to index PortableType annotated classes in an index file at META-INF/pof.idx.	true
pofIndexFileName	pof.index.name	The index file name and path.	META-INF/pof.idx

pofIndexIncludes	pof.index.includes	You can optionally provide one or more regular expressions to only include classes you need. For example, if you only need classes that end in MyClass, you can provide the following regular expression: {}.*MyClass$. For example:
		
		`<configuration>`
		`...`
		`  <pofIndexIncludes>`
		`  <first>.*Address$</first>`
		`  </pofIndexIncludes>`
		`...`
		`</configuration>`
			Empty, by default.

pofIndexPackages	pof.index.packages	Allows you to include one or more packages when indexing PortableType annotated classes. This is an optional property but limiting the scanning of PortableType annotated classes to a set of packages may speed up indexing substantially.	None.

Using the Gradle POF Plug-In

The Gradle POF Plug-In provides automated instrumentation of classes with the @PortableType annotation to generate consistent (and correct) implementations of Evolvable POF serialization methods.

It is not a trivial exercise to manually write serialization methods that support serializing inheritance hierarchies that support the Evolvable concept. However, with static type analysis, these methods can be generated deterministically.

Generating methods deterministically enables developers to focus on business logic rather than implementing boilerplate code for Evolvable POF serialization methods. For more information about portable types, see About Portable Types.

To use the Gradle POF Plug-In, you need to declare it as a plug-in dependency in the build.gradle file:

```
plugins {
    id 'java'
    id 'com.oracle.coherence' version '14.1.2-0-0'
    }
```

Without any further configuration, the plug-in will add a task named `coherencePof` to your project. The `coherencePof` task will be executed at the end of the `compileJava` task. At the same time, `coherencePof` also depends on the `compileJava` task.

Therefore, calling `gradle compileJava` will execute the `coherencePof` task. Similarly, calling `gradle coherencePof` will execute the `compileJava` task first. By default, the `coherencePof` task will take the build output directory as input for classes to be instrumented, excluding any test classes.

By just adding the plug-in as a dependency, the Gradle POF Plug-In will discover and instrument all project classes annotated with the `@PortableType` annotation, excluding test classes. If you do need to instrument test classes, you can add the `coherencePof` closure and provide additional configuration properties.

This section includes the following topics:

Customizing the Gradle Configuration

You can customize the default behavior of the Coherence Gradle Plug-in by using several optional properties. Simply provide a `coherencePof` closure to the `build.gradle` script containing any additional configuration properties, as shown in the following example:

```
coherencePof {
  debug=true
}
```

In this example, `debug=true` will instruct Coherence to provide more logging output for the instrumented classes.

Available Configuration Properties

The following configuration properties are available for use:

- **Enable Debugging**: Set the Boolean `debug` property to `true` to instruct the underlying `PortableTypeGenerator` to generate the debug code for the instrumented classes. If not specified, this property defaults to `false`.
- **Instrumentation of Test Classes**: Set the Boolean `instrumentTestClasses` property to `true` to instrument test classes. If not specified, this property defaults to `false`.
- **Set a Custom TestClassesDirectory**: Provide a path to a custom test classes directory using the `testClassesDirectory` property. If not set, it will default to the default test output directory.
- **Set a Custom MainClassesDirectory**: Provide a path to a custom classes directory using the `mainClassesDirectory` property. If not set, it will default to the default output directory.

- **Create an index of PortableType annotated classes**: Using the property `indexPofClasses`, you can specify whether `PortableType` annotated classes will be added to an index file at `META-INF/pof.idx`. If not specified, this property defaults to `true`.
- **Limit the packages to scan**: Include one or more Java packages when indexing `PortableType` annotated classes. This is an optional property that you can specify with `pofIndexPackages`. Using the property may speed up indexing. For example:

```
coherencePof {
    pofIndexPackages = ["com.foo.my.package"]
}
```

> **Note:**
>
> Coherence can also load POF instrumented classes without the class index. This, however, may potentially be an expensive operation. Therefore, this feature is disabled by default, but can be activated by setting the system-property `coherence.pof.classpath.scanning.enabled` to `true`.

Using Classes Without the @PortableType Annotation

In some cases, it may be necessary to expand the type system with the types that are not annotated with the `@PortableType` annotation, and are not discovered automatically. This is typically the case when some of the portable types have 'enum' values or existing classes that implement the `PortableObject` interface explicitly as attributes.

You can add those types to the schema by creating a `META-INF/schema.xml` file and specifying them explicitly. For example, if you are using the `Color` class:

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

<schema xmlns="http://xmlns.oracle.com/coherence/schema"
        xmlns:java="http://xmlns.oracle.com/coherence/schema/java"
external="true">

  <type name="Color">
    <java:type name="petstore.Color"/>
  </type>
</schema>
```

Processing the Person Class with the Plug-In - An Example

An example `Person` class (source code shown below) when processed with the plug-in, results in the bytecode shown as the output below:

Example 27-1 Processing the Person Class with the Plug-In

```
@PortableType(id=1000)
public class Person
    {
    public Person()
        {
        }
```

```
    public Person(int id, String name, Address address)
        {
        super();
        this.id = id;
        this.name = name;
        this.address = address;
        }

    int id;
    String name;
    Address address;

    // getters and setters omitted for brevity
    }
```

Inspect the generated bytecode:

```
javap Person.class
```

This should yield the following output:

```
public class demo.Person implements
com.tangosol.io.pof.PortableObject,com.tangosol.io.pof.EvolvableObject {
  int id;
  java.lang.String name;
  demo.Address address;
  public demo.Person();
  public demo.Person(int, java.lang.String, demo.Address);
  public int getId();
  public void setId(int);
  public java.lang.String getName();
  public void setName(java.lang.String);
  public demo.Address getAddress();
  public void setAddress(demo.Address);
  public java.lang.String toString();
  public int hashCode();
  public boolean equals(java.lang.Object);

  public void readExternal(com.tangosol.io.pof.PofReader) throws
java.io.IOException;
  public void writeExternal(com.tangosol.io.pof.PofWriter) throws
java.io.IOException;
  public com.tangosol.io.Evolvable getEvolvable(int);
  public com.tangosol.io.pof.EvolvableHolder getEvolvableHolder();
}
```

The last part of the output shows the additional methods generated by Coherence POF plug-in.

Skipping the Execution of the Task

You can skip the execution of the `coherencePof` task by running the Gradle build using the `-x` flag, as shown in the following example:

```
gradle clean build -x coherencePof
```

Testing the Plug-In Code During Development

During development, it is extremely useful to rapidly test the plug-in code against separate example projects. For this, you can use Gradle's composite build feature. See Composing builds. Therefore, the Coherence POF Gradle Plug-in module itself provides a separate sample module. From within the sample directory, run the following command:

```
gradle clean compileJava --include-build ../plugin
```

This command will not only build the sample but will also build the plug-in. Thus, developers can make plug-in code changes and see changes reflected rapidly in the execution of the sample module.

Alternatively, you can build and install the Coherence Gradle plug-in to the local Maven repository using the following command:

```
gradle publishToMavenLocal
```

For projects to pick up the local changes, ensure the following configuration:

- `Build.gradle`

```
plugins {
  id 'java'
  id 'com.oracle.coherence' version '14.1.2-0-0'
}
```

- `Settings.gradle`

```
pluginManagement {
  repositories {
    mavenLocal()
    gradlePluginPortal()
  }
}
```

Registration and Discovery

Portable Object Format is not a self-describing serialization format. It replaces platform-specific class names with integer-based type identifiers. Therefore, it needs a way of mapping those type identifiers back to the platform-specific classes. This mapping enables portability across platforms, which is the main objective of POF.

To manage the mappings between the type identifiers and concrete types, POF uses `com.tangosol.io.pof.PofContext`:

```
public interface PofContext extends Serializer
    {
    PofSerializer getPofSerializer(int nTypeId);

    int getUserTypeIdentifier(Object o);
    int getUserTypeIdentifier(Class<?> clz);
    int getUserTypeIdentifier(String sClass);

    String getClassName(int nTypeId);
    Class<?> getClass(int nTypeId);

    boolean isUserType(Object o);
    boolean isUserType(Class<?> clz);
    boolean isUserType(String sClass);
    }
```

It is worth noting that `PofContext` extends the `com.tangosol.io.Serializer` interface, which means that you can use any `PofContext` implementation wherever Coherence expects a `Serializer` to be specified, that is, within cache services as a storage-level serializer for data classes, as a transport-level serializer between thin clients and the proxy servers, and so on. The `PofContext` performs the actual serialization by delegating to the appropriate `PofSerializer`, which is obtained through the `PofContext.getPofSerializer` method, based on a type identifier.

There are several built-in implementations of `PofContext`. The `SimplePofContext` allows you to programmatically register type mappings by providing all the metadata needed for serialization, such as type identifier, class, and the `PofSerializer` to use:

```
SimplePofContext ctx = new SimplePofContext();
ctx.registerUserType(1, Pet.class, new PortableTypeSerializer<>(1,
Pet.class));
ctx.registerUserType(2, Dog.class, new PortableTypeSerializer<>(2,
Dog.class));
ctx.registerUserType(3, Color.class, new EnumPofSerializer());
```

Notice that a lot of this information is somewhat repetitive and unnecessary when working with Portable Types, as all the metadata you need can be obtained from the class itself or from the `@PortableType` annotation.

`SimplePofContext` also provides several convenient methods, specifically for Portable Types:

```
ctx.registerPortableType(Pet.class);
ctx.registerPortableType(Dog.class);
```

Or even simpler:

```
ctx.registerPortableTypes(Pet.class, Dog.class);
```

While the `SimplePofContext` is useful for testing and quick prototyping, a `PofContext` implementation that is much more widely used within Coherence applications is `ConfigurablePofContext`.

The `ConfigurablePofContext` allows you to provide type mappings through an external XML file:

```
<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>

    <user-type>
      <type-id>1</type-id>
      <class-name>petstore.Pet</class-name>
    </user-type>

    <user-type>
      <type-id>2</type-id>
      <class-name>petstore.Dog</class-name>
    </user-type>

    <user-type>
      <type-id>3</type-id>
      <class-name>petstore.Color</class-name>
      <serializer>
        <class-name>com.tangosol.io.pof.EnumPofSerializer</class-name>
      </serializer>
    </user-type>

  </user-type-list>

</pof-config>
```

Notice that you did not specify a serializer explicitly for the `Pet` and `Dog` classes. This is because `ConfigurablePofContext` has the logic to determine which of the built-in `PofSerializer` implementations to use depending on the interfaces implemented by, or the annotations present on the specified class. In this case, it will automatically use `PortableTypeSerializer` because the classes have the `@PortableType` annotation.

However, you can make the configuration even simpler by enabling portable type discovery:

```
<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>

    <user-type>
      <type-id>3</type-id>
      <class-name>petstore.Color</class-name>
      <serializer>
```

```
        <class-name>com.tangosol.io.pof.EnumPofSerializer</class-name>
      </serializer>
    </user-type>

  </user-type-list>

  <enable-type-discovery>true</enable-type-discovery>

</pof-config>
```

After you set the `enable-type-discovery` flag to `true`, the `ConfigurablePofContext` will discover all the classes annotated with `@PortableType` and register them automatically, based on the annotation metadata. If you do not use the `Color` enum that has to be registered explicitly, you can even omit the configuration file completely, as the default `pof-config.xml` file that is built into Coherence looks like this:

```
<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>
    <!-- by default just include coherence POF user types -->
    <include>coherence-pof-config.xml</include>
  </user-type-list>

  <enable-type-discovery>true</enable-type-discovery>

</pof-config>
```

> **Note:**
>
> The portable type discovery feature depends on the availability of class index file under `META-INF/pof.idx`. The Coherence POF Maven and Gradle Plug-In can generate the class index file at build time. See Using the Maven POF Plug-In and Using the Gradle POF Plug-In.

Providing IDE Support

After you have annotated, instrumented, and registered the Portable Types as described in Class Versioning and Evolution, Instrumenting the Classes at Compile-Time, and Registration and Discovery, you can use them with Coherence just as easily as you would use plain Java `Serializable` classes, by configuring Coherence services to use `pof` serializer instead of the default `java` serializer.

However, there is still one problem. Serialization code is implemented by the `pof-maven-plugin` plug-in at compile-time, and only if you run the Maven build, which can make it a bit cumbersome to run unit and integration tests within the integrated development environment (IDE).

To solve this problem, Oracle has implemented IDE plug-ins for IntelliJ IDEA and Eclipse. These plug-ins can instrument your classes during incremental or full compilation performed by

your IDE. This enables you to test both the serialization of your classes and the code that depends on it without having to run the Maven build or leave your IDE.

For detailed instructions to install and use the plug-in for your favorite IDE, see the documentation for these plug-ins:

- Coherence IntelliJ Plugin
- Coherence Eclipse Plugin

Using the Advanced POF Serialization Options

Portable types are the recommended way to serialize POF objects. This section describes the various advanced serialization options related to the use of `com.tangosol.io.pof.PofSerializer` interface.
This section includes the following topics:

- Implementing the PofSerializer Interface
- Implementing the Evolvable Interface
- Guidelines for Assigning POF Indexes
- Using POF Object References
- Registering POF Objects
- Configuring Coherence to Use the ConfigurablePofContext Class
- Using POF Extractors and POF Updaters

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:

```
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();

   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);
}
```

Implementing the Evolvable Interface

When implementing the `Evolvable` interface, care must be taken to handle adding new elements in new versions of classes. Newer classes should guard against reading newer elements to prevent mishandling data when reading older versions of the classes.

For example, when implementing `PortableObject`:

```
public void readExternal(PofReader in)
   throws IOException
   {
   m_symbol    = (Symbol) in.readObject(0);
   m_ldtPlaced = in.readLong(1);
   m_fClosed   = in.readBoolean(2);

   if (in.getVersionId() >= 2)
       {
       m_lShares = in.readInt(3);
       }
   }
```

Or, when implementing `PofSerializer`:

```
public Object deserialize(PofReader in)
   throws IOException
   {
   Symbol symbol    = (Symbol)in.readObject(0);
   long   ldtPlaced = in.readLong(1);
   bool   fClosed   = in.readBoolean(2);

   if (in.getVersionId() >= 2)
       {
       lShares = in.readInt(3);
       }

   // mark that reading the object is done
   in.readRemainder();

   if (in.getVersionId() >= 2)
       {
       return new Trade(symbol, ldtPlaced, fClosed, lShares);
       }
   else
       {
       return new Trade(symbol, ldtPlaced, fClosed);
```

```
        }
    }
```

Writing or serializing does not need to do these checks since the checking is done on reading.

Guidelines for Assigning POF Indexes

> **Note:**
>
> These guidelines are not relevant if you are using Portable Types because property index management is handled internally.

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. Indexes that are labeled must still be read and written out in the same order as mentioned above.

Using POF Object References

This section includes the following topics:

- Overview of Using POF Object References
- Enabling POF Object References
- Registering POF Object Identities for Circular and Nested Objects

Overview of 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.

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 use of object identity and references has the following limitations:

- Object references are only supported for user defined object types.
- Object references are not supported for `Evolvable` objects.
- Object references are not supported for keys.
- 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.
- POF objects that use object identity and references cannot be queried using POF extractors. Instead, use the `ValueExtractor` API to query object values or disable object references.
- The use of the `PofNavigator` and `PofValue` API has the following restrictions when using object references:
  - Only read operations are allowed. Write operations result in an `UnsupportedOperationException`.
  - User objects can be accessed in non-uniform collections but not in uniform collections.
  - For read operations, if an object appears in the data stream multiple times, then the object must be read where it first appears before it can be read in the subsequent part of the data. Otherwise, an `IOException: missing identity: <ID>` may be thrown. For example, if there are 3 lists that all contain the same person object, `p`. The `p` object must be read in the first list before it can be read in the second or third list.

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);
```

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 identity 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 POF User Type 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

This section includes the following topics:

- Overview of Using the ConfigurablePofContext Class
- Configuring the ConfigurablePofContext Class Per Service
- Configuring the ConfigurablePofContext Class for All Services
- Configuring the ConfigurablePofContext Class for a JVM Instance
- Making POF Configuration Files Discoverable at Runtime

Overview of Using 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.

Configuring 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.

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. See serializer.

```
 <distributed-scheme>
    <scheme-name>example-distributed</scheme-name>
    <service-name>DistributedCache</service-name>
    <serializer>pof</serializer>
 </distributed-scheme>
```

Configuring 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.

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. See 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>
   ...
```

Configuring the ConfigurablePofContext Class for a JVM Instance

You can configure an entire JVM instance to use POF, using the following system properties:

- `coherence.pof.enabled=true` - Enables POF for the entire JVM instance.
- `coherence.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`).

Making POF Configuration Files Discoverable at Runtime

In Java applications, where new modules may be added at deploy time or runtime, the full set of required POF configuration files may not be known ahead of time. Hence, it may not be possible to create a POF configuration file with all the correct `<include>` elements. To allow for this type of use case, it is possible to make POF configuration files discoverable at runtime by the `ConfigurablePofContext` class instead of needing to put them inside `<include>` elements.

To make a configuration file discoverable, create a class that implements `com.tangosol.io.pof.PofConfigProvider`. The `PofConfigProvider` has a single method that must be implemented to return the name of the configuration file. At runtime, the `ConfigurablePofContext` class uses the Java `ServiceLoader` to discover implementations of `PofConfigProvider` and load their provided POF configuration files, exactly as if they had been added in `<include>` elements.

For example, the `RuntimeConfigProvider` class below provides `discovered-pof-config.xml` as the configuration file name. At runtime, the `ConfigurablePofContext` class loads the `discovered-pof-config.xml` POF configuration file.

RuntimeConfigProvider.java

```
package com.oracle.coherence.examples;

import com.tangosol.io.pof.PofConfigProvider;

public class RuntimeConfigProvider
        implements PofConfigProvider
    {
    @Override
    public String getConfigURI()
        {
        return "discovered-pof-config.xml";
        }
    }
```

To make the `RuntimeConfigProvider` class discoverable:

- Create a file `META-INF/services/com.tangosol.io.pof.PofConfigProvider` containing the following:

  ```
  com.oracle.coherence.examples.RuntimeConfigProvider
  ```

- If using Java Modules, add it to the `module-info.java` file:

  ```
  module com.oracle.coherence.examples {
      provides com.tangosol.io.pof.PofConfigProvider
          with com.oracle.coherence.examples.RuntimeConfigProvider;
  }
  ```

If required, a `PofConfigProvider` implementation may return multiple POF configuration files by overriding the `PofConfigProvider.getConfigURIs()` method. In this case, the singular `getConfigURI()` is not called.

In the example below, the `RuntimeConfigProvider`, the `getConfigURIs()` method returns the POF configuration file names `discovered-pof-config.xml` and `additional-pof-config.xml`, both of which are loaded by the `ConfigurablePofContext` at runtime.

```
package com.oracle.coherence.examples;

import com.tangosol.io.pof.PofConfigProvider;

import java.util.Set;

public class RuntimeConfigProvider
        implements PofConfigProvider
    {
    @Override
    public Set<String> getConfigURIs()
        {
        return Set.of("discovered-pof-config.xml",
                "additional-pof-config.xml");
        }

    @Override
    public String getConfigURI()
        {
        return null;
        }
    }
```

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 values 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 and 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.

This section includes the following topics:

- Using POF Extractors
- Using POF Updaters

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.

Serializing Keys Using POF

Key objects, such as value objects, can be serialized using POF. However, there are a few points to consider when serializing keys:

- POF defines a cross-platform object format, it cannot always provide a symmetrical conversion. That is, when a serialized object is deserialized, the object type may be different than the original type. This occurs because some data types in Java do not have equivalents in the .NET and C++ platforms. As a result, avoid using classes that potentially have an asymmetrical POF conversion as keys, or parts of keys, for caches and other Java collections.
- Avoided using the `java.util.Date` type. POF is designed to serialize to `java.sql.Timestamp` (which extends `java.util.Date`). The wire formats for those two classes are identical, and a deserialization of that wire representation always results in a `java.sql.Timestamp` instance. Unfortunately, the `equals` method of those two base classes breaks the symmetry requirement for keys in Java collections. That is, if you have two objects: `D` (of `java.util.Date`) and `T` (of `java.sql.Timestamp`) that are equivalent from the POF wire format perspective, then `D.equals(T)` yields true, while `T.equals(D)` yields false. Therefore, the use of `java.util.Date` must be avoided. Use a `Long` representation of the date or the `java.sql.Timestamp` type to avoid breaking the key symmetry requirement.
- Keys that are using POF object references cannot be serialized. In addition, POF object references support circular references. Therefore, you must ensure that your key class does not have circular references.

Using Protobuf With POF

Coherence supports serializing protocol buffers (protobuf) messages to a POF stream. This allows protobuf messages to be used as cache values or as fields inside other POF serialized classes.

> **Note:**
>
> Protobuf messages should not be used as cache keys or in fields of cache key classes. The protobuf format does not guarantee a consistent serialization format, so the same message serialized twice may result in a different binary value. They also do not guarantee a stable hash code or equality. This makes them unsuitable to use as cache keys.

To use protobuf messages in POF add the `coherence-protobuf` module as an application dependency:

```
<dependencyManagement>
    <dependency>
        <groupId>${coherence.groupId}</groupId>
        <artifactId>coherence-bom</artifactId>
        <version>${coherence.version}</version>
    </dependency>
</dependencyManagement>

<dependency>
```

```
    <groupId>${coherence.groupId}</groupId>
    <artifactId>coherence-protobuf</artifactId>
</dependency>
```

Replacing `${coherence.groupId}` with the correct `groupId` (`com.oracle.coherence` for commercial Coherence or `com.oracle.coherence.ce` for the CE version) and `${coherence.version}` with the version being used, for example, 14.1.2-0-0.

When the `coherence-protobuf` module is on the application class path, the Coherence POF serializer automatically discovers the configuration for serializing protobuf messages. Any protobuf message class that is auto-generated from protocol files will be supported. The default Protobuf POF serializer will look for a static method on the message class named, `parseFrom` (taking a byte array as a parameter), which is used to create a new instance of the class on deserialization. All classes generated by the protobuf Java code generator will contain this method.

28

Querying Data in a Cache

You can perform queries and use indexes to retrieve data in a cache that matches certain 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.
This chapter includes the following sections:

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. Queries currently apply only to 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 can be evaluated locally against unindexed or indexed data. For Partitioned caches, queries are typically performed in parallel across the cluster and use indexes. Access to unindexed attributes requires object deserialization (though indexing on other attributes can reduce the number of objects that must be evaluated). Lastly, Coherence includes a Cost-Based Optimizer (CBO) and also provides support for trace and explain reports that help ensure the efficiency of a query.

> **Note:**
>
> All classes that implement the Filter interface must explicitly implement the `hashCode()` and `equals()` methods in a way that is based solely on the object's serializable state. This is particularly important when using filters in `ObservableMap.addMapListener` and other similar places where the filter gets serialized, transported over the network, and eventually used as a map key. The same is valid for ValueExtractor implementations.

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). In the past, most developers needed to use only the `ReferenceExtractor` implementation of this interface and pass a method name as a string. The replacement for `ReflectionExtractor` is `UniversalExtractor` that works for both properties and method names. Class `UniversalExtractor` works for POJOs, Java Records, and JSON Object. Class `com.tangosol.util.Extractors` provides simple factory methods for various `ValueExtractor` classes and now uses `UniversalExtractor` because it works for all supported values. With the advent of Java 8, Oracle strongly recommends using Method References to provide compile-time type safety. For example:

```
ValueExtractor extractor = ValueExtractor.of(Customer::getName);
```

Coherence provides a simple filter DSL, through the `com.tangosol.util.Filters` class. This class contains many factory methods for easy creation of filters. Usage of these filters makes the code more readable, especially if imported statically. Therefore, their use is strongly encouraged instead of direct construction of `Filter` classes.

To query a cache that contains objects with `getName` attributes, a `Filter` must be used. The filter below uses the `Filters.equals` factory method and takes a method reference and a value to compare with.

```
Filter filter = Filters.equal(Customer::getName, "Bob Smith");
```

The following example shows a routine to select the entries of a cache that satisfy a particular filter:

```
for (Map.Entry<Integer, Person> entry : cache.entrySet(filter)) {
   Integer key = entry.getKey();
   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 Queries

Coherence includes many pre-built filters available in the `com.tangosol.util.Filters` class. Example 28-1 demonstrates how to create a query and uses the `Filters.greaterEqual` factory method.

Example 28-1 Querying the Cache with a Filter

```
Filter filter = Filters.greaterEqual(Person::getId, 18);

   for (Map.Entry<Integer, Person> entry : cache.entrySet(filter)) {
      Integer key = entry.getKey();
      Person person = entry.getValue();
      System.out.println("key=" + key + " person=" + person);
   }
```

> **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);
> ```

Efficient Processing of Filter Results

You can query large data sets in batches to guard against running out of heap space.

Example 28-2 illustrates a pattern to process query results when using large data sets. 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:**
>
> The `LimitFilter` API can process results in parts, similar to the example below. 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.

Example 28-2 Processing Query Results in Batches

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

```
    // Search for entries that start with 'c'
    Filter query = Filters.like(Extractors.identity(), "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());
        }
    }
```

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.
This section includes the following topics:

- 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 28-3 demonstrates how to create an index:

Example 28-3 Sample Code to Create an Index

```
NamedCache<Integer, Customer> myCache = CacheFactory.getCache("MyCache");
myCache.addIndex(Customer::getOutstandingBalance, 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`. See StorageManagerMBean in *Managing Oracle 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.

This section includes the following topics:

- Implementing the MapIndex Interface
- Implementing the IndexAwareExtractor Interface
- Using a Conditional Index

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 = Extractors.extract("lastName");
Filter filter = new NotEqualsFilter(Person::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:

```
// key set(Filter filter)
Set setKeys = cache.keySet(filter);
Set setPageKeys = new HashSet();
int PAGE_SIZE = 100;
for (Iterator iter = setKeys.iterator(); iter.hasNext();)
    {
```

```
    setPageKeys.add(iter.next());
    if (setPageKeys.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. The use of `LimitFilter` has two assumptions for it to function correctly:

- There are no concurrent modifications to the data set in question.
- Data is evenly distributed across all storage nodes of the cluster, such that each node has a fair sample of the entire set of data.

> **Note:**
>
> In the case of redistribution, data is not evenly distributed across all storage nodes and results in the wrong result set for any incoming query.

```
int pageSize = 25;
Filter filter = Filters.greaterEqual(Person::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 ...
    }
```

Using Sorted Views

Sorted Views allows you to create a client-side view of data managed in Coherence that is sorted based on the natural sort order of the entry values or on the provided `Comparator`.

- Creating a Sorted View

Creating a Sorted View

You can create a Sorted View by using the following `ViewBuilder` API:

```
NamedMap<String, String> states       = session.getMap("states");
NamedMap<String, String> sortedStates = states.view().sorted().build();
```

1. Obtain a reference to a distributed cache that stores master copy of state names which is keyed by two-letter state code.
2. Create a client-side view of states that is sorted by the natural order of map values. For example, state name.

The contents of the view are kept in sync by Coherence automatically like the other views, so any changes made to the master list of states is reflected in each client-side view automatically.

You can also create a view for more complex data types by passing a custom `Comparator` to the following `sorted` method:

```
NamedMap<Long, Person> people       = session.getMap("people");
NamedMap<Long, Person> sortedPeople = people.view()
                                            .sorted(Comparator.comparing(Perso
n::getAge).reversed())
                                            .build();
```

The above example gives you a view of all people sorted by age from the oldest to the youngest person.

You can perform all other operations that the `ViewBuilder` API supports, such as filtering entries in a view before they are sorted. For example, to create a view of all women sorted by age from youngest to oldest, define the view as follows:

```
NamedMap<Long, Person> people       = session.getMap("people");
NamedMap<Long, Person> sortedPeople = people.view()
                                            .filter(Filters.equal(Person::getG
ender, Gender.FEMALE))
                                            .sorted(Comparator.comparing(Perso
n::getAge))
                                            .build();
```

> **Note:**
>
> The sorting is always performed on the client after the data is retrieved from the server. In most cases, this happens regardless of the order that you specify in the operations.

For example, if you reversed the order of the `filter` and `sorted` operations, and created a view as follows for the above example it works the same:

```
NamedMap<Long, Person> people       = session.getMap("people");
NamedMap<Long, Person> sortedPeople = people.view()
                                            .sorted(Comparator.comparing(Perso
n::getAge))
                                            .filter(Filters.equal(Person::getG
ender, Gender.FEMALE))
                                            .build();
```

The only exception is the `map` operation, as it changes the type of values that are stored on the client by transforming them on the server to reduce the amount of data that is transferred across the network. As you can only sort what you have, the sort operation always has to be specified after the `map` operation in order to use the correct value type:

```
NamedMap<Long, Person> people      = session.getMap("people");
NamedMap<Long, Name>   sortedNames = people.view()
                                            .filter(Filters.equal(Person::getG
ender, Gender.MALE))
                                            .map(Person::getName)
                                            .sorted(Comparator.comparing(Name:
:getLast)
                                                                   .thenComparing(N
ame::getFirst))
                                            .build();
```

The above example extracts the `Names` of all the men on the server and then sorts them from the first name by the last name and then by the first name on the client.

It is recommended to specify the operations in the order of their execution such as `filter`, `map`, and `sorted` because that is the order in which the operations are executed.

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.

```
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 = Filters.containsAll(Document::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 the following example, the extractor first uses a property reference to access the `name` property on each cached `Person` object, and then uses reflection to call the `length` method on the returned `String`.

```
ValueExtractor extractor = Extractors.chained("name", "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 `name.trim().length()`.

POF extractors and POF updaters offer the same functionality as `ChainedExtractors` through the use of the `SimplePofPath` class. See Using POF Extractors and POF Updaters.

Options to Skip Query Result Consistency Check

By default, Coherence will ensure query result with entries that match the provided filter. However, the consistency check can result in repeat query re-evaluations if the targeted partitions are modified concurrently.

The following options allow you to decide if Coherence should relax the consistency check for a faster response:

- At aggregator level, by Overriding the characteristics() of the individual aggregator. For example:

  ```
  @Override
  public int characteristics()
       {
       return super.characteristics() | ALLOW_INCONSISTENCIES;
       }
  ```

- At JVM level via a JVM argument:
  Pass system property `-Dcoherence.query.retry = 0`.

For queries with the aggregate method, you can use both these options for a quick response. For queries using filters (for example, `entrySet`), you can use the JVM option to skip query result re-evaluation.

Evaluating Query Cost and Effectiveness

You can create 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 StorageManagerMBean in *Managing Oracle Coherence* for details on viewing query-based statistics.
This section includes the following topics:

- 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<String, Person> cache = CacheFactory.getCache("people");
cache.addIndex(Person::getAge, true, null);

AllFilter filter = new AllFilter(new Filter[]
    {
    Filters.equal(Person::getAge, 16).or(Filters.equal(Person::getAge, 19)),
    Filters.equal(Person::getLastName, "Smith"),
    Filters.equal(Person::getFirstName, "Bob"),
    });

QueryRecorder<String, Person> agent = new QueryRecorder<>(RecordType.EXPLAIN);
Object resultsExplain = cache.aggregate(filter, agent);
System.out.println("\nExplain Plan=\n" + resultsExplain + "\n");
```

To create a trace record, change the `RecordType` parameter to `TRACE`:

```
QueryRecorder<String, Person> 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, an index is added for `getAge`. See Running The Query Record Example.

```
NamedCache<String, Person> cache = CacheFactory.getCache("people");
cache.addIndex(Person::getAge, true, null);

AllFilter filter = new AllFilter(new Filter[]
    {
    Filters.equal(Person::getAge, 16).or(Filters.equal(Person::getAge, 19)),
    Filters.equal(Person::getLastName, "Smith"),
```

```
    Filters.equal(Person::getFirstName, "Bob"),
    });
```

This section includes the following topics:

- Query Explain Plan Record
- Query Trace Record

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 28-4 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 28-4 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 28-4 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 28-5 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 28-5 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 28-5 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 28-6 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 com.tangosol.util.filter.AllFilter;

import java.io.Serializable;

import java.util.HashMap;
import java.util.Map;

import static com.tangosol.util.Filters.equal;
import static com.tangosol.util.aggregator.QueryRecorder.RecordType;

public class QueryRecordExample
    {
    public static void main(String[] args)
        {
        System.setProperty("coherence.wka", "127.0.0.1");
        System.setProperty("coherence.distributed.localstorage", "true");

        // ((age == 16 OR age == 19) AND lastName == 'Smith' AND firstName ==
'Bob')
        AllFilter filter = new AllFilter(new Filter[]
            {
            equal(Person::getAge, 16).or(equal(Person::getAge, 19)),
            equal(Person::getLastName, "Smith"),
            equal(Person::getFirstName, "Bob"),
            });

        NamedCache<String, Person> cache = CacheFactory.getCache("people");
        cache.addIndex(Person::getAge, true, null);

        populateCache(cache);

        testExplain(cache, filter);
```

```
        testTrace(cache, filter);
        }

    public static void testExplain(NamedCache<String, Person> cache, Filter
filter)
       {
       QueryRecorder<String, Person> agent = new
QueryRecorder<>(RecordType.EXPLAIN);
       Object resultsExplain = cache.aggregate(filter, agent);
       System.out.println("\nExplain Plan=\n" + resultsExplain + "\n");
       }

    public static void testTrace(NamedCache<String, Person> cache, Filter
filter)
        {
        QueryRecorder<String, Person> agent = new
QueryRecorder<>(RecordType.TRACE);
        Object resultsExplain = cache.aggregate(filter, agent);
        System.out.println("\nTrace =\n" + resultsExplain + "\n");
        }

    private static void populateCache(NamedCache<String, Person> cache)
        {
        Map<String, Person> buffer = new HashMap<>();
        for (int i = 0; i < 1500; ++i)
            {
            Person person = new Person(i % 3 == 0 ? "Joe" : "Bob",
                    i % 2 == 0 ? "Smith" : "Jones", 15 + i % 10);
            buffer.put("key" + i, person);
            }
        cache.putAll(buffer);
        }

    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 final String m_sFirstName;
        private final String m_sLastName;
        private final int    m_nAge;
        }
    }
```

29

Using Continuous Query Caching

You can use continuous query caching to ensure that a query always retrieves the latest results from a cache in real-time.
This chapter includes the following sections:

Overview of Using Continuous Query Caching

Queries provide the ability to obtain a point in time query result from a Coherence cache and it is possible to receive events that would change the result of that query. The continuous query feature 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. See Querying Data in a Cache.
Continuous query is implemented 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 continuous query is a cached query result that never gets out-of-date.

Understanding Use Cases for Continuous Query Caching

There are several 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 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.

Understanding the Continuous Query Cache Implementation

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 Java `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 you must define when using continuous query caching.

- The underlying cache that the continuous query cache is based on
- A query of the 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` instance, 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");
```

The query is the same type of query that would be used to filter data. For example:

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

See Querying Data in a Cache.

Normally, to query a cache, a method from `QueryMap` is used. For example, to obtain a snapshot of all open trades for a trader object:

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

Similarly, the continuous query cache is constructed from those same two pieces:

```
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. If a continuous query cache is only used for a short amount of time, then an application must explicitly 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:

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

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

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

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. If the continuous query cache has a non-null transformer, it is inferred that cache values should be stored locally, then the `CacheValues` property is automatically set to `true`.

Listening to a Continuous Query Cache

You can use synchronous and asynchronous event listeners to observe a continuous query cache.
This section includes the following topics:

- Overview of Listening to a Continuous Query Cache
- Achieving a Stable Materialized View
- Support for Synchronous and Asynchronous Listeners

Overview of Listening to a Continuous Query Cache

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:

```
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:

```
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:

```
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:

```
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 Using Map Events.

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 Using Continuous Query Caching.

Making a Continuous Query Cache Read-Only

The `ContinuousQueryCache` can be made into a read-only cache. For example:

```
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.

30
Processing Data In a Cache

You can use entry processors and aggregators to perform data grid processing across a cluster.These data grid features perform in a similar manner to other map-reduce patterns and allow the processing of large amounts of data at very low latencies.
This chapter includes the following sections:

Overview of 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 large amounts 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.
This section includes the following topics:

Performing Targeted Processing

Coherence provides for the ability to execute an agent against an entry in any map of data managed by a data grid:

```
map.invoke(key, agent);
```

In the case of partitioned data, the agent executes on the grid node that owns the data. 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 (at most a few hundred bytes) than to handle distributed concurrency control, coherency and data updates.

For request and response processing, the agent returns a result:

```
Object oResult = map.invoke(key, agent);
```

Coherence, as a data grid, determines the location to execute the agent based on the configuration for the data topology. It moves the agent to the determined location, executes the

agent (automatically handling concurrency control for the item while executing the agent), backs up the modifications (if any), and returns a result.

Performing Parallel Processing

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 and 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. It then moves the agent to the determined locations, executes the agent (automatically handling concurrency control for the item(s) while executing the agent), backs up the modifications (if any), and returns the coalesced results. See Performing Data Grid Aggregation.

Performing Query-Based Processing

Coherence supports the ability to query across the entire data grid. See Querying Data in a Cache. For example, in a trading system it is possible to query for all open `Order` objects for a particular trader:

```
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 the use of parallel executions in the data grid, Coherence provides 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 agent against the entries:

```
map.invokeAll(filter, agent);
```

For request and response processing, the agent returns one result for each key processed:

```
Map mapResults = map.invokeAll(filter, agent);
```

Coherence combines parallel query and parallel execution to achieve query-based agent invocation against a data grid.

Performing Data-Grid-Wide Processing

Passing an instance of `AlwaysFilter` (or `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 and 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.

Using Agents for Targeted, Parallel and Query-Based Processing

You can process data in a cache using agents that are commonly referred to as entry processors. Coherence includes many predefined entry processors that can be used to perform many common operations.
This section includes the following topics:

Overview of Entry Processor Agents

Agents implement the `EntryProcessor` interface, typically by extending the `AbstractProcessor` class. Coherence includes the following predefined `EntryProcessor` implementations that are included in the `com.tangosol.util.processor` package:

- `AbstractProcessor` - an abstract base class for building an `EntryProcessor`
- `AsynchronousProcessor` - A wrapper class that allows for an asynchronous invocation of an underlying entry processor. See Processing Entries Asynchronously.
- `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`
- `ConditionalPut` - performs an `Entry.setValue` operation if the specified condition is satisfied
- `ConditionalPutAll` - performs an `Entry.setValue` operation for multiple entries that satisfy the specified condition
- `ConditionalRemove` - performs an `Entry.remove` operation if the specified condition is satisfied
- `ExtractorProcessor` - extracts and returns a value (such as a property value) from an object stored in an `InvocableMap`
- `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
- `PreloadRequest` - performs an `Entry.getValue` call. No results are reported back to the caller. The processor provides a means to load an entry or a collection of entries into the cache using a cache loader without incurring the cost of sending the value(s) over the

network. If the corresponding entry (or entries) already exists in the cache, or if the cache does not have a loader, then invoking this processor has no effect.

- `PriorityProcessor` - explicitly controls the scheduling priority and timeouts for execution of `Entryprocessor` methods.
- `PropertyProcessor` - an abstract base class for `EntryProcessor` implementations that depend on a `PropertyManipulator`. The `NumberIncrementor` and `NumberMultiplier` entry processors extend this processor.
- `UpdaterProcessor` - updates an attribute of an object cached in an `InvocableMap`.
- `VersionedPut` - performs an `Entry.setValue` operation if the version of the specified value matches the version of the current value. For a match, the processor increments the version indicator before the value is updated. Entry values must implement the `Versionable` interface.
- `VersionedPutAll` - performs an `Entry.setValue` operation only for entries whose versions match to versions of the corresponding current values. For a match, the processor increments the version indicator before each value is updated. Entry values must implement the `Versionable` interface.

The `EntryProcessor` interface (contained within the `InvocableMap` interface) contains only two methods: `process` and `processAll`. Historically, the `AbstractProcessor` provided the default implementation of the `processAll` method.

However, as of Coherence 14c (14.1.2.0.0), `processAll` will delegate to the default implementation by the `EntryProcessor` interface. Therefore, custom Entry Processor implementations should implement only the `EntryProcessor` interface (no need to extend `AbstractProcessor`).

> **Note:**
>
> If the `processAll` call throws an exception, changes are only made to the underlying `Map` for entries that were removed from the `setEntries`. Changes that are made to the remaining entries are not processed.

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.

Processing Entries Using Lambda Expressions

Lambda expressions can be used as entry processors and can result in more concise client code that does not require the processor to be serialized or registered in a POF configuration file. The following example creates an entry processor as a lambda expression and uses the entry processor within the `invokeAll` method:

```
InvocableMap.EntryProcessor<ContactId, Contact, Void> processor = (entry) ->
   {
      Contact contact = entry.getValue();
      contact.setFirstName(contact.getFirstName().toUpperCase());
      entry.setValue(contact);
      return null;
```

```
   };

cache.invokeAll(processor);
```

The following example creates an entry processor as a lambda expression directly within the `invokeAll` method.

```
Address addrWork = new Address("201 Newbury St.", "Yoyodyne, Ltd.",
   "Boston", "MA", "02116", "US");

ValueExtractor extractor =
   Lambda.extractor(Contact::getHomeAddress).andThen(Address::getState);
Filter filter = equal(extractor, "MA");

addrWork.setStreet1("200 Newbury St.");

cache.invokeAll(filter, entry ->
   {
   Contact contact = entry.getValue();
   contact.setWorkAddress(addrWork);
   entry.setValue(contact);
   return null;
   });
```

Alternatively, you can invoke lambda expressions by using the functional-style methods, such as `compute()`, `computeIfAbsent()`, and so on, that are provided through `java.util.Map`.

Using these functional-style methods leads to slightly more concise code. For example, the `compute()` method implicitly updates the value in the cache to whatever you return. Whereas, when you use `invoke()`, you must call `entry.setValue()` explicitly. A possible drawback using `compute()` is that it returns the entire object, whereas with `invoke()` you can specify what the return value should be.

Comparing the two options: functional-style methods such as `compute()` are more focused and part of the standard Java API. However, using `invoke()` is the most generic, yet slightly more verbose, option.

> **Note:**
>
> Lambda expressions cannot be nested. For example:
>
> ```
> cache.invoke(filter, entry -> {Runnable r = () ->
> System.out.println("");
> r.run();}
> ```

This section includes the following topics:

- About Lambdas in a Distributed Environment
- Configuring Lambda Serialization Mode
- Considerations for a Rolling Upgrade

About Lambdas in a Distributed Environment

Executing lambda expressions in distributed environments can be problematic due to the static nature of lambdas. Only the metadata that describes the lambda is sent across the wire. Therefore, the same compiled code is required in both the client and server classpath/modulepath. Any changes or additions of new lambda expressions on the client require a redeployment and restart of both the client and the server. In addition, synthetic lambda method names are not stable across class versions. Therefore, all cluster members must have the exact version of a class and must be upgraded, including extend clients, at the same time.

To overcome these limitation, a dynamic implementation for lambdas is provided. The dynamic implementation sends both the lambda metadata and the actual byte code to be executed. Client-side byte code is parsed and then from it a new lambda class is generated. On the server, a lambda class is created based on the byte code received from the client, and executed. The dynamic implementation:

- allows modification of existing (or the introduction of new) behavior without the need for redeployment or server restart.
- eliminates the issues related to lambda naming instability.
- allows multiple different versions of a class throughout the cluster.

To ensure that the dynamic implementation works correctly, do not refer to named methods and constructors in a lambda expression, because method and constructor references are always treated as static lambdas. In addition, the dynamic implementation captures only enclosing method arguments and local variables.

Coherence has been using dynamic lambdas implementation in a distributed environment exclusively since Coherence release 12.2.1. To provide users the ability to choose between the convenience and flexibility of dynamic lambdas and the enhanced security of static lambdas, a configuration option has been added to enable selection between using dynamic or static lambda serialization mode.

Configuring Lambda Serialization Mode

Dynamic lambdas present a security vulnerability due to reliance on remote code deployment across the distributed environment.

You can use the system property `coherence.lambdas` to explicitly configure `static` or `dynamic` lambda serialization mode, as shown below:

```
-Dcoherence.lambdas=static
```

Or, in the operational configuration file, you can explicitly configure the `cluster-config` subelement `lambdas-serialization` to `static` or `dynamic` lambda serialization mode, as shown below:

```
<cluster-config>
     …
    <lambdas-serialization>static</lambdas-serialization>
     …
</cluster-config>
```

There is nothing more to do if one configures dynamic lambda serialization mode. Dynamic lambdas send byte code between Coherence members in the distributed environment.

There is an additional step needed when using static lambdas serialization mode. The static lambda serialization only sends lambda metadata references between members in the distributed environment and the same class files containing the lambda expressions must be in the Java classpath/modulepath of all JVMs in the distributed environment, or the static lambda metadata reference will not resolve in the Coherence receiving member.

Example: When the class defining the serialized lambda is missing from the server's java classpath/modulepath context

Here is an example print stack trace of an exception handled by the client invoking the lambda:

```
Failed request execution for DistributedCache service on Member(Id=3,,…))
java.lang.ClassNotFoundException: lambda.AbstractRemoteFunctionTests
…
Caused by: java.lang.RuntimeException: Failed to deserialize static lambda lambda/
RemoteTests$lambda$testLambdaInvoke$393ba5d0$1(Lcom/tangosol/util/
InvocableMap$Entry;)Ljava/lang/Object; due to missing context class lambda.RemoteTests.
At
com.tangosol.internal.util.invoke.lambda.StaticLambdaInfo.toSerializedLambda(StaticLambda
Info.java:430
```

The above failure is resolved by ensuring that the missing class `lambda.RemoteTests` is in all server classpaths/modulepaths so that the static lambda reference sent by the invoking client can be resolved on the server side.

Example: When static lambda names are different due to different versions of the Java class in the client and server

Client side code fragment:

```
public void testSimple() {
    NamedCache cache = …;
    cache.invoke(1, entry -> entry.setValue(entry.getValue() + 1));
    cache.invoke(2, entry -> entry.setValue(entry.getValue() + 1));
```

Server side code fragment:

```
public void testSimple() {
    NamedCache cache = …;
    cache.invoke(1, entry -> entry.setValue(entry.getValue() + 1));
```

Here is an example print stack trace of an exception handled by the client invoking the lambda. The second static lambda expression is in the client side code and cannot be resolved on the server side.

```
(Wrapped: Failed request execution for DistributedCache service on Member(Id=1,…))
java.lang.IllegalArgumentException: Invalid lambda deserialization
…
Caused by: java.lang.RuntimeException: Exception resolving static lambda
SerializedLambda[capturingClass=class lambda.AbstractRemoteFunctionTests,
functionalInterfaceMethod=com/tangosol/util/InvocableMap$EntryProcessor.process:(Lcom/
tangosol/util/InvocableMap$Entry;)Ljava/lang/Object;, implementation=invokeStatic lambda/
AbstractRemoteFunctionTests.lambda$testSimple$393ba5d0$2:(Lcom/tangosol/util/
InvocableMap$Entry;)Ljava/lang/Object;, instantiatedMethodType=(Lcom/tangosol/util/
InvocableMap$Entry;)Ljava/lang/Object;, numCaptured=0]
```

Resolution is to configure the server with same version of Java class containing the lambda as the client is using. For this error scenario, the client had two static lambda invocations in the

`AbstractRemoteFunctionTests#simpleTest` method and the server Java class version had only one invocation, thus, the invocation target `lambda/AbstractRemoteFunctionTests.lambda$testSimple$393ba5d0$2` was not found on the server classpath/modulepath.

Example: When static lambda expression has different number of capture arguments on the client and server version of the Java class

Client side code fragment:

```
public void testSimple() {
    NamedCache cache = …;
    Integer nInc = 5;
    cache.invoke(1, entry -> entry.setValue(entry.getValue() + nInc));
```

Server side code fragment:

```
public void testSimple() {
    NamedCache cache = …;
    cache.invoke(1, entry -> entry.setValue(entry.getValue() + 1));
```

Here is an example print stack trace of an exception handled by the client invoking the lambda. The client side code fragment has one capture argument, `nInc`, in the lambda expression and it can not be resolved on the server side that has the lambda expression with no capture arguments.

```
<Info> (thread=main, member=2): (Wrapped: Failed request execution for DistributedCache
service on Member(Id=1, Timestamp=2021-08-06 09:37:57.057, Address=127.0.0.1:8888,
MachineId=10131, Location=machine:localhost,process:19159,
Role=RemoteFunctionJavaStaticLambda_Simple)) java.lang.IllegalArgumentException: Invalid
lambda deserialization
…
Caused by: java.lang.RuntimeException: Exception resolving static lambda
SerializedLambda[capturingClass=class lambda.AbstractRemoteFunctionTests,
functionalInterfaceMethod=com/tangosol/util/InvocableMap$EntryProcessor.process:(Lcom/
tangosol/util/InvocableMap$Entry;)Ljava/lang/Object;, implementation=invokeStatic lambda/
AbstractRemoteFunctionTests.lambda$testSimple$6572cde9$1:(Ljava/lang/Integer;Lcom/
tangosol/util/InvocableMap$Entry;)Ljava/lang/Object;, instantiatedMethodType=(Lcom/
tangosol/util/InvocableMap$Entry;)Ljava/lang/Object;, numCaptured=1]
    at
com.tangosol.internal.util.invoke.lambda.StaticLambdaInfo.createLambda(StaticLambdaInfo.j
ava:398)
```

Resolution is to configure the server with the same version of the Java class containing the lambda as the client is using. For this error scenario, the client had two static lambda invocations in the `AbstractRemoteFunctionTests#simpleTest` method and the server Java class version had only one invocation, thus, the invocation target `lambda/AbstractRemoteFunctionTests.lambda$testSimple$393ba5d0$2` was not found on the server classpath/modulepath.

Example: Failed dynamic lambda invocation to a static lambda configured server

Following is an example print stack trace of an exception handled by the client invoking the remote lambda expression:

```
(Wrapped: Failed request execution for DistributedCache service on Member(Id=3,…))
java.io.NotSerializableException: com.tangosol.internal.util.invoke.RemoteConstructor
…
```

```
at
com.tangosol.coherence.component.util.daemon.queueProcessor.service.grid.partitionedServi
ce.PartitionedCache.onInvokeRequest(PartitionedCache.CDB:93)
```

Resolution is to configure the client to the static lambda serialization mode and to ensure the server classpath/modulepath contains the client side Java class containing the lambda expression. If the server was incorrectly in the static lambda serialization mode, this issue can be addressed by configuring the server to the dynamic lambda serialization mode.

Considerations for a Rolling Upgrade

The failures described in Configuring Lambda Serialization Mode can occur when performing a rolling upgrade in the static lambda serialization mode. Both the application code and Coherence versions have to be the same in the Java clients and servers in the distributed environment for the synthetic lambda method names to be stable across the Java class files. The Java application code containing the lambda expressions sent from one JVM to another in a Coherence distributed environment must be on all JVM classpath/modulepath. Oracle recommends you to perform a rolling upgrade with the dynamic lambda serialization mode enabled if you observe any of these failures while testing the rolling upgrade.

Processing Entries in Multiple Caches

Entry processors can update cache entries in multiple caches within a single `process` or `processAll` operation. The caches must be managed by the same service and the entries must be located in the same partition. See Specifying Data Affinity.

The `process` and `processAll` operations are performed in a transaction-like manner that uses implicit locks when accessing, inserting, updating, modifying, or removing cache entries. If an exception is thrown during the processing of the entries, the entries are rolled back leaving all underlying values unchanged. The `processAll` operation is atomic with respect to all entries within a single partition (or member if no service threads are configured) rather than the individual entry or entire request.

> **Note:**
>
> The implicit lock may create a deadlock if entries are locked in conflicting orders on different threads. The application is responsible for ensuring that cache entries are accessed (locked) in a deadlock-free manner. In the case where a deadlock is detected, an exception is thrown but the underlying service is not stopped.

The `com.tangosol.net.BackingMapContext` API is used to process entries in multiple caches and provides a way to directly access entries in a cache's backing map. The backing map is the actual `Map` implementation where entries are stored (as opposed to the logical representation of a cache typically used by an application). Entries in a backing map are stored in binary format and therefore require an application to handle the serialized form of an entry. See Implementing Storage and Backing Maps.

The `com.tangosol.util.BinaryEntry` API provides easy access to a backing map context and is typically used by an application. The following sample code demonstrates how to update entries in two different caches within the `process` method of an entry processor using the `BinaryEntry` API.

```
public Object process(Entry entry) {
   BinaryEntry binEntry = (BinaryEntry) entry;
```

```
   Binary binKey = binEntry.getBinaryKey();
   Trade trade = (Trade) binEntry.getValue();

   // Update a Trade object in cache1

   trade.setPrice(trade.getPrice() + factor);
   binEntry.setValue(trade);

   // update a Trade object in cache2

   BackingMapManagerContext ctx = binEntry.getContext();
   BinaryEntry binEntry2 =
      (BinaryEntry) ctx.getBackingMapContext("cache2").getBackingMapEntry(binKey);
   Trade trade2 = (Trade) binEntry2.getValue();
   trade2.setPrice(trade2.getPrice() + factor);
   binEntry2.setValue(trade2);

   return null;
}
```

> **Note:**
>
> The `getBackingMapEntry` method may only be called within the context of an entry processor invocation. Any changes made to the entry are persisted with the same lifecycle as those made by the enclosing invocation. The returned entry is only valid for the duration of the enclosing invocation and multiple calls to this method within the same invocation context returns the same entry object.

Ignoring the Results of an Entry Processor

The `processAll` method of the `AbstractProcessor` class returns a map of results to a client application. The map contains the keys and values for every entry that was processed. Most often, the entry processor returns results that the client uses. However, there may be situations where some results are not usable by the client. More importantly, there may be situations where the processor must evaluate all the entries in a cache; in which case, the return map contains every key in the cache. In both situations, the agent should be designed to ignore results that are not wanted.

Designing an agent that only returns wanted results is a good pattern and best practice, because it:

- Makes the client's memory footprint independent of the size of the cache.
- Avoids transferring all affected keys to the client which could result in an `OutOfMemoryError` exception on the client if the cache is too large.
- Avoids deserialization costs in the client for all keys.
- Avoids transferring the map and all keys through a proxy node (for Extend clients).

To ignore entry processor results, override the processor's `processAll` method to return an empty `Map` or `null`. The following example demonstrates a simple entry processor agent that always returns `null` after processing an entry. The example is not very realistic but does show how to override the `processAll` method.

```
public static class Agent
   implements InvocableMap.EntryProcessor
   {
```

```
private static final long serialVersionUID = 1L;

@Override
public Object process(Entry entry)
   {
   return null;
   }

@Override
public Map processAll(Set set)
   {
   for (Entry entry : (Set<Entry>) set)
      {
      process(entry);
      }
   return null;
   }
```

Performing Synthetic Operations

Entry processors can perform synthetic operations on entries. Both the `remove` and `setValue` methods for an entry can be declared synthetic by including a `true` parameter in the method call. For example:

```
entry.setValue(value, true)
```

The `setValue` method in a synthetic operation does not return the previous value. In addition, synthetic operations are not propagated to cache stores or binary entry stores; applications are able to commit changes after all processing is complete.

Applications typically use synthetic operations to prepare a cache and perform operations (state changes) that do not require listeners and cache stores to be notified, especially when doing so may be expensive in terms of network, memory, and CPU usage. Applications also use synthetic operations when pre-warming a cache; applications that load cache data may want to avoid having a cache store called, especially if the entries that are being loaded into the cache are coming from the back-end data source.

Applications can differentiate between client driven (natural) events and cache internal (synthetic) events using the `isSynthetic` method on the `CacheEvent` class. See Using Synthetic Events.

Processing Entries Asynchronously

Entry processors can be invoked asynchronously using the `AsynchronousProcessor` class. The class implements the standard Java `Future` interface and also includes a Coherence-specific flow control mechanism to guard against excessive backlogs.

> **Note:**
>
> - Entries can also be processed asynchronously using the `AsyncNameCache<K, V>` interface. See Performing NamedMap Operations Asynchronously.
> - This feature is not available on Coherence*Extend clients.

The `AsynchronousProcessor` class is used to wrap an entry processor implementation. For example:

```
UpdaterProcessor up = new UpdaterProcessor(null, value);
AsynchronousProcessor ap = new AsynchronousProcessor(up);

cache.invokeAll(filter, ap);
ap.getResult();
```

The above example invokes the underlying entry processor and uses automatic flow control (as defined by the underlying service's flow control logic) and a default unit-of-order ID (assigned to the calling thread's `hashCode` – as a thread's requests are naturally expected to execute in order). Ordering is guaranteed for each partition even during failover. Additional constructors are available to manually control request flow and assign the unit-of-order ID as required.

For advanced use cases, the `AsynchronousProcessor` class can be extended to define custom asynchronous functionality. The following example extends the `AsynchrounousProcessor` class and overrides the `onResult`, `onComplete` and `onException` methods.

> **Note:**
>
> Overriding implementations of the `onComplete`, `onResult`, and `onException` methods must be non-blocking and short-lived, because this call is made on the service thread of the client and blocks processing for responses on other threads.

```
AsynchronousProcessor processor = new AsynchronousProcessor(null)
   {

   public synchronized void onResult(Entry entry)
      {
      super.onResult(entry);

      // process the result
      }

   public void onComplete()
      {
      super.onComplete();

      if (m_eReason == null)
         {

         // process the result
         }
      else
         {

         // process the (potentially partial) failure
         }
      }

   public void onException(Throwable eReason)
      {
         super.onException(eReason);

         // process the observed exception
```

```
        }
};
```

Performing Data Grid Aggregation

The `InvocableMap` interface 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. For details, see the `aggregate` method.
In addition, the `EntryAggregator` interface can be used to processes a set of `InvocableMap.Entry` objects to achieve an aggregated result.

For efficient execution in a data grid, an aggregation process must be designed to operate in a parallel manner. The `StreamingAggregator` interface is an advanced extension to the `EntryAggregator` interface that is explicitly capable of being run in parallel in a distributed environment.

> **Note:**
>
> The `ParallelAwareAggregator` interface has been deprecated and should no longer be used. Applications should use the `StreamingAggregator` interface to implement custom aggregators. See Performing Data Grid Aggregation Using Streams.

Coherence includes many natural aggregator functions. The functions include:

- `Count`
- `DistinctValues`
- `DoubleAverage`
- `DoubleMax`
- `DoubleMin`
- `DoubleSum`
- `LongMax`
- `LongMin`
- `LongSum`

See the `com.tangosol.util.aggregator` package for a list of Coherence aggregators. To implement your own aggregator, see the `StreamingAggregator` interface.

Performing Data Grid Aggregation Using Streams

Data grid aggregation can be performed using Java streams. The use of streams provides a simplified programming model especially when combined with Java lambda expressions. For example:

```
ValueExtractor<Person, Integer> ageExtractor = Person::getAge;
double avgAge = cache.stream()
 .mapToInt(entry -> entry.extract(ageExtractor))
 .average()
 .getAsDouble();
```

When using Coherence filters, pass a filter object as the source of the stream. For example:

```
ValueExtractor<Person, Integer> ageExtractor = Person::getAge;
int max = personCache.stream(filter)
 .mapToInt(entry -> entry.extract(ageExtractor))
 .max();
```

As an alternative, use the Coherence `Stream` API extension to specify the extractor when creating a stream and rely on the extension for any optimizations. For example:

```
int max = personCache.stream(filter, Person::getAge)
 .mapToInt(Number::intValue)
 .max();
```

Note that in this case you must use `mapToInt(Number::intValue)` to convert `Stream<Integer>` into `IntStream`. This conversion can also be done internally. For example:

```
int max = RemoteStream.toIntStream(personCache.stream(filter, Person::getAge)).max();
```

The Java streams implementation has been extended in Coherence to allow aggregation across the cluster. The API defines a set of aggregators that support streams by implementing the `InvocableMap.StreamingAggregator` interface. In addition, Coherence includes many useful collectors that can be executed in parallel in a distributed environment. The collectors are called using the `RemoteCollectors` class. For example:

```
avgAge = cache.stream()
   .map(Map.Entry::getValue)
   .collect(RemoteCollectors.averagingInt(Contact::getAge));
System.out.println("\nThe average age of all contacts using collect() is: " +
   avgAge);
```

To define custom aggregators that also support streams, you can extend the `CollectorAggregator` class.

Performing Node-Based Processing

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 a particular node in the grid, in parallel on a 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. See `invocation-scheme`. 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:

```
/**
* 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:

```
/**
* 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 only requires a single 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:

```
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.

```
/**
* 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;
}
```

31
Using Map Events

You can use map event listeners to receive cache events and events from any class in Coherence that implements the `ObservableMap` interface.
This chapter includes the following sections:

Overview of Map Events

Coherence provides cache events using the JavaBean Event model.

The implementation allows applications to receive the events when and where they are needed, 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.

> **Note:**
>
> Coherence also includes the live event programming model. Live events provide support for common event types and can be used instead of map events. See Using Live Events.

This section includes the following topics:

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.

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.

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`

Signing Up for All Events

To sign up for events, pass an object that implements the `MapListener` interface to an `addMapListener` method on the `ObservableMap` interface.The following example illustrates a sample `MapListener` implementation that prints each event it receives.

```
/**
* 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, you can print all events from any given cache (since all caches implement the `ObservableMap` interface):

```
cache.addMapListener(new EventPrinter());
```

To be able to later remove the listener, it is necessary to hold on to a reference to the listener:

```
Listener listener = new EventPrinter();
cache.addMapListener(listener);
m_listener = listener; // store the listener in a field
```

The listener can then be removed:

```
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

You can use the `AbstractMapListener` base class 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).The following example, prints out only the insert events for the cache.

```
cache.addMapListener(new AbstractMapListener()
    {
    public void entryInserted(MapEvent evt)
        {
        out(evt);
        }
    });
```

Another helpful base class for creating a `MapListener` implementation is the `MultiplexingMapListener`, which routes all events to a single method for handling. 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`:

```
public static class EventPrinter
        extends MultiplexingMapListener
    {
    public void onMapEvent(MapEvent evt)
        {
        out(evt);
        }
    }
```

Using Lambda Expressions to Add Map Listeners

A lambda expression can be used to add `MapListener<K, V>` implementations.The following example uses a lambda expression to add the `SimpleMapListener<K, V>` implementation that is delivered with Coherence. The implementation delegates to an appropriate event handler based on the event type.

```
MapListener<ContactId, Contact> listener = new SimpleMapListener<ContactId,
   Contact>().addInsertHandler((event) -> System.out.println("\ninserted:\n" +
   event.getNewValue()));

cache.addMapListener(listener);
```

Configuring a MapListener For a Cache

If a listener should always be on a particular cache, then place it into the cache configuration using the `<listener>` element and the listener is automatically added to the cache.See `listener`.

Signing Up For Events On Specific Identities

You can sign up for events that occur against specific identities (keys).For example, to print all events that occur against the Integer key `5`:

```
cache.addMapListener(new EventPrinter(), new Integer(5), false);
```

Thus, the following code would only trigger an event when the Integer key `5` 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

You can use filters to listen to particular events.The following example adds a listener to a cache with a filter that allows the listener to only receive delete 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 Listening to Queries.

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}
```

Using Lite Events

You can use lite events if an event should only include new values. By default, Coherence provides both the old and the new value as part of an event. Consider the following example:

```
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 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:

```
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:

```
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.

Listening to Queries

The same filters that are used to query a cache can listen to events from a cache. This section includes the following topics:

- Overview of Listening to Queries
- Filtering Events Versus Filtering Cached Data

Overview of 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`).

For example, in a trading system it is possible to query for all open `Order` objects for a particular trader:

```
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:

```
// 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. For example:

```
// 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 for new trades assigned to this trader
nMask = MapEventFilter.E_INSERTED;
mapTrades.addMapListener(listener, new MapEventFilter(nMask, filter), true);
```

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.

Using Synthetic Events

You can choose to monitor 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:

```
public static class EventPrinter
        extends MultiplexingMapListener
    {
    public void onMapEvent(MapEvent evt)
        {
        if (evt instanceof CacheEvent && ((CacheEvent) evt).isSynthetic())
            {
            out(evt);
            )
        }
    }
```

> **Note:**
>
> Not all cache service types support the dispatching of synthetic events. Synthetic events will only be dispatched by a partitioned cache service and its derivatives, such as a federated cache service, or by near, view, or remote caches that are backed by a cache service that supports the dispatching of synthetic events. In all other cases, no event will be dispatched for synthetic events such as expiry.

Listening to Backing Map Events

You can listen to events for the map that backs a cache (partitioned, replicated, near, continuously-query, read-through/write-through and write-behind).
This section includes the following topics:

- Overview of Listening to Backing Map Events
- Producing Readable Backing MapListener Events from Distributed Caches

Overview of Listening to Backing Map Events

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 itself. Often times for internal backing map listeners, events are not processed immediately, but are queued and processed later asynchronously.

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 https://coherence.java.net/ to download the 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:

```
-Dcoherence.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 overridden to send the results to standard output.

```
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);
        }
}
```

The following example demonstrates setting the `<listener>` element in a distributed cache scheme and identifies the `VerboseBackingMapListener` implementation as being of type `com.tangosol.net.BackingMapManagerContext`.

```
<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>
```

Using 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 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. That is, on a put operation into a near cache, the local copy and the distributed primary and backup copies of the entry are updated as a synchronous operation. Once this update is complete, asynchronous events are sent to all the other listening near caches. This invalidates the local copies so that the entries are retrieved from the back cache on the next get operation.

Using Durable Events (Experimental)

Coherence has provided the ability for clients to asynchronously observe data changes for almost two decades. This has proven to be an incredibly powerful feature allowing Coherence to offer components such as NearCaches and ViewCaches/CQCs, in addition to allowing customers to build truly event-driven systems. See Understanding Near Caches and Understanding View Caches.

For a comprehensive overview of MapEvents in terms of the call back interface, in addition to the registration mechanisms, see Using Map Events. However, it is worth drawing attention to

some of the guarantees offered by MapEvents to provide context of why the Durable Events feature is useful.

- Guaranteeing a MapEvent
- Replaying Generic Events
- Availability in Production

Guaranteeing a MapEvent

Coherence guarantees that a `MapEvent`, which represents a change to an entry in the source map, will be delivered exactly once, given the client remains a member of the associated service.

> **Note:**
>
> Each `NamedMap` is associated with a `Service` and typically this service is a `PartitionedService`. Therefore, it provides distributed storage and partitioned/sharded access. The remaining description of `MapEvent` guarantees will assume the use of a `PartitionedService` and therefore providing resilience to process, machine, rack, or site failure.

Importantly, this guarantee is maintained regardless of failures in the system that the services are configured to handle. Therefore, a `backup-count` of '1' results in the service being able to tolerate the loss of a single unit where unit can be a node (JVM/member), machine, rack, or site. Upon encountering a fault, Coherence restores data and continues service for the affected partitions. This data redundancy is extended to MapEvent delivery to clients. Therefore, if a member hosting primary partitions was to die, and the said member had sent the backup message for some change but failed to deliver the MapEvent to the client, the new primary member (that was a backup member and went through the automatic promotion protocol) would emit MapEvent messages that had not been confirmed by clients. The client is aware of MapEvent messages it had already processed, and therefore if the MapEvent was received by the client but the primary did not receive the ACK, causing the backup to send a duplicate, the client will not replay the event.

The aforementioned guarantee of MapEvent delivery are all valid under the assumption that the member that has registered for MapEvents, does **not** leave the service. If the member leaves the associated service, then these guarantees no longer apply. This can be problematic under a few contexts (described in the sections below) and has led to the need deliver a more general feature of event replay

This section includes the following topics:

- Abnormal Service Termination
- Extending Proxy Failover
- Abnormal Process Termination

Abnormal Service Termination

While it is rare, there are some scenarios that result in abnormal service termination. The abnormal termination causes references to `Services` or `NamedMaps` to become invalid, and therefore unusable. Instead of having numerous applications and internal call sites be defensive to these invalid handles, Coherence chose to introduce a 'safe' layer between the

call site and the raw/internal service. The 'safe' layer remains valid when services abnormally terminate, and additionally may cause the underlying service to restart.

In the case of a 'safe' `NamedMap`, any `MapEvent` listeners registered will automatically be re-registered. However, any events that had occurred after the member left the service and before it re-joined will be lost, or more formally, not observed by this member's listener. This has been worked around in many cases by these members/clients re-synchronizing their local state with the remote data; this is the approach taken for `ContinuousQueryCaches`.

Extending Proxy Failover

Coherence Extend provides a means for a client to connect to a cluster through a conduit referred to as a proxy. An extend client wraps up the intended request and forwards to a proxy which executes the said request with the results either streamed back to the client or sent as a single response. There are many reasons for using 'extend' over being a member of the cluster. For more information about extend, see Introduction to Coherence*Extend in *Developing Remote Clients for Oracle Coherence*.

The liveness of the proxy is important to the extend clients that are connected to it. If the proxy becomes unresponsive or dies, the extend client transparently reconnects to another proxy. Similar to Abnormal Service Termination, there is a potential of not observing MapEvents that had occurred at the source due to the proxy leaving the associated service or the extend client reconnecting to a different proxy, and therefore re-registering the listener.

There are means to work around this situation by observing the proxy disconnect/re-connect and causing a re-synchronization of extend client and the source. However, extend proxy failover is a significantly more likely event to occur.

Abnormal Process Termination

A logical client receiving MapEvents may experience a process restart and it may be desirable to continue receiving MapEvents after the last received event, instead of receiving events only after registration. For example, a client may be responsible for updating some auxiliary system by applying the contents of each MapEvent to that system. Additionally, the client may be tracking the last received events, and therefore can inform the source of the last event it received. Thus, a capable source can replay all the events that were missed.

Replaying Generic Events

In considering how to address the various scenarios that result in event loss, it became evident that the product needs an ability to retain events beyond receipt of delivery such that they can be replayed on request. This pushes certain requirements on different parts of the system:

- Storage Nodes
  - Track by storing the `MapEvents` as they are generated.
  - Expose a monotonically increasing version per partition within a single cache.
  - Ensure that version semantics are maintained regardless of faults.
  - Ensure that `MapEvent` storage is redundant.
  - Provide a `MapEvent` storage retention policy.
- Client Nodes
  - Have a trivial facility to suggest received `MapEvent` versions are tracked and therefore events replayed when faced with a restart.

– Provide a more advanced means such that the client can control the tracking of event versions.

With the above features, a `MapListener` can opt in by suggesting they are version aware and Coherence will automatically start tracking versions on the client. This does require a complementing server-side configuration in which the storage servers are tracking `MapEvents`. For example:

1. Start a storage server that is tracking events:

```
$ java -Dcoherence.distributed.persistence.mode=active -
Dcoherence.distributed.persistence.events.dir=/tmp/events-dir -jar
coherence.jar
```

2. Start a client that registers a version aware `MapEvent` listener. A snippet taken from a functional test in the repo is provided below:

```
List<MapEvent> listEvents = Collections.synchronizedList(new
ArrayList<>());

MapListener<Integer, String> listener = new SimpleMapListener<Integer,
String>()
        .addEventHandler(listEvents::add)
        .versioned();

cache.addMapListener(listener, 1, false);
```

   The above registration results in the client receiving events and tracking the latest version per partition. Upon abnormal service restart, the client will automatically re-register itself and request versions after the last received version.

More advanced clients can implement the VersionAwareMapListener interface (see Interface VersionAwareMapListener) directly. Implementors must return an implementation of VersionedPartitions (see Interface VersionedPartitions) which will be interrogated by Coherence upon registration, either directly due to a call to `NamedMap.addMapListener` or indirectly due to a service restart. These versions are sent to the relevant storage servers and if a version is returned for a partition, Coherence will return all known versions later than or equal to the specified version. Additionally, certain formal constants are defined to allow a client to request storage servers to send:

- all known events
- current head and all future events (previously known as priming events)
- all future events (the current behavior)

> **Note:**
>
> There is a natural harmony between the registration mechanism and the partitions returned from `VersionedPartitions` that occurs and is worth noting. For example, when registering a `MapListener` against a specific key, only `MapEvents` for the said key will be received by this `MapListener`, and therefore only versions for the associated partition will be tracked. The `VersionedPartitions` returned by this `VersionAwareMapListener` will only return a version for a single partition. However, this is worth noting if you implement your own `VersionAwareMapListener` or `VersionedPartitions` data structure.

Availability in Production

The Durable Events feature is not production ready and Oracle does not recommend using it in production at this point. There are some features that are required prior to this feature graduating to a production ready status. However, Oracle is making this feature available in its current form to garner feedback prior to locking down the APIs and semantics.

The following improvements will be required prior to this feature being production ready:

- Redundant MapEvent storage
- MapEvent retention policy
- MapEvent delivery flow control
- Extend and gRPC client support
- Snapshots of MapEvent storage
- Allow a logical client to store its `VersionedPartitions` in Coherence
- Monitoring metrics

To get started with using this feature in its current form, see Durable Events.

32
Controlling Map Operations with Triggers

You can use map triggers to validate, reject, or modify map operations before a change is committed to a map entry.
This chapter contains the following sections:

Overview of Map Triggers

Map triggers supplement the standard capabilities of Coherence to provide a highly customized cache management system.Map triggers can be used to prevent invalid transactions, enforce complex security authorizations or complex business rules, provide transparent event logging and auditing, and gather statistics on data modifications. Other possible use for triggers include restricting operations against a cache to those issued during application re-deployment time.
For example, assume that you have code that is working with a `NamedCache`, and you want to change an entry's behavior or contents before the entry is inserted into the map. The addition of a map trigger 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 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 runtime exception
- 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. The following example assumes that the `createMapTrigger` method would return a new `MapTriggerListener(new MyCustomTrigger());`:

```
<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, 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`. The following example registers the `PersonMapTrigger` with the `People` 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.

A Map Trigger Example

Learn how to use map triggers by following a basic example.The code in Example 32-1 illustrates a map trigger and how it can be called.
In the `PersonMapTrigger` class, 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 32-1 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 32-2, calls the `PersonMapTrigger`. The new `MapTriggerListener` passes the `PersonMapTrigger` to the `People NamedCache`.

Example 32-2 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");
        }
    }
```

33
Using Live Events

You can be notified of events using event interceptors.Applications use live events to react to cluster operations with application logic.
This chapter includes the following sections:

Overview of Live Events

Coherence provides an event programming model that allows extensibility within a cluster when performing operations against a data grid. The model uses events to represent observable occurrences of cluster operations. The events that are currently supported include:

- Partitioned Cache Events – A set of events that represent the operations being performed against a set of entries in a cache. Partitioned cache events include both entry events and entry processor events. Entry events are related to inserting, removing, and updating entries in a cache. Entry processor events are related to the execution of entry processors.
- Partitioned Cache Lifecycle Events – A set of events that represent the operations for creating a cache, destroying a cache, and clearing all entries from a cache.
- Partitioned Service Events – A set of events that represent the operations being performed by a partitioned service. Partitioned service events include both partition transfer events and partition transaction events. Partition transfer events are related to the movement of partitions among cluster members. Partition transaction events are related to changes that may span multiple caches and are performed within the context of a single request.
- Lifecycle Events – A set of events that represent the activation and disposal of a `ConfigurableCacheFactory` instance.
- Federation Events – A set of events that represent the operations being performed by a federation service. Federation events include both federated connection events and federated change events. Federated connection events are related to the interaction of federated participants and federated change events are related to cache updates.

Applications create and register event interceptors to consume events. Event interceptors handle the events and implement any custom logic as required. Events have different rules which govern whether or not mutable actions may be performed upon receiving the event.

Understanding Live Event Types

Event types represent observable occurrences of cluster operations.Applications handle the events using event interceptors and decide what action to take based on the event type. This section describes each of the supported event types and is organized according to the functional areas in which the events are raised.
This section includes the following topics:

Understanding Partitioned Cache Events

Partitioned cache events represent operations that are performed against a set of entries in a cache. Partitioned cache events include entry events (`EntryEvent`) and entry processor events (`EntryProcessorEvent`). These events are defined within the `com.tangosol.net.events.partition.cache` package.

This section includes the following topics:

Entry Events

Entry events represent operations for inserting, removing, and updating entries in a cache. Precommit entry events (`INSERTING`, `REMOVING`, and `UPDATING`) are raised before the operation is performed to allow modification to an entry. The following holds true when modifying entries:

- Event interceptors that are registered for precommit entry events are synchronously called.
- A lock is held for each entry during the processing of the event to prevent concurrent updates.
- Throwing an exception prevents the operation from being committed.

Postcommit entry events (`INSERTED`, `REMOVED`, and `UPDATED`) are raised after an operation is performed and in the same order as the events occurred. Postcommit events indicate that an entry is no longer able to be modified. Event interceptors for postcommit entry events are asynchronously processed.

> **Note:**
>
> Event interceptors for postcommit entry events are processed on the cache service's `EventDispatcher` thread. For application liveness and performance, it is important to not overload this thread by performing expensive operations in `EventInterceptors` for postcommit entry events, such as making blocking calls to external systems or back into Coherence services through the `NamedCache` API. Instead, such operations should be run on application threads (for example, `java.util.concurrent.ExecutorService` threads to which Runnables are submitted by the `EventInterceptor`).

Table 33-1 lists the entry event types.

Table 33-1 Entry Events

Event Types	Description
INSERTING	Indicates that an entry (or multiple entries) is going to be inserted in the cache.
INSERTED	Indicates that an entry (or multiple entries) has been inserted in the cache.
REMOVING	Indicates that an entry (or multiple entries) is going to be removed from the cache.
REMOVED	Indicates that an entry (or multiple entries) has been removed from the cache.
UPDATING	Indicates that an entry (or multiple entries) is going to be updated in the cache.
UPDATED	Indicates that an entry (or multiple entries) has been updated in the cache.

Entry Processor Events

Entry processor events represent the execution of an entry processor on a set of binary entries. Precommit entry processor events (EXECUTING) are raised before an entry processor is executed to allow modification to the entry processor instance. The following holds true when modifying an entry processor:

- Event interceptors that are registered for precommit entry processor events can modify the entries being processed and expect the modifications to be durable.
- Event interceptors that are registered for precommit entry processor events are synchronously called.
  Entry processors can be shared across threads; therefore, ensure thread safety when modifying in an entry processor.
- A lock is held for each entry during the processing of the event to prevent concurrent updates.
- Throwing an exception prevents the entry processor from being executed.

Postcommit entry processor events (EXECUTED) are raised after an entry processor is executed and in the same order that the events occurred. Postcommit events indicate that an entry is no longer able to be modified. Event interceptors for postcommit entry processor events are asynchronously processed.

Table 33-2 lists the entry processor event types.

Table 33-2 Entry Processor Events

Event Types	Description
EXECUTING	Indicates that an entry processor is going to be executed on a set of entries.
EXECUTED	Indicates that an entry processor has been executed on a set of entries.

Understanding Partitioned Cache Lifecycle Events

Partitioned cache lifecycle events (`CacheLifecycleEvent`) represent the creation, destruction, and truncation of a partitioned cache. These events are defined within the `com.tangosol.net.events.partition.cache` package.

Partitioned cache lifecycle events are raised after a partitioned cache has been created, destroyed (`NamedCache.destory`), or truncated (`NamedCache.truncate`). A `TRUNCATED` event does not include the entries that are removed from a cache.

Table 33-3 Partitioned Cache Lifecycle Events

Event Types	Description
`CREATED`	Indicates that storage for a given cache has been created.
`DESTROYED`	Indicates that storage for a given cache has been destroyed.
`TRUNCATED`	Indicates that all mappings for a given cache have been removed from storage.

Understanding Partitioned Service Events

Partitioned service events represent operations being performed by a partitioned service. Partitioned service events include transfer events (`TransferEvent`), transaction events (`TransactionEvent`) and unsolicited commit events (`UnsolicitedCommitEvent`). These events are defined within the `com.tangosol.net.events.partition` package.

This section includes the following topics:

- Transfer Events
- Transaction Events
- Unsolicited Commit Events

Transfer Events

Partitioned service transfer events represent partition transfers between storage enabled members. The event includes the service name for which the transfer is taking place, the partition ID, the cluster members involved in the transfer and a map of cache names to entries. The entries cannot be modified.

> **Note:**
>
> Transfer events are raised while holding a lock on the partition being transferred that blocks any operations for the partition.

Table 33-4 lists the transition event types.

Table 33-4 Transfer Events

Event Types	Description
DEPARTING	Indicates that a set of entries are being transferred from the current member.
DEPARTED	Indicates that a set of entries has been successfully transferred from the current member.
ARRIVED	Indicates that a set of entries has been transferred to or restored by the current member.
ASSIGNED	Indicates that a set of entries has been assigned to a member. This event may be emitted only at the ownership senior during the initial partition assignment.
LOST	Indicates that a partition is orphaned (data loss may have occurred). This event may be emitted only at the ownership senior.
RECOVERED	Indicates that a set of entries has been successfully recovered from persistent storage.
ROLLBACK	Indicates that a partition transfer has failed and therefore rolled back.

Transaction Events

Partitioned service transaction events represent changes to binary entries (possibly from multiple caches) that are made in the context of a single service request. Precommit transaction events (COMMITTING) are raised before any operations are performed to allow modification to the entries. The following holds true when modifying entries:

- A lock is held for each entry during the processing of the event to prevent concurrent updates.
- Throwing an exception prevents the operation from being committed.

Postcommit transaction events (COMMITTED) are raised after an operation is performed. Postcommit events indicate that the entries are no longer able to be modified.

Table 33-5 lists the transaction event types.

Table 33-5 Transaction Events

Event Types	Description
COMMITTING	Indicates that entries are going to be inserted in their respective cache.
COMMITTED	Indicates that entries have been inserted in their respective cache.

Unsolicited Commit Events

Partitioned service unsolicited commit events represent changes pertaining to all observed mutations performed against caches that were not directly caused (solicited) by the partitioned service. These events may be due to changes made internally by the backing map, such as eviction, or references to the backing map that caused changes. The event contains all modified entries which may span multiple backing maps. The entries cannot be modified.

Table 33-6 lists the unsolicited commit event types.

Table 33-6 Unsolicited Commit Events

Event Types	Description
COMMITTED	Indicates that entries have been inserted in their respective cache.

Coherence Lifecycle Listeners

When using the Bootstrap API to start Coherence, listeners can be added to receive events as configured `Coherence` instances are started and stopped.

Coherence lifecycle events (`CoherenceLifecycleEvent`) represent actions that occur for a `Coherence` instance. These events are defined in the `com.tangosol.net.events` package.

Table 33-7 Lifecycle Events

Event Types	Description
STARTING	This event is raised just before the `Coherence` instance starts. At this point `Coherence` instance associated with the event is not running and no services will be available.
STARTED	This event is raised after the `Coherence` instance has started. At this point all services and `Session` instances owned by the `Coherence` instance associated with the event will be available.
STOPPING	This event is raised just before the `Coherence` instance is stopped. At this point all services and `Session` instances owned by the `Coherence` instance associated with the event will still be available.
STOPPED	This event is raised just before the `Coherence` instance starts. At this point the `Coherence` instance associated with the event is not running and no services will be available.

Apart from adding Coherence lifecycle event interceptors via the normal live events interceptor pattern, or via the Bootstrap API configuration, instances of the `com.tangosol.net.Coherence.LifecycleListener` are discovered using the Java `ServiceLoader` mechanism. This allows Coherence lifecycle listeners to be added by just including the jar containing those listeners on the class path. Instances of `LifecycleListener` suitably configured under `META-INF/services` or in a `module-info.java` file are discovered on the class path when a `Coherence` instance starts.

Session Lifecycle Listeners

When using the Bootstrap API to start Coherence, listeners can be added to receive events as configured `Session` instances are started and stopped.

Session lifecycle events (`SessionLifecycleEvent`) represent actions that occur for a `Session` instance. These events are defined in the `com.tangosol.net.events` package.

Table 33-8 Lifecycle Events

Event Types	Description
`STARTING`	This event is raised just before the `Session` instance starts. At this point Session instance associated with the event is not running and no services will be available.
`STARTED`	This event is raised after the `Session` instance has started. At this point all services owned by the `Session` associated with the event will be available.
`STOPPING`	This event is raised just before the `Session` instance is stopped. At this point all services owned by the `Session` instance associated with the event will still be available.
`STOPPED`	This event is raised just before the `Session` instance starts. At this point the `Session` instance associated with the event is not running and no services will be available.

- ConfigurableCacheFactory Lifecycle Events
- DefaultCacheServer Lifecycle Events

ConfigurableCacheFactory Lifecycle Events

Lifecycle events (`LifecycleEvent`) represent actions that occur on a `ConfigurableCacheFactory` instance. These events are defined within the `com.tangosol.net.events.application` package.

An `ACTIVATED` event is raised after all services that are associated with a cache factory are started. The services are defined in the cache configuration file and must be configured to autostart. A `DISPOSING` event is raised before all services are shut down and any resources are reclaimed. Event interceptors that handle a `DISPOSING` event are notified before the services are shutdown. A `ConfigurableCacheFactory` instance can only be activated and disposed of once.

> **Note:**
>
> Lifecycle events are dispatched to event interceptors by the same thread calling the lifecycle methods on the `ConfigurableCacheFactory` implementation. This thread may be synchronized. Event interceptors must ensure that any spawned threads do not synchronize on the same `ConfigurableCacheFactory` object.

The following table lists the lifecycle event types.

Table 33-9 Lifecycle Events

Event Types	Description
`ACTIVATED`	Indicates that a `ConfigurableCacheFactory` instance is active.

Table 33-9 (Cont.) Lifecycle Events

Event Types	Description
DISPOSING	Indicates that a ConfigurableCacheFactory instance is going to be disposed.

DefaultCacheServer Lifecycle Events

When using the legacy DefaultCacheServer to start and stop Coherence, it is possible to register a com.tangosol.application.LifecycleListener with a DefaultCacheServer to receive lifecycle events. Listeners can be added in two ways, programmatically or by discovery.

The DefaultCacheServer class has an addLifecycleListener method and a removeLifecycleListener method that allow listeners to be added programmatically to an instance of a DefaultCacheServer.

Instances of com.tangosol.application.LifecycleListener are also discovered at runtime using Java's ServiceLoader mechanism. Instances of LifecycleListener suitably configured under META-INF/services or in a module-info.java file are discovered on the class path when a DefaultCacheServer starts.

Understanding Federation Events

Federation events represent a set of events that represent the operations being performed by a federation service. Federation events include federated connection events (FederatedConnectionEvent), federated change events (FederatedChangeEvent), and federated partition events (FederatedPartitionEvent) . These events are defined within the com.tangosol.net.events.federation package.

This section includes the following topics:

- Federated Connection Events
- Federated Change Events
- Federated Partition Events

Federated Connection Events

Federated connection events represent the communication between participants of a federated service.

> **Note:**
>
> Federated connection events are raised on the same thread that caused the event. Interceptors that handle these events must never perform blocking operations.

Table 33-10 lists the federated connection event types.

Table 33-10 Federated Connection Events

Event Types	Description
`CONNECTING`	Indicates that a connection is about to be initiated to a participant.
`DISCONNECTED`	Indicates that participant is no longer connected.
`BACKLOG_EXCESSIVE`	Indicates that a participant is backlogged. If the participant is remote, then it indicates the remote participant has more work than it can handle. If the participant is local, then it indicates this participant has more work than it can handle.
`BACKLOG_NORMAL`	Indicates that a participant is no longer backlogged.
`ERROR`	Indicates that an error occurred while applying changes on the remote participant or that the maximum connection retry has been reached.

Federated Change Events

Federated change events represent a transactional view of the changes that occur on the local cluster participant in a federation. The transaction is for a partition; that is, all changes that belong to a single partition are captured in a single `FederatedChangeEvent` object.

> **Note:**
>
> Synthetic operations are not included in federation change events.

Federated change events allow conflict resolution by allowing or disallowing changes to cache entries before they are committed. See Processing Federated Change Events in *Administering Oracle Coherence*.

The following holds true when modifying entries:

- A lock is held for each entry during the processing of the event to prevent concurrent updates.
- Throwing an exception prevents the operation from being committed.

Table 33-11 lists the federated change event types.

Table 33-11 Federated Change Events

Event Types	Description
`COMMITTING_LOCAL`	Indicates that entries are going to be inserted in the cache on the local participant.
`COMMITTING_REMOTE`	Indicates that entries from other participants are going to be inserted in the cache on the local participant.
`REPLICATING`	Indicates that entries are going to be replicated to remote participants.

Federated Partition Events

Federated partition events represent a change in the federation state of a partition during federation.

> **Note:**
>
> This event is dispatched for each partition.

Table 33-12 lists the federated partition event types.

Table 33-12 Federated Partition Events

Event Types	Description
`SYNCING`	Indicates that a partition is being federated to a destination as a result of a `replicateAll` operation.
`SYNCED`	Indicates that partition federation to a destination is complete as a result of a `replicateAll` operation.

Handling Live Events

Applications handle live events using event interceptors.The interceptors explicitly define which events to receive and what action, if any, to take. Any number of event interceptors can be created and registered for a specific cache or for all caches managed by a specific partitioned service. Multiple interceptors that are registered for the same event type are automatically chained together and executed in the context of a single event.
This section includes the following topics:

- Creating Event Interceptors
- Understanding Event Threading
- Registering Event Interceptors
- Chaining Event Interceptors

Creating Event Interceptors

Event interceptors are created by implementing the `EventInterceptor` interface. The interface is defined using generics and allows you to subscribe to events by specifying the generic type of the event as a type parameter. The inherited `onEvent` method provides the ability to perform any necessary processing upon receiving an event. The following example demonstrates subscribing to all transfer events and is taken from Example 33-1:

```
public class RedistributionInterceptor
   implements EventInterceptor<TransferEvent>

   public void onEvent(TransferEvent event)
   {
      ...
```

The `@Interceptor` annotation can be used to further restrict the events to specific event types and also provides further configuration of the interceptor. The following example defines an interceptor identifier and restricts the events to only transfer `DEPARTING` events:

```
@Interceptor(identifier = "redist", transferEvents = TransferEvent.Type.DEPARTING)
public class RedistributionInterceptor
   implements EventInterceptor<TransferEvent>
```

```
public void onEvent(TransferEvent event)
{
   ...
```

The `@Interceptor` annotation includes the following attributes:

- `identifier` – Specifies a unique identifier for the interceptor. The identifier can be overridden when registering an interceptor class in the cache configuration file. This attribute is optional. A unique name is automatically generated by the event infrastructure if the attribute is omitted.
- `entryEvents` – Specifies an array of entry event types to which the interceptor wants to subscribe.
- `entryProcessorEvents` – Specifies an array of entry processor event types to which the interceptor wants to subscribe.
- `transferEvents` – Specifies an array of transfer event types to which the interceptor wants to subscribe.
- `transactionEvents` – Specifies an array of transaction event types to which the interceptor wants to subscribe.
- `unsolicitedEvents` – Specifies an array of unsolicited event types to which the interceptor wants to subscribe.
- `cacheLifecycleEvent` – Specifies an array of cache lifecycle event types to which the interceptor wants to subscribe.
- `federatedChangeEvents` – Specifies an array of federation change event types to which the interceptor wants to subscribe.
- `federatedConnectionEvent` – Specifies an array of federation connection event types to which the interceptor wants to subscribe.
- `order` – Specifies whether the interceptor is placed at the front of a chain of interceptors. See Chaining Event Interceptors. The legal values are `LOW` and `HIGH`. A value of `HIGH` indicates that the interceptor is placed at the front of the chain of interceptors. A value of `LOW` indicates no order preference. The default value is `LOW`. The order can be overridden when registering an interceptor class in the cache configuration file.

The following example demonstrates a basic event interceptor implementation that subscribes to all transfer event types (both `DEPARTING` and `ARRIVED`). The `onEvent` method simply logs the events to show partition activity. The example is part of the Coherence examples.

> **Note:**
>
> Event instances are immutable and their lifecycle is controlled by the underlying system. References to event classes must not be held across multiple invocations of the `onEvent()` method.

Example 33-1 Example Event Interceptor Implementation

```
package com.tangosol.examples.events;
import com.tangosol.net.CacheFactory;
import com.tangosol.net.events.EventInterceptor;
import com.tangosol.net.events.annotation.Interceptor;
import com.tangosol.net.events.partition.TransferEvent;
```

```
@Interceptor(identifier = "redist")
public class RedistributionInterceptor
   implements EventInterceptor<TransferEvent>
   {

   public void onEvent(TransferEvent event)
      {
      CacheFactory.log(String.format("Discovered event %s for partition-id %d
         from remote member %s\n", event.getType(), event.getPartitionId(),
         event.getRemoteMember()), CacheFactory.LOG_INFO);
      }
   }
```

Understanding Event Threading

Event interceptors can have a significant impact on cache operations and must be careful not to block or otherwise affect any underlying threads. The impact for both precommit event types and postcommit event types should be carefully considered when creating event interceptors.

> **Note:**
>
> `EventInterceptor` instances can be reused; however, they should be immutable or thread-safe so that they can be dispatched by multiple threads concurrently.

Precommit Events

Precommit event types allow event interceptors to modify entries before the entries are committed to a cache. The interceptors are processed synchronously and must not perform long running operations (such as database access) that could potentially block or slow cache operations. Calls to external resource must always return as quickly as possible to avoid blocking cache operations.

The dynamic thread pool is automatically enabled for partitioned services. The thread pool creates additional threads to process cache operations and helps alleviate the overall impact of event interceptors that handle precommit events, but the potential for blocking still exists. Use the `<thread-count-min>` and `<thread-count-max>` elements within a distributed cache definition to explicitly set the size of the thread pool. See Table B-19.

Postcommit Events

Postcommit events do not allow an event interceptor to modify entries. The events are raised in the same order as the events occurred and the interceptors are processed asynchronously. Event interceptors for postcommit entry events are processed on the cache service's `EventDispatcher` thread. Event interceptors that perform long running operations can cause a backlog of requests that could ultimately affect performance. It is a best practice to use the Java Executor service to perform such operations on a separate thread.

Registering Event Interceptors

Event interceptors are registered within a cache configuration file. Event interceptors are registered either for a specific cache or for a partitioned service. Event interceptor that are registered for a specific cache only receives events that pertain to that cache. Event interceptors that are registered for a partitioned service receive events for all caches that are managed by the service.

> **Note:**
>
> Event interceptors for service-level events (such as transfer events, transaction events, and federation events) must be registered for a partition service and cannot be restricted to a specific cache.

This section includes the following topics:

- Registering Event Interceptors For a Specific Cache
- Registering Event Interceptors For a Partitioned Service
- Registering Event Interceptors For a Cache Configuration Factory
- Using Custom Registration
- Guidelines for Registering Event Interceptors

Registering Event Interceptors For a Specific Cache

To register interceptors on a specific cache, include an `<interceptors>` element, within a `<cache-mapping>` element, that includes any number of `<interceptor>` subelements. Each `<interceptor>` element must include an `<instance>` subelement and provide a fully qualified class name that implements the `EventInterceptor` interface. See interceptor element. The following example registers and event interceptor class called `MyInterceptor`.

> **Note:**
>
> Registering interceptors in a cache map is intended only for cache-level events and not for service-level events. If you register service-level event interceptors in a cache map, then the interceptor is triggered on all caches that belong to the service.

```
<caching-scheme-mapping>
   <cache-mapping>
      <cache-name>example</cache-name>
      <scheme-name>distributed</scheme-name>
       <interceptors>
          <interceptor>
             <name>MyInterceptor</name>
             <instance>
                <class-name>
                   com.tangosol.examples.events.MyInterceptor
                </class-name>
             </instance>
          </interceptor>
      </interceptors>
   </cache-mapping>
</caching-scheme-mapping>
```

Registering Event Interceptors For a Partitioned Service

To register interceptors on a partitioned service, include an `<interceptors>` element, within a `<distributed-scheme>` element, that includes any number of `<interceptor>` subelements. Each `<interceptor>` element must include an `<instance>` subelement and provide a fully

qualified class name that implements the `EventInterceptor` interface. See interceptor. The following example registers the `RedistributionInterceptor` class defined in Example 33-1.

```
<distributed-scheme>
   <scheme-name>distributed</scheme-name>
   <service-name>PartitionedService1</service-name>
   <backing-map-scheme>
      <local-scheme/>
   </backing-map-scheme>
   <autostart>true</autostart>
   <interceptors>
      <interceptor>
         <name>MyInterceptor</name>
         <instance>
            <class-name>
               com.tangosol.examples.events.RedistributionInterceptor
            </class-name>
         </instance>
      </interceptor>
   </interceptors>
</distributed-scheme>
```

Registering Event Interceptors For a Cache Configuration Factory

To register interceptors on a `ConfigurableCacheFactory` instance, include an `<interceptors>` element, within a `<cache-config>` element, that includes any number of `<interceptor>` subelements. Each `<interceptor>` element must include an `<instance>` subelement and provide a fully qualified class name that implements the `EventInterceptor` interface. See interceptor. The following example registers and event interceptor class called `MyInterceptor`.

```
<cache-config>
   <interceptors>
      <interceptor>
         <name>MyInterceptor</name>
         <instance>
            <class-name>com.tangosol.examples.events.MyInterceptor</class-name>
         </instance>
      </interceptor>
   </interceptors>
   ...
</cache-config>
```

Using Custom Registration

The `@Interceptor` annotation and generic types are used by the event infrastructure to register event interceptors with the appropriate event dispatcher. This mechanism is acceptable for most uses cases. However, for advanced use cases, an event interceptor can choose to implement the `EventDispatcherAwareInterceptor` interface and manually register an event interceptor with the required event dispatcher.

The `introduceEventDispatcher` method includes the event dispatcher to which the interceptor will be registered. The methods on the dispatcher are then used to add and remove interceptors, restrict specific event types, and configure the interceptor as required. The following example shows a custom implementation that explicitly registers an interceptor, subscribes to entry `INSERTING` events, and configures ordering to ensure that the interceptor is called and notified first:

```
public void introduceEventDispatcher(String sIdentifier, EventDispatcher
   dispatcher)
```

```
{
dispatcher.addEventInterceptor(sIdentifier, this,
   new HashSet(Arrays.asList(EntryEvent.Type.INSERTING)), true);
}
```

> **Note:**
>
> If an interceptor is configured without using the annotations, then the configuration cannot be overridden using the cache configuration file.

Interceptors can also be programmatically registered using the `InterceptorRegistry` API. Registering an interceptor causes it to be introduced to all currently registered and future event dispatchers. An interceptor can determine whether or not to bind to a dispatcher by using the `introduceEventDispatcher` method as shown in the previous example.

The `InterceptorRegistry` API is available from the `ConfigurableCacheFactory` interface and is called using the `getInterceptorRegistry` method. The API can be used together with the cache configuration file when declaratively registering interceptors. The API is often used with custom `DefaultCacheServer` implementations to add interceptors programmatically, or the API is used to selectively register interceptors when using the `InvocationService` interface. The following example demonstrates registering an interceptor.

```
CacheFactory.getConfigurableCacheFactory().getInterceptorRegistry()
.registerEventInterceptor(new MyEventIntercepor());
```

Guidelines for Registering Event Interceptors

Interceptors can be registered in multiple distributed schemes for the same service. In addition, interceptor classes can be inherited if a distributed scheme uses scheme references. In both cases, the interceptor classes are registered with the service.

For most cases, registering multiple interceptor classes is not an issue. However, there are increased chances for duplicating the same interceptor classes and identifier names for a given service. The following guidelines should be followed to ensure registration errors do not occur because of duplication:

- An interceptor class can be duplicated multiple time in a distributed scheme or in multiple schemes of the same service as long as the identifier names are unique or no identifier name is defined. For the later, a unique name is automatically generated by the event infrastructure.
- An interceptor class (duplicated or not) with the same identifier name cannot be registered multiple times in a distributed scheme and results in a registration error.
- An interceptor class that is inherited by scheme reference is registered once for each distinct service name in the reference chain.

Chaining Event Interceptors

This section includes the following topics:

- Overview of Chaining Event Interceptors
- Specifying an Event Interceptor Chain Order

Overview of Chaining Event Interceptors

Event interceptors that are registered for the same event type are serially called by the thread responsible for dispatching an event. The ability to chain interceptors in this manner allows complex processing scenarios where custom logic is executed based on the outcome of other interceptors in the chain. Each event interceptor in a chain can:

- Modify data associated with the event if allowed. For example, precommit operations such as `INSERTING`, `UPDATING`, and `REMOVING` entry events allow data modifications.
- Deny a precommit operations by throwing an exception.
- Enlist a new entry into a partition-level transaction.
- Observe the results of any side effects caused by event interceptors further down the chain. If the interceptor chain is associated with a precommitted storage event, the ability to observe the results provides a second opportunity to deny the processing.

  Observing the side effects of downstream event interceptors is accomplished using the `Event.nextInterceptor` method. When this method returns, it signifies that all downstream event interceptors executed and any changes to the state associated with the event can be inspected. The `Event` object holds state on the processing of the event. The event calls each event interceptor's `onEvent` method as long as there are still event interceptors in the chain. If the event interceptor itself calls the `Event.nextInterceptor` method, it triggers the next event interceptor in the chain to execute. If the event interceptor returns, the event itself triggers the next event interceptor in the chain's `onEvent` method. Either way, all event interceptors are executed, but those that choose to call the `nextInterceptor` method have the option of taking action based on the side effects that are observed.

Specifying an Event Interceptor Chain Order

Event interceptors in a chain are executed based on the order in which they are registered. Use the `order` attribute on the `@Interceptor` annotation to specify whether an interceptor is placed in the front of the chain. A value of `HIGH` places the interceptor at the front of the chain. A value of `LOW` indicates no order preference and is the default value. For example:

```
@Interceptor(identifier = "MyInterceptor",
             entryEvents = {Type.INSERTING, Type.INSERTED}
             order = Order.HIGH)
public class MyInterceptor
        implements EventInterceptor<EntryEvent<?,?>>
...
```

The order can also be specified declaratively when registering an event interceptor in the cache configuration file and overrides the `order` attribute that is specified in an event interceptor class.

To specify the ordering of interceptors in the cache configuration file, include an `<order>` element, within an `<interceptor>` element, that is set to `HIGH` or `LOW`. For example:

```
<interceptors>
   <interceptor>
      <name>MyInterceptor</name>
      <order>HIGH</order>
      <instance>
         <class-name>package.MyInterceptor</class-name>
      </instance>
   </interceptor>
```

```
   <interceptor>
      <name>MySecondInterceptor</name>
      <instance>
         <class-name>package.MySecondInterceptor</class-name>
      </instance>
   </interceptor>
</interceptors>
```

34

Using Coherence Query Language

You can 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.

> **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.

This chapter includes the following sections:

Understanding Coherence Query Language Syntax

CohQL includes many statements that are used to interact with a cache.

Review each statement description, the statement syntax, and an example of how the statement is used. See also Additional Coherence Query Language Examples.

> **Note:**
>
> CohQL does not support subqueries.

This section includes the following topics:

- Working with Indexes
- Issuing Multiple Query Statements
- Persisting Cache Data to Disk
- Viewing Query Cost and Effectiveness
- Handling Errors

Coherence Query Language Statements

Table 34-1 lists the Coherence query statements, clauses, and expressions in alphabetical order.

Table 34-1 Coherence Query Language Statements

For this statement, clause, or expression...	See this section
ARCHIVE SNAPSHOT	Archiving Snapshots
bind variables	Using Bind Variables
CREATE CACHE	Creating a Cache
CREATE INDEX	Creating an Index on the Cache
CREATE SNAPSHOT	Creating Snapshots
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
LIST [ARCHIVED] SNAPSHOTS	Recovering Snapshots and Retrieving Archived Snapshots
LIST ARCHIVER	Archiving Snapshots
path-expressions	Using Path-Expressions
RECOVER SNAPSHOT	Recovering Snapshots
REMOVE [ARCHIVED] SNAPSHOT	Removing Snapshots
RESUME SERVICE	Suspending Services During Persistence Operations
RETRIEVE ARCHIVED SNAPSHOT	Retrieving Archived Snapshots
SELECT	Retrieving Data from the Cache
SOURCE	Issuing Multiple Query Statements
SUSPEND SERVICE	Suspending Services During Persistence Operations
TRUNCATE	Deleting Entries in the Cache
UPDATE	Changing Existing Values
VALIDATE ARCHIVED SNAPSHOT	Validating Archived Snapshots
VALIDATE SNAPSHOT	Validating Snapshots

Table 34-1 (Cont.) Coherence Query Language Statements

For this statement, clause, or expression...	See this section
value() pseudo-function	Using Key and Value Pseudo-Functions
WHENEVER COHQLERROR THEN [EXIT| CONTINUE]	Handling Errors
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.

This section includes the following topics:

- Using Path-Expressions
- Using Bind Variables
- Using Key and Value Pseudo-Functions
- Using Aliases
- Using Quotes with Literal Arguments

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 a Java Bean, Java Record, or JSON Object. It is backed by a `UniversalExtractor` that can extract a property from any of the above target types. 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 on a Java Bean target:

```
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.

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. See Key and Value Pseudo-Function Examples and A Command-Line Example.

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.

Managing the Cache Lifecycle

This section describe how to create and remove caches.

This section includes the following topics:

- Creating a Cache
- Removing a Cache from the Cluster
- Writing a Serialized Representation of a Cache to a File
- Restoring Cache Contents from a File

Creating a Cache

Use the `CREATE CACHE` or `ENSURE CACHE` statements to create a new cache or connect to an existing cache, respectively. This statement first attempts to connect to a cache with the specified *`cache-name`*. If the cache is not found in the cluster, an attempt is made 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, then 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"
  ```

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"
  ```

Writing a Serialized Representation of a Cache to a File

> **Note:**
>
> The `BACKUP CACHE` statement is deprecated. Use the persistence statements instead. See Persisting Cache Data to Disk.

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"
```

Restoring Cache Contents from a File

> **Note:**
>
> The `RESTORE CACHE` statement is deprecated. Use the persistence statements instead. See Persisting Cache Data to Disk.

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"
```

Retrieving Data

This section describe the `SELECT` statement and the `WHERE` clause. These entities are the basic building blocks of most cache queries.

This section includes the following topics:

- Retrieving Data from the Cache
- Filtering Entries in a Result Set

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.

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 `LIKE` operator performs its comparison in a case-sensitive manner. If the comparison should not be case-sensitive, use the `ILIKE` operator instead.

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.

See also Simple SELECT * FROM Statements that Highlight Filters.

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
  ```

Working with Cache Data

This section describe how to work with data in the cache, such as inserting and deleting cache data and filtering result sets.

This section includes the following topics:

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`. The functions can operate on the `Double`, `Long`, and `BigDecimal` types except for the `COUNT` function, which calculates the number of values in an entry set and does not require a type. To specify a type for a function, include the type followed by an underscore (_) as a prefix to the function. For example:

```
long_sum, bd_sum
```

The `Double` type is assumed if a type is not explicitly specified.

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
  ```

- Select the total `amount` and average `price` (using a `BigDecimal` type) for items from the `orders` cache, grouped by `supplier`.

  ```
  select supplier,bd_sum(amount),bd_avg(price) from "orders" group by supplier
  ```

See Complex Queries that Feature Projection, Aggregation, and Grouping.

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.

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.

> **Note:**
>
> 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"
```

Use the `TRUNCATE` statement to delete all entries in the cache. The removal of entries caused by the `TRUNCATE` operation is not observable by listeners, triggers, and interceptors. However, a `CacheLifecycleEvent` event is raised to notify all subscribers of the execution of this operation. See Clearing Caches for more information.

The syntax is:

```
TRUNCATE CACHE "cache-name"
```

For example, to delete all entries from the employee cache, use:

```
truncate cache "employee"
```

Working with Indexes

This section 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.

This section includes the following topics:

- Creating an Index on the Cache
- Removing an Index from the Cache

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 `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
  ```

Persisting Cache Data to Disk

The statements in this section are used to backup and restore caches. The persistence statements rely on the persistence settings that are configured for a service. See Persisting Caches in *Administering Oracle Coherence*.

This section includes the following topics:

- Creating Snapshots
- Validating Snapshots
- Recovering Snapshots
- Archiving Snapshots
- Validating Archived Snapshots
- Retrieving Archived Snapshots
- Removing Snapshots
- Forcing Recovery
- Suspending Services During Persistence Operations

Creating Snapshots

The `CREATE SNAPSHOT` statement persists the data partitions of a service to disk. The syntax is:

```
CREATE SNAPSHOT "snapshot-name" "service"
```

The *`snapshot-name`* argument is any user defined name and can include any of the following macros: `%y` - Year, `%m` - Month, `%d` - Day, `%hh` - Hour, `%mm` - Minute, `%w` - weekday, `%M` - Month name. The *`service`* argument is the name of the partitioned or federated cache service for which the snapshot is created.

Example:

Create a snapshot that is named `Backup` for a partitioned cache service that is named `OrdersCacheService`.

```
create snapshot "Backup" "OrdersCacheService"
```

Use the `LIST SERVICES` statement to see all active services and the currently configured persistence mode, quorum, and status of each service. Include the `ENVIRONMENT` option to see details about the persistence environment configuration. For example:

```
list services environment
```

Validating Snapshots

The `VALIDATE SNAPSHOT` statement is used to check whether a snapshot is complete and without error. The syntax is:

```
VALIDATE SNAPSHOT ["snapshot-directory" | "snapshot-name" "service-name"] [VERBOSE]
```

The *`snapshot-directory`* argument is the full path to a snapshot and must include the snapshot name. The *`snapshot-name`* argument is the name of an archived snapshot to be validated. The *`service-name`* argument is the name of the partitioned or federated cache service for which the snapshot was created. If the *`snapshot-name`* and *`service-name`* arguments are used, then the location is derived. The default snapshot location is *`USER_HOME`*`/coherence/snapshots`. To see detailed validation information, use the `VERBOSE` option. When specifying the `VERBOSE` option, each partition in the snapshot is opened and read and a summary of the caches persisted, including the number of entries and actual size, are displayed. Information about the number indexes, triggers, locks, and listeners are also displayed.

Example:

Validate a snapshot that is named `Backup`.

```
validate snapshot "c:\coherence\snapshots\MyCluster\OrdersCacheService\Backup"
   verbose
```

Validate a snapshot that is named `Backup` which is managed by the `OrdersCacheService` service:

```
validate snapshot "Backup" "OrdersCacheService" verbose
```

Recovering Snapshots

The `RECOVER SNAPSHOT` statement restores the data partitions of a service from disk. Any existing data in the caches of a service are lost.

> **Caution:**
>
> recovering a snapshot causes the current contents of the caches within the service to be dropped.

The syntax is:

```
RECOVER SNAPSHOT "snapshot-name" "service"
```

The *snapshot-name* argument is the name of a snapshot to recover. The *service* argument is the name of the partitioned or federated cache service for which the snapshot was created. If the service has not been explicitly suspended, then: the service is suspended; the snapshot recovered; and the service is resumed. See Suspending Services During Persistence Operations.

Example:

Recover a snapshot that is named `Backup` for a partitioned cache service that is named `OrdersCacheService`.

```
recover snapshot "Backup" "OrdersCacheService"
```

Use the `LIST SNAPSHOTS` statement to see a list of available snapshots. For example:

```
list snapshots "OrdersCacheService"
```

Archiving Snapshots

The `ARCHIVE SNAPSHOT` statement saves a snapshot to a central location. The location is specified in the snapshot archiver definition that is associated with a service. The syntax is:

```
ARCHIVE SNAPSHOT "snapshot-name" "service"
```

The *snapshot-name* argument is the name of a snapshot to archive. The *service* argument is the name of the partitioned or federated cache service for which the snapshot was created.

Example:

Archive a snapshot that is named `Backup` for a partitioned cache service that is named `OrdersCacheService`.

```
archive snapshot "Backup" "OrdersCacheService"
```

Use the `LIST ARCHIVER` statement to view the snapshot archiver definition location that is currently associated with a service. For example:

```
list archiver "OrdersCacheService"
```

Validating Archived Snapshots

The `VALIDATE ARCHIVED SNAPSHOT` statement is used to check whether an archived snapshot is complete and without error. The syntax is:

```
VALIDATE ARCHIVED SNAPSHOT "snapshot-name" "service-name" [VERBOSE]
```

The *snapshot-name* argument is the name of an archived snapshot to be validated. The *service-name* argument is the name of the partitioned or federated cache service for which the snapshot was created. To see detailed validation information, use the `VERBOSE` option.

> **Note:**
>
> - The cluster operational configuration file and cache configuration file must be available in the classpath so that the defined snapshot archiver can be found.
> - Validating archived snapshots involves retrieving each individual partition, unpacking and validating the contents. The operating system default Java temporary directory is used for this operation (for example, the `TEMP` environment variable on Windows environments). If an archived snapshot is large, then consider changing the default Java temporary directory by using `-Djava.io.tmpdir=/path`.

Example:

Validate an archived snapshot that is named `Backup`.

```
validate archived snapshot "Backup" "OrdersCacheService" verbose
```

Retrieving Archived Snapshots

The `RETRIEVE ARCHIVED SNAPSHOT` statement is used to retrieve an archived snapshot so that it can be recovered using the `RECOVER SNAPSHOT` statement. See Recovering Snapshots. The syntax is:

```
RETRIEVE ARCHIVED SNAPSHOT "snapshot-name" "service" [OVERWRITE]
```

The *`snapshot-name`* argument is the name of an archived snapshot to retrieve. The *`service`* argument is the name of the partitioned or federated cache service for which the snapshot was created. Use the `OVERWRITE` option if a snapshot with the same name already exist in the persistence snapshot directory.

Example:

Retrieve an archived snapshot that is named `Backup` for a partitioned cache service that is named `OrdersCacheService` and overwrite the existing snapshot.

```
retrieve archived snapshot "Backup" "OrdersCacheService" overwrite
```

Use the `LIST SNAPSHOTS` statement with the `ARCHIVED` option to see a list of available archived snapshots. For example:

```
list archived snapshots "OrdersCacheService"
```

Removing Snapshots

The `REMOVE SNAPSHOT` statement is used to delete a snapshot or an archived snapshot from disk. The syntax is:

```
REMOVE [ARCHIVED] SNAPSHOT "snapshot-name" "service"
```

The *`snapshot-name`* argument is the name of a snapshot to remove. The *`service`* argument is the name of the partitioned or federated cache service for which the snapshot was created. Use the `ARCHIVED` option to remove an archived snapshot.

Example:

Remove a snapshot that is named `Backup` from a partitioned cache service that is named `OrdersCacheService`.

```
remove snapshot "Backup" "OrdersCacheService"
```

Remove an archived snapshot that is named `Backup` from a partitioned cache that is named `OrdersCacheService`.

```
remove archived snapshot "Backup" "OrdersCacheService"
```

Forcing Recovery

The `FORCE RECOVERY` statement is used to proceed with persistence recovery despite dynamic quorum policy objections. See Using the Dynamic Recovery Quorum Policy in *Administering Oracle Coherence*

> **Note:**
>
> Forcing persistence recovery may lead to partial or full data loss at the corresponding cache service.

The syntax is:

```
FORCE RECOVERY "service"
```

The *`service`* argument is the name of the partitioned cache service being recovered.

Example:

Force recovery for a partitioned cache service that is named `OrdersCacheService`.

```
force recovery "OrdersCacheService"
```

Suspending Services During Persistence Operations

Use the `SUSPEND SERVICE` and `RESUME SERVICE` to ensure persistence operations are performed on a non-active service. For some persistence operations, the service is automatically suspended and resumed when the statement is executed. The syntax is:

```
[RESUME | SUSPEND] SERVICE "service"
```

The `service` argument is the name of a partitioned cache service to be suspended or resumed.

> **Note:**
>
> Clients that send requests to a suspended service are blocked until the service resumes and the request completes. Services can be configured with a request timeout to ensure that clients are not blocked while waiting for a service to be resumed. A request timeout can be configured using the `<request-timeout>` element in a distributed scheme definition or the `coherence.distributed.request.timeout` system property.

Example:

- Suspend a partitioned cache service that is named `OrdersCacheService`.

  ```
  suspend service "OrdersCacheService"
  ```

- Resume a partitioned cache service that is named `OrdersCacheService`.

  ```
  resume service "OrdersCacheService"
  ```

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. 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
```

Handling Errors

The `WHENEVER COHQLERROR THEN` statement is used to specify the action CohQL takes when an error is encountered while executing a statement. The statement is often helpful when CohQL is used as part of a script. See Using Command-Line Tool Arguments. The syntax:

```
WHENEVER COHQLERROR THEN [EXIT|CONTINUE]
```

Example:

Exit immediately if the validate snapshot statement returns and error:

```
whenver cohqlerror then exit
validate snapshot "/snapshots/MyCluster/DistCacheService/snapshot"
```

Using the CohQL Command-Line Tool

The CohQL command-line tool provides a non-programmatic way to interact with caches.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 single 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. See 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.

The CohQL command-line tool can use 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. CohQL supports JLine when the JLine library is included in the classpath of the CohQL Command-Line Tool JVM. To use JLine, update the cohql command line script to include version 3.x of the JLine Bundle jar in the classpath. If the JLine library is not found, a message displays and CohQL starts without JLine capabilities.

- Starting the Command-line Tool
- Using Command-Line Tool Arguments
- Setting the Request Timeout
- A Command-Line Example

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. You can also load the Coherence artifacts from a Grid ARchive (GAR) file using the `-g` argument when starting the command-line tool. See Using Command-Line Tool Arguments.

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.

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 34-2 Coherence Query Language Command-Line Tool Arguments

Argument	Description
`-a` *`name`*	Associate an application name with a GAR file. The application name is used to scope caches and services so that they are isolated from other applications that run on the same cluster. This argument is only used if the `-g` argument is specified. If the `-a` argument is not used, then the default application name is assigned when loading a GAR file.
`-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.
`-dp` *`domain-partition list`*	Specifies a comma delimited list of domain partition names to use in a multi-tenant environment. This argument is only used if the `-g` argument is specified. Note that the first domain partition in the list is automatically selected as the active domain partition and is used when creating a cache. To select a different partition, use the domain partition statement from CohQL command line to select the partition: `ALTER SESSION SET DOMAIN PARTITON` *`partition-name`* The *`partition-name`* value is the name of the domain partition to make active.
`-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.

Table 34-2 (Cont.) Coherence Query Language Command-Line Tool Arguments

Argument	Description
`-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.
`-g` *`gar`*	Load the given Grid ARchive (GAR) file or an exploded GAR file directory before running CohQL and use the default application name. The default application name is the GAR file name without the parent directory name. Use the `-a` argument to explicitly specify an application name.
`-l` *`statement`*	Execute the given statement. Statements must be enclosed in single or double quotes. Any number of `-l` arguments can be used.
`-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`).
`-t`	enable trace mode to print debug information.
`-timeout` *`value`*	Specifies the timeout value for CohQL statements in milli-seconds.

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
```

Start the command line tool and load the application artifacts from a GAR file named `contacts.gar` that is located in the `/applications` directory.

```
query.sh -g /applications/contacts.gar
```

Start the command line tool and load the application artifacts from a GAR file named `contacts.gar` that is located in the `/applications` directory. Scope the application name to `HRContacts`.

```
query.sh -g /applications/contacts.gar -a HRContacts
```

Setting the Request Timeout

The default request timeout value for a service is set to infinity unless the value is explicitly set using either the `<request-timeout>` element for a service or the `coherence.distributed.request.timeout` system property for all services.

The CohQL command-line tool can be used to set a timeout value which applies to statements that are executed in the current session. The default timeout value for the command-line tool is 30 seconds. A timeout exception is thrown if a statement takes longer than 30 seconds to execute. There are two ways to change the timeout value:

- The `-timeout` command line argument is used to set the timeout value in milliseconds. For example,

  ```
  query.sh -timeout 60000
  ```

- The `ALTER SESSION` statement changes the timeout from the default or from any value specified by the `-timeout` argument and remains in effect until another `ALTER SESSION` statement is executed. The syntax is:

  ```
  ALTER SESSION SET TIMEOUT value
  ```

  The value can be an integer that specifies the total number of milliseconds. For example:

  ```
  ALTER SESSION SET TIMEOUT 45000
  ```

  The value can also be a string that specifies a duration. Valid values to use in the duration string are `h` for hours, `m` for minutes, `s` for seconds, and `ms` for milliseconds. For example:

  ```
  ALTER SESSION SET TIMEOUT '5m 30s'
  ```

A Command-Line Example

The following examples illustrate using the command-line tool on Windows. This example is intended for a single cluster member, 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 persisting caches to disk. It also highlights the use of the `key()` and `value()` pseudo-functions.

When starting the `query.cmd` script at the command prompt, information about the Java environment, the Coherence version and edition, and Coherence cache server is displayed. Enter query statements at the prompt (`CohQL>`).

Start the CohQL command-line tool:

```
C:/coherence/bin/query.cmd
```

Create a cache named `employees`:

```
CohQL> create cache "employees"
```

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 "employees" 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"
```

Retrieve the values in the cache:

```
CohQL> select * from "employees"
```

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"
```

Create a snapshot of the service to backup the cache to disk:

```
CohQL> create snapshot "Backup" "DistributedCache"
```

Delete an existing key in the cache. An empty result set is returned:

```
CohQL> delete from employees where key() = "David"
```

Delete the contents of the employees cache. An empty result set is returned:

```
CohQL> delete from "employees"
```

Destroy the employees cache:

```
CohQL> drop cache "employees"
```

Re-create the employees cache:

```
CohQL> create cache "employees"
```

Recover the cache contents from the backup:

```
CohQL> recover snapshot "Backup" "DistributedCache"
```

Retrieve the keys and value in the employees cache. Notice that the deleted key and value are present:

```
CohQL> select key(), value() from "employees"
```

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 (see Filtering Entries in a Result Set), 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

Review many additional CohQL examples from basic queries to complex queries.The simple `select *` examples that highlight `Filters` can understand the translation for the `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.

This section includes the following topics:

- Simple SELECT * FROM Statements that Highlight Filters
- Complex Queries that Feature Projection, Aggregation, and Grouping
- UPDATE Examples

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. The `Double` type is assumed.

```
select sum(amount) from "orders"
```

- Select the total `amount` for items from the `orders` cache as a `Long` type.

```
select long_sum(amount) from "orders"
```

- Select the total `amount` for items from the `orders` cache as a `BigDecimal` type.

```
select bd_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. See A Command-Line Example.

- Select the employees from the `ContactInfoCache` whose home address is in Massachusetts, but work out of state.

```
select key().firstName, key().lastName from "ContactInfoCache" where
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
```

Working with Java Objects (POJO's)

For these examples, assume that there is a cache called `customers` that has a POJO `Customer`, which has the following fields and associated getters and setters plus a `toString()` implementation:

```
private int id;
private String name;
private int age;
private String address;
private String city;
private double balance;
```

> **Note:**
>
> Some of these examples are shown earlier (see Additional Coherence Query Language Examples), but this section is specifically about working with POJOS.

- Insert a new customer.

```
insert into "customers" key(1) value new com.oracle.demo.Customer(1,
"Tim", 50, "Address", "Boston", 1000d)
```

```
select key(), value() from customers
```

```
Results
[1, Customer{id=1, name='Tim', age=50, address='Address', city='Boston',
balance=1000.0}]
```

For more information about changing a value, see Changing Existing Values.

- Insert a customer without specifying a key in the command.

```
insert into "customers" value new com.oracle.demo.Customer(2, "Mary", 45,
"Address", "Nashua", 2000d)
```

```
select key(), value() from customers
```

```
Results
[2, Customer{id=2, name='Mary', age=45, address='Address', city='Nashua',
balance=2000.0}]
```

> **Note:**
>
> If you do not specify the key as part of the command, CohQL will look for a `getKey()` method on the POJO to generate the key for the entry.

- Select specific fields from a cache entry.

```
select id, name, city from "customers"
```

```
Results
[1, "Tim", "Boston"]
[2, "Mary", "Nashua"]
```

> **Note:**
>
> Where ever you specify fields other than `key()` or `value()`, in CohQL these are interpreted as path expressions and appropriate getters should exist. See Using Path-Expressions. For example, selecting `id` means that a `getId()` method should exist on the customer class.

> **Tip:**
>
> If you want to see what CohQL will use without executing any statement, you can prefix the commands with `show plan`.
> For example:
>
> ```
> show plan select id, name, city from "customers"
>
> plan: CacheFactory.getCache("customers").aggregate(AlwaysFilter,
> ReducerAggregator(MultiExtractor(.id(), .name(), .city())))
> ```
>
> In the above example, notice that the `getId()`, `getName()`, and `getCity()` accessors are called.

- Select cache entries using filters.

```
select value() from "customers" where city = "Boston";
```

```
Results
Customer{id=4, name='James', age=31, address='Address', city='Boston', balance=300.0}
Customer{id=1, name='Tim', age=50, address='Address', city='Boston', balance=1000.0}
Customer{id=3, name='Helen', age=23, address='Address', city='Boston',
balance=30000.0}
```

For more information about using filters, see Filtering Entries in a Result Set.

- Aggregate cache entries.

```
select city, sum(balance), avg(balance) from "customers" group by city
```

```
Results
"Boston": [31300.0, 10433.333333333334]
"Nashua": [2500.0, 1250.0]
```

For more information about aggregating entries, see Aggregating Query Results.

- Update entries in a cache.
  In this example, the customer balance is set to zero for all Boston customers.

```
update "customers" set balance = 0d where city = "Boston"


Results
4: true
1: true
3: true


select value() from "customers" where city = "Boston";


Results
Customer{id=4, name='James', age=31, address='Address', city='Boston', balance=0.0}
Customer{id=1, name='Tim', age=50, address='Address', city='Boston', balance=0.0}
Customer{id=3, name='Helen', age=23, address='Address', city='Boston', balance=0.0}
```

For more information about changing a value, see Changing Existing Values.

- Delete a cache entry based upon a key.

```
delete from customer where id = 1


Results
```

For more information about deleting an entry, see Deleting Entries in the Cache.

Working with Java Record Class

For these examples, assume that there is a cache called `customers` that has a Java record class named `CustomerRecord` in its values as defined as follows:

```
public record CustomerRecord(int id, String name, int age, String address, String city,
double balance){}
```

> **Note:**
>
> Some of these examples have been shown previously (see Additional Coherence Query Language Examples), but this section is specifically about working with Java Record Class.

- Insert a new customer.

```
insert into "customers" key(1) value new com.oracle.demo.CustomerRecord(1,
"Tim", 50, "Address", "Boston", 1000d)


select key(), value() from customers
```

```
Results
[1, CustomerRecord{id=1, name='Tim', age=50, address='Address', city='Boston',
balance=1000.0}]
```

- Select specific fields from a cache entry.

```
select id, name, city from "customers"
```

```
Results
[1, "Tim", "Boston"]
[2, "Mary", "Nashua"]
```

> **Note:**
>
> Wherever you specify fields other than `key()` or `value()`, in CohQL, these are interpreted as path expressions and appropriate properties should exist. See Using Path-Expressions.

> **Tip:**
>
> If you want to see what CohQL will use without running any statement, you can prefix the commands with `show plan`.
> For example:
>
> ```
> show plan select id, name, city from "customers"
>
> plan: CacheFactory.getCache("customers").aggregate(AlwaysFilter,
> ReducerAggregator(MultiExtractor(.id(), .name(), .city())))
> ```
>
> In the previous example, notice that the `id()`, `name()`, and `city()` accessors are called.

- Select cache entries using filters.

```
select value() from "customers" where city = "Boston";
```

```
Results
CustomerRecord{id=4, name='James', age=31, address='Address', city='Boston',
balance=300.0}
CustomerRecord{id=1, name='Tim', age=50, address='Address', city='Boston',
balance=1000.0}
CustomerRecord{id=3, name='Helen', age=23, address='Address', city='Boston',
balance=30000.0}
```

For more information about using filters, see Filtering Entries in a Result Set.

- Aggregate cache entries.

```
select city, sum(balance), avg(balance) from "customers" group by city
```

```
Results
"Boston": [31300.0, 10433.333333333334]
"Nashua": [2500.0, 1250.0]
```

For more information about aggregating entries, see Aggregating Query Results.

- Delete a cache entry based upon a key.

```
delete from customer where id = 1
```

```
Results
```

For more information about deleting an entry, see Deleting Entries in the Cache.

Working with JSON Objects

The Coherence Query Language provides the ability to insert JSON objects as keys or values. This is specified by providing the `new json` followed by a JSON string, in brackets, for value or key. See the following examples.

To take advantage of this option, you must include the following dependency in your application for all cluster members:

```
<dependency>
    <groupId>${coherence.groupId}</groupId>
    <artifactId>coherence-json</artifactId>
    <version>${coherence.version}</version>
</dependency>
```

- Insert a new JSON object as a value.

```
insert into cache key 1 value new json ('{"customerId": 1, "name": "J.
Smith"}')
```

```
CohQL> select key(), value() from cache
```

Result:

```
[1, {"customerId": 1, "name": "J. Smith"}]
```

- Update a JSON object value.

```
update cache set value() = new json ('{"customerId": 1, "name": "J.
Smithers"}') where key() = 1
```

Result:

```
1: true
```

```
select key(), value() from cachem cache
```

Result:

```
[1, {"customerId": 1, "name": "J. Smithers"}]
```

- Add a JSON object with key and value being JSON.

```
insert into test key new json ('{"foo": 1}') value new json ('{"bar": 2}')


CohQL> select key(), value() from test
```

Result:

```
[{"foo": 1}, {"bar": 2}]
```

- Query using JSON attributes in `where` clause.
  Assuming the following data has been added:

```
{"id": 2, "name": "Customer 2", "type": "SILVER"}
{"id": 1, "name": "Customer 1", "type": "GOLD"}
{"id": 3, "name": "Customer 23", "type": "SILVER"}
```

Then, you can perform the following query:

```
select * from customers where type = 'GOLD'
```

Result:

```
{"id": 1, "name": "Customer 1", "type": "GOLD"}
```

- Select individual JSON attributes.
  Using the same data as in the previous bullet:

```
select name, type from customers where type = 'GOLD'
```

Result:

```
["Customer 1", "GOLD"]
```

Using Extended Language Features

The extended language mode allows object literals in update and insert statements. This enables:

- elements between '[' and']' denote an ArrayList.
- elements between '{' and'}' denote a HashSet.
- elements between '{' and'}' with key/value pairs separated by ':' denote a HashMap.
  A literal HashMap preceded by a class name are processed by calling a zero argument constructor, and then followed by each key pair being turned into a setter and invoked with the value.

To enable extended language, use the `-e` option. For information about using this option, see Using Command-Line Tool Arguments. Alternatively, you can use the CohQL command `EXTENDED LANGUAGE ON`.

This section provides various examples using extended language features.

- ArrayLists
- HashSets
- HashMap

ArrayLists

In the following example, assume a cache that holds results for a person's test, keyed by name, and the value is an ArrayList of integers, which can be repeated. In this example, there is no POJO, but these examples can be applied to attributes of the ArrayList type.

> **Note:**
>
> An array list **can** have duplicate values.

- Insert values which are an ArrayList.

```
insert into results key("Tim") value [90, 80, 85, 90, 100];


insert into results key("Helen") value [95, 98, 99, 90, 100];


select key(), value() from "results"


Results
["Helen", [95, 98, 99, 90, 100]]
["Tim", [90, 80, 85, 90, 100]]
```

- Retrieve cache entries where an ArrayList contains a value.

```
select key() from "results" where value() contains 99


Results
"Helen"
```

- Update the cache entry with a new ArrayList.

```
update results set value() = [100] where key() = "Tim"


Results
"Tim": true


select key(), value() from "results"


Results
["Helen", [95, 98, 99, 90, 100]]
```

```
["Tim", [100]] ["Helen", ["English", "French"]]
["Tim", ["English", "Romanian"]]
```

HashSets

In the following example, assume a cache that holds the names of languages a person speaks, keyed by name, and the value is a HashSet of languages. In this example, there is no POJO, but these examples can be applied to attributes of the HashSet type.

> **Note:**
>
> A HashSet **cannot** have duplicate values.

- Insert values which are a HashSet.

```
insert into "languages" key("Tim") value {"English"};


insert into "languages" key("Helen") value {"English", "French"};


insert into "languages" key("Sharon") value {"English", "Japanese"};


select key(), value() from "languages"


Results
["Sharon", {"English", "Japanese"}]
["Helen", {"English", "French"}]
["Tim", {"English"}]
```

- Retrieve cache entries where an HashSet contains a value.

```
select key() from "languages" where value() contains "French"


Results
"Helen"
```

- Update the cache entry with a new HashSet.

```
update "languages" set value() = {"English", "Romanian"} where key() =
"Tim"


Results
"Tim": true
```

HashMap

In the following example, assume a cache that holds the scores a person has achieved for subjects, keyed by name, and the value is a HashMap subject and score. In this example, there is no POJO, but these examples can be applied to attributes of the HashMap type.

> **Note:**
>
> This is just an example and may not be the best way to store this type of data in a cache.

- Insert values which are HashMaps.

```
insert into scores key("Tim") value{"Maths": 100, "Science": 95,
"English": 70};


insert into scores key("Helen") value{"Maths": 98, "Biology": 95,
"English": 99, "Physics": 87}


select key(), value() from "scores"scores"


Results
["Helen", {"Maths": 98, "English": 99, "Biology": 95, "Physics": 87}]
["Tim", {"Maths": 100, "English": 70, "Science": 95}]
```

- Retrieve cache entries where a HashMap contains more that three entries.

```
select key(), value() from "scores" where value().size() > 3


Results
["Helen", {"Maths": 98, "English": 99, "Biology": 95, "Physics": 87}]
```

- Update the cache entry with a new HashMap.

```
update scores set value() = {"Maths": 99} where key() = "Tim"


select key(), value() from "scores"


Results
["Helen", {"Maths": 98, "English": 99, "Biology": 95, "Physics": 87}]
["Tim", {"Maths": 99}]
```

35
Performing Transactions

Coherence provides several transaction and data concurrency features that can be used as required.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.
This chapter includes the following sections:

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 35-1 summarizes the various transactions option that Coherence offers.

Table 35-1 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. See Using Explicit Locking for Data Concurrency.
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. See Using Entry Processors for Data Concurrency.
Implicit locking	Coherence supports partition-level transactions using implicit locking through the `EntryProcessor` API. Partition-level transactions provide atomic guarantees when updating multiple caches in a single operation. See Processing Entries in Multiple Caches.
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. See Using the Transaction Framework API.
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. See Using the Coherence Resource Adapter.

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`:

```
...
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. Note that if a timeout value is not passed to the `lock` method, then 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. See DistributedCache Service Parameters.

Using Entry Processors for Data Concurrency

The `InvocableMap` interface allows for concurrent lock-free execution of processing code within a cache.This processing is performed by an entry processor. In exchange for reduced flexibility compared to the more general `ConcurrentMap` explicit locking API, the `EntryProcessor` API provides the highest levels of efficiency without compromising data reliability. See Processing Data In a Cache.
Since entry processors perform an implicit low-level lock on the entries they are processing, the end user can place processing code in an `EntryProcessor` implementation 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 entry processors:

- Invoke an entry processor on a specific key. Note that the key need not exist in the cache to invoke an entry processor on it.
- Invoke an entry processor on a collection of keys.
- Invoke an entry processor on a `Filter`. In this case, the `Filter` is executed against the cache entries. Each entry that matches the `Filter` criteria has the entry processor executed against it. See 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` implementations 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 35-1 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 35-2 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;
    }
}
```

entry processors are individually executed atomically; however, multiple entry processor invocations, through the use of `InvocableMap.invokeAll()`, do not execute as one atomic unit. As soon as an individual entry processor has completed, any updates made to the cache is immediately visible while the other entry processors are executing. Furthermore, an uncaught exception in an entry processor does not prevent the other entry processors from executing. Should the primary node for an entry fail while executing an entry processor, the backup node performs the execution instead. Even if the node fails after the completion of an entry processor, idempotency is maintained and the entry processor is not executed twice.

Generally, entry processors should be short lived. Applications with longer running entry processor cause additional threads in the distributed service thread pool to be created so that other operations performed by the distributed service are not blocked by a long running entry processor. See DistributedCache Service Parameters.

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.

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.
- Persistence – This feature is not supported for transactional caches.
- Key Partitioning Strategy – You cannot specify a custom `KeyPartitioningStrategy` for a transactional cache; although, `KeyAssociation` or a custom `KeyAssociator` works.

The Transaction Framework API is also the underling transaction framework for the Coherence JCA resource adapter. See Using the Coherence Resource Adapter.

This section includes the following topics:

- Defining Transactional Caches
- Performing Cache Operations within a Transaction
- Creating Transactional Connections
- Using Transactional Connections
- Using the OptimisticNamedCache Interface
- Configuring POF When Performing Transactions
- Configuring Transactional Storage Capacity
- Performing Transactions from Java Extend Clients
- Viewing Transaction Management Information

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. See transactional-scheme.

> **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.

Example 35-3 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-min>2</thread-count-min>
         <thread-count-max>10</thread-count-max>
      <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-min>2</thread-count-min>
         <thread-count-max>10</thread-count-max>
   </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:

- Applications use the `NamedCache` API to implicitly perform cache operations within a transaction.
- Applications use the `Connection` API to explicitly perform cache operations within a transaction.
- 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. See Using the Coherence Resource Adapter.

This section includes the following topics:

- Using the NamedCache API
- Using the Connection API

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.

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 35-3. 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 35-3. The `Connection` API is discussed in detail following the examples.

Example 35-4 demonstrates an auto-commit transaction; where, two `insert` operations are each executed as separate transactions.

Example 35-4 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 35-5 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 35-5 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 35-6 demonstrates performing a transaction that spans multiple caches. Each transactional cache must be defined in a cache configuration file.

Example 35-6 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 35-4 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 `-Dcoherence.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. See Defining Transactional Caches.
- `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. See Defining Transactional Caches.
- `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.
- A transaction uses the read committed isolation level. See Setting Isolation Levels.
- A transaction immediately performs operations on the cache. See Using Eager Mode.
- A transaction has a default timeout of 300 seconds. See Setting Transaction Timeout.

A connection's default behaviors can be changed using the `Connection` instance's methods as required.

This section includes the following topics:

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 35-4 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 35-5 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.

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 35-4 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. See Defining Transactional Caches.

```
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.

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 are required to perform transactions from Java extend clients. The instructions 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 *Developing Remote Clients for Oracle Coherence*. For details on performing transactions from

C++ or .NET clients, see Performing Transactions for C++ Clients and Performing Transactions for .NET Clients in *Developing Remote Clients for Oracle Coherence*.

This section includes the following topics:

- 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 35-7 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 35-7 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. See Defining Transactional Caches.

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 35-7. 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-min>2</thread-count-min>
         <thread-count-max>10</thread-count-max>
         <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>
         <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 35-7.

```
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 Using JMX to Manage Oracle Coherence in *Managing Oracle Coherence*.

This section describes two MBeans that provide transaction information: `Cache` MBean and `TransactionManager` MBean.

This section includes the following topics:

- Cache MBeans for Transactional Caches
- TransactionManager MBean

Cache MBeans for Transactional Caches

The `Cache` MBean 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 35-2 describes the `Cache` MBean attributes that are supported for transactional caches.

Table 35-2 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.
`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.

TransactionManager MBean

The `TransactionManager` MBean 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 35-3 describes TransactionManager MBean attributes.

Table 35-3 TransactionManager MBean 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.
`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 `TransactionManager` MBean 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.

This section includes the following topics:

- Performing Cache Operations within a Transaction
- Packaging the Application
- Using the Coherence Cache Adapter for Transactions

Performing Cache Operations within a Transaction

This section includes the following topics:

- Overview Performing Cache Operations within a Transaction
- Creating a Coherence Connection
- Getting a Named Cache
- Demarcating Transaction Boundaries

Overview 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 35-8 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 35-9 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 35-3.

Example 35-8 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 35-9 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 35-9 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. See Defining Transactional Caches.

A `Connection` instance always has an associated transaction which is scoped within the connection. A new transaction is started when a transaction is completed. See Using Transactional Connections. The following default behaviors are associated with a connection. .

- 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 35-9 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 35-9 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.

This section includes the following topics:

- 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 35-9:

```
<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 Deploying Coherence Applications in *Administering Oracle Coherence*.

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 35-10 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.

Example 35-10 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 35-11 demonstrates performing cache operations within a transaction when using the `CacheAdapter` class with the `coherence-tx.rar` resource adapter.

Example 35-11 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);
        }
    }
```

36
Working with Partitions

You can use data affinity with Coherence and also change the default partition setup.The instructions are specific to distributed caches.
This chapter includes the following sections:

- Specifying Data Affinity
- Changing the Number of Partitions
- Changing the Partition Distribution Strategy
- Logging Partition Events
  The most commonly used service in Coherence to store and access data is the distributed / partitioned service. This service offers partitioned access to store and retrieve data, and thus provides scalability, in addition to redundancy by ensuring that replicas are in sync.

Specifying Data Affinity

Learn about using data affinity with Coherence; how to configure data affinity; and review some data affinity examples.
This section includes the following topics:

- Overview of Data Affinity
- Specifying Data Affinity with a KeyAssociation
- Specifying Data Affinity with a KeyAssociator
- Deferring the Key Association Check
- Example of Using Affinity

Overview of 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. These include 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).

Specifying Data Affinity with a KeyAssociation

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

Example 36-1 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 class the implements the `KeyAssociator` interface:

Example 36-2 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 is configured for a `NamedCache` in the `<distributed-scheme>` element that defined the cache:

Example 36-3 Configuring a Key Associator

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

Deferring the Key Association Check

Key association can be implemented either on the cluster or on the extend client. When using extend clients, the best practice is to implement key association on the client, which provides the best performance by processing the keys before they are sent to the cluster. Key association is processed on the client by default. Existing client implementations that rely on key association on the cluster must set the `defer-key-association-check` parameter in order to force the processing of key classes on the cluster.

To force key association processing to be done on the cluster side instead of by the extend client, set the `<defer-key-association-check>` element, within a `<remote-cache-scheme>` element, in the client-side cache configuration to `true`. For example:

```
<remote-cache-scheme>
   ...
   <defer-key-association-check>true</defer-key-association-check>
</remote-cache-scheme>
```

> **Note:**
>
> If the parameter is set to `true`, a key class implementation must be found on the cluster even if key association is no being used.

See Implementing a Java Version of a .NET Object and Implementing a Java Version of a C++ Object) in *Developing Remote Clients for Oracle Coherence* for more information on deferring key association with .NET and C++ clients, respectively.

Example of Using Affinity

Example 36-4 illustrates how to use affinity to create a more efficient query (`NamedCache.entrySet(Filter)`) and cache access (`NamedCache.getAll(Collection)`).

Example 36-4 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 Number of Partitions

The default partition count for a distributed cache service is 257 partitions. Setting the system property override, `coherence.service.partitions`, overrides the default partition count for all distributed services.

Each cache server in the cluster that hosts a distributed cache service manages a balanced number of the partitions. For example, each cache server in a cluster of four cache servers manages 64 partitions. The default partition count is typically acceptable for clusters containing up to 16 cache servers. However, larger clusters require more partitions to ensure optimal performance.

All members of the same service must have the same consistent partition count. Changing the partition count with active persistence is not supported out-of-the-box. See Workarounds to Migrate a Persistent Service to a Different Partition Count in *Administering Oracle Coherence*.

This section includes the following topics:

- Define the Partition Count
- Deciding the number of Partitions

Define the Partition Count

To change the number of partitions for a distribute cache service, edit the cache configuration file and add a `<partition-count>` element, within the `<distributed-scheme>` element, that includes the number of partitions to use for the service. For example:

```
<distributed-scheme>
   <scheme-name>distributed</scheme-name>
   <service-name>DistributedCache</service-name>
```

```
   <partition-count>1181</partition-count>
   ...
</distributed-scheme>
```

To have finer control over configuring the partition count for a specific distributed service, using a system property override rather than changing a cache configuration file, set the system property, `coherence.service.<scoped-service-name>.partitions`, to an integer value.

> **Note:**
>
> A scoped service name can have special characters `'/'` (forward slash) and `':'` (colon) in it; replace those characters with a `'.'` (period). For example, scoped service name `tenant/appscope:PartitionedCache` is transformed to `tenant.appscope.PartitionedCache`.

Deciding the number of Partitions

There is no exact formula for selecting a partition count. An ideal partition count balances the number of partitions on each cluster member with the amount of data each partition holds. Use the following guidelines when selecting a partition count and always perform tests to verify that the partition count is not adversely affecting performance.

- The partition count should always be a prime number. A list of primes can be found at http://primes.utm.edu/lists/.
- The number of partitions must be large enough to support a balanced distribution without each member managing too few partitions. For example, a partition count that results in only two partitions on each member is too constraining.
- The number of partitions must not be too large that network bandwidth is wasted with transfer overhead and bookkeeping for many partition transfers (a unit of transfer is a partition). For example, transferring thousands of partitions to a new cache server member requires a greater amount of network resources and can degrade cluster performance especially during startup.
- The amount of data a partition manages must not be too large (the more data a partition manages: the higher the partition promotion and transfer costs). The amount of data a partition manages is only limited by the amount of available memory on the cache server. A partition limit of 50MB typically ensures good performance. A partition limit between 50MB-100MB (even higher with 10GbE or faster) can be used for larger clusters. Larger limits can be used with the understanding that there will be a slight increase in transfer latency and that larger heaps with more overhead space are required.

As an example, consider a cache server that is configured with a 4G heap and stores approximately 1.3G of primary data not including indexes (leaving 2/3 of the heap for backup and scratch space). If the decided partition limit is a conservative 25MB, then a single cache server can safely use 53 partitions (1365M/25M rounded down to the previous prime). Therefore, a cluster that contains 20 cache servers can safely use 1051 partitions (53*20 rounded down to the previous prime) and stores approximately 25G of primary data. A cluster of 100 cache servers can safely use 5297 partitions and can store approximately 129G of primary data.

Changing the Partition Distribution Strategy

Partition distribution defines how partitions are assigned to storage-enabled cluster members.Coherence uses centralized partition distribution to allow global distribution decision to be carried out by each storage-enabled member. Centralized distribution allows for expressive distribution algorithms and uses a complete, global view of the service. In addition, custom centralized distribution strategies can be created by implementing the `com.tangosol.net.partition.PartitionAssignmentStrategy` interface.
This section includes the following topics:

- Specifying a Partition Assignment Strategy
- Enabling a Custom Partition Assignment Strategy

Specifying a Partition Assignment Strategy

The following predefined partition assignment strategies are available:

- `simple` – (default) The simple assignment strategy attempts to balance partition distribution while ensuring machine-safety.
- `mirror:<service-name>` – The mirror assignment strategy attempts to co-locate the service's partitions with the partitions of the specified service. This strategy is used to increase the likelihood that key-associated, cross-service cache access remains local to a member.
- custom – a class that implements the `com.tangosol.net.partition.PartitionAssignmentStrategy` interface.

To configure a partition assignment strategy for a specific partitioned cache service, add a `<partition-assignment-strategy>` element within a distributed cache definition:

```
<distributed-scheme>
   ...
   <partition-assignment-strategy>mirror:<MyService>
   </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>mirror:<MyService></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. A custom class can also extend the `com.tangosol.net.partition.SimpleAssignmentStrategy` class. See instance. 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>
```

Logging Partition Events

The most commonly used service in Coherence to store and access data is the distributed / partitioned service. This service offers partitioned access to store and retrieve data, and thus provides scalability, in addition to redundancy by ensuring that replicas are in sync.

The concept of partitioning can be entirely opaque to you (the end user) as Coherence transparently maps keys to partitions and partitions to members. As ownership members join and leave the partitioned service, the partitions are redistributed across the new/remaining members, thus avoiding an entire rehash of the data. Coherence also designates replicas providing them an initial snapshot followed by a journal of updates as they occur on the primary member.

These partition lifecycle events (members joining and leaving the service) result in partitions being blocked. Therefore, Coherence attempts to reduce this time of unavailability as much as possible. Until now, there has been minimal means to track this partition unavailability. Logging provides insight into these partition lifecycle events and highlight when they start and end. This feature helps you correlate the increased response times with these lifecycle events.

This section includes the following topics:

Data Availability

To preserve data integrity when partition events occur, for example, partition movements between members, the read and write access to data is temporarily blocked. Data gets blocked when re-distribution takes place or indices are built. The time involved is usually short, but can add up if the cache contains a significant amount of data.

Table 36-1 Types of Events

Event Type	Description
Redistribution	When a server joins or leaves a cluster, many events occur on each member in the cluster, both existing and new, which correspond to the movement of data and backup partitions according to the partitioning strategy. This scheme helps determine the names of members who own partitions and the members who own the backups.
Restoring from backup	After the primary partitions are lost, the backups are moved into the primary storage in the members where they are located. These partitions are locked until the event is complete.
Recovery from persistence	Persistence maintenance, such as snapshot creation and recovery from persistence, will cause the affected partitions to be unavailable.
Index building	If an application needs to have data indexed, it is typically done by calling `addIndex` on a `NamedCache`. If the cache already contains a significant amount of data, or the cost of computing the index per entry (the `ValueExtractor`) is high, this operation can take some time during which any submitted queries will be blocked, waiting for the index data structures to be populated. **Note**: Regular index maintenance, such as adding or deleting elements, does not incur the same unavailability penalty.

Using Partition Events Logs

By default, logging of times when partitions are unavailable is turned off as it generates a significant amount of logs. To enable logging of partition events, set the `coherence.distributed.partition.events` property to `log` and set the log level value to `8` or greater.

For example:

```
-Dcoherence.distributed.partition.events=log
```

Logging Events

The events listed in the table below are logged, one per partition except during the initial assignment of partitions. Along with the events, the owning member and the time it made the partition unavailable, are also logged.

Table 36-2 List of Events Logged

Event Type	Description
`ASSIGN`	The partition is, either initially or because of losing the primary and all backups, assigned to a cluster member.
`PRIMARY_TRANSFER_OUT`	The partition is transferred to a different member.
`BACKUP_TRANSFER_OUT`	The primary partition owner transfers a snapshot of the partition and all its content to the targeted member as it will be in the chain of backup replicas.
`PRIMARY_TRANSFER_IN`	This member receives a partition and all related data for primary ownership. This event occurs due to the `PRIMARY_TRANSFER_OUT` event from the existing owner of the partition.
`RESTORE`	The loss of primary partitions results in backup owners (replicas) restoring data from the backup storage to the primary member for the affected partitions.
`INDEX_BUILD`	Index data structures are populated for the relevant partitions. This event affects queries that use the indices but does not block the key-based data access or mutation.
`PERSISTENCE`	The relevant partitions are made unavailable due to persistence maintenance operations. These operations include recovery from persistence and snapshot creation.

> **Note:**
>
> The times logged for these events are in millisecionds.

Example 36-5 Logging Events

On Member 1:

At startup:

```
2021-06-11 09:26:10.159/5.522 Oracle Coherence GE 14.1.1.2206.1 <D8>
(thread=PartitionedTopicDedicated:0x000A:5, member=1): PartitionSet{0..256}, Owner: 1,
Action: ASSIGN, UnavailableTime: 0
...
```

Application calls `addIndex()` on a cache:

```
2021-06-11 09:28:36.872/152.234 Oracle Coherence GE 14.1.1.2206.1 <D8>
(thread=PartitionedCacheDedicated:0x000B:152, member=1): PartitionId: 43, Owner: 1,
Action: INDEX_BUILD, UnavailableTime: 3
...
```

The partitions listed are transferred to another member, along with backups (note that backups and partitions are not the same):

```
2021-06-11 09:28:45.573/160.935 Oracle Coherence GE 14.1.1.2206.1 <D8>
(thread=DistributedCache:PartitionedCache, member=1): PartitionId: 132, Owner: 1,
Action: BACKUP_TRANSFER_OUT, UnavailableTime: 1
2021-06-11 09:28:45.678/161.040 Oracle Coherence GE 14.1.1.2206.1 <D8>
(thread=DistributedCache:PartitionedCache, member=1): PartitionId: 133, Owner: 1,
Action: BACKUP_TRANSFER_OUT, UnavailableTime: 1
...
2021-06-11 09:28:49.911/165.273 Oracle Coherence GE 14.1.1.2206.1 <D8>
(thread=DistributedCache:PartitionedCache, member=1): PartitionId: 2, Owner: 1, Action:
PRIMARY_TRANSFER_OUT, UnavailableTime: 5
2021-06-11 09:28:50.017/165.379 Oracle Coherence GE 14.1.1.2206.1 <D8>
(thread=DistributedCache:PartitionedCache, member=1): PartitionId: 3, Owner: 1, Action:
PRIMARY_TRANSFER_OUT, UnavailableTime: 3
...
```

On Member 2:

Partitions are received. If the partitions have indices, they are rebuilt immediately after they are received:

```
2021-06-11 09:28:49.805/8.033 Oracle Coherence GE 14.1.1.2206.1 <D8>
(thread=DistributedCache:PartitionedCache, member=2): PartitionId: 1, Owner: 2, Action:
PRIMARY_TRANSFER_IN, UnavailableTime: 1
2021-06-11 09:28:49.806/8.034 Oracle Coherence GE 14.1.1.2206.1 <D8>
(thread=PartitionedCacheDedicated:0x000B:8, member=2): PartitionId: 1, Owner: 2, Action:
INDEX_BUILD, UnavailableTime: 0
```

Member 2 stops, back on Member 1:

Partitions are restored from backup, and the indices related to them are rebuilt:

```
2021-06-11 10:29:19.041/3794.322 Oracle Coherence GE 14.1.1.2206.1 <D8>
(thread=DistributedCache:PartitionedCache, member=1): PartitionId: 0, Owner: 1, Action:
RESTORE, UnavailableTime: 109
2021-06-11 10:29:19.041/3794.322 Oracle Coherence GE 14.1.1.2206.1 <D8>
(thread=DistributedCache:PartitionedCache, member=1): PartitionId: 1, Owner: 1, Action:
RESTORE, UnavailableTime: 109
...
2021-06-11 10:29:19.062/3794.343 Oracle Coherence GE 14.1.1.2206.1 <D8>
(thread=PartitionedCacheDedicated:0x000E:3794, member=1): PartitionId: 1, Owner: 1,
Action: INDEX_BUILD, UnavailableTime: 12
2021-06-11 10:29:19.066/3794.347 Oracle Coherence GE 14.1.1.2206.1 <D8>
(thread=PartitionedCacheDedicated:0x000D:3794, member=1): PartitionId: 0, Owner: 1,
Action: INDEX_BUILD, UnavailableTime: 16
2021-06-11 10:29:19.067/3794.349 Oracle Coherence GE 14.1.1.2206.1 <D8>
(thread=PartitionedCacheDedicated:0x000E:3794, member=1): PartitionId: 2, Owner: 1,
Action: INDEX_BUILD, UnavailableTime: 5
...
```

> **Note:**
>
> The Coherence VisualVM Plug-in provides an option to analyze the logs that are generated with events logging enabled. See Coherence VisualVM Plugin Release Notes.

37
Managing Thread Execution

You can control the execution behavior of Coherence service threads using task timeouts and the `PriorityTask` API for custom execution processing.
This chapter includes the following sections:

- Overview of Priority Tasks
- Setting Priority Task Timeouts
- Creating Priority Task Execution Objects

Overview of 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.

Setting Priority Task Timeouts

You can configure various execution timeout settings.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.
This section includes the following topics:

- Configuring Execution Timeouts
- Execution Timeout Command Line Options

Configuring Execution Timeouts

The `<request-timeout>`, `<task-timeout>`, and the `<task-hung-threshold>` elements are used to configure execution timeouts for a service's worker threads. These timeout settings are configured for a service in a cache configuration file and can also be set using command line parameters. See also Using the Service Guardian .

Table 37-1 describes the execution timeout elements.

Table 37-1 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, entry processors that 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-min>` and `<thread-count-max>` values are positive). If `zero` is specified, then 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-min>` and `<thread-count-max>` values are positive).

The following distributed cache example explicitly configures the service dynamic thread pool, a task time out of `5000` milliseconds, and a task hung threshold of `10000` milliseconds:

```
<caching-schemes>
    <distributed-scheme>
      <scheme-name>example-distributed</scheme-name>
      <service-name>DistributedCache</service-name>
      <thread-count-min>7</thread-count-min>
      <thread-count-max>20</thread-count-max>
      <task-hung-threshold>10000</task-hung-threshold>
      <task-timeout>5000</task-timeout>
    </distributed-scheme>
</caching-schemes>
```

Setting the client request timeout to `15` milliseconds

```
<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`.

Execution 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 37-2 describes the options.

Table 37-2 Command Line Options for Setting Execution Timeout

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

Creating Priority Task Execution Objects

The `PriorityTask` interface enables you to control the ordering in which a service schedules tasks for execution using a thread pool and hold the task 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

This section includes the following topics:

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. Table 37-3 describes these methods.

Table 37-3 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 37-3 (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`. For example:

```
com.tangosol.net.RequestTimeoutException: Request timed out after 4015 millis
        at
com.tangosol.coherence.component.util.daemon.queueProcessor.Service.checkRequestTimeout(S
ervice.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.que
ry(InvocationService.CDB:17)
        at
com.tangosol.coherence.component.util.safeService.SafeInvocationService.query(SafeInvocat
ionService.CDB:1)
```

38

Constraints on Re-Entrant Calls

There are constraints on service re-entrant calls and guidelines for making calls between service threads.
This chapter includes the following sections:

- Overview of Constraints on Re-Entrant Calls
- Re-entrancy, Services, and Service Threads
- Re-entrancy and Listeners

Overview of Constraints on Re-Entrant Calls

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 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.

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, Distributed, or Federated). Applications can make service calls between services and service types.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 federated service named `Fed-Catalog`. Service names are configured in the cache configuration file using the `<service-name>` element.
This section includes the following topics:

- Parent-Child Object Relationships
- Avoiding Deadlock

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 possibly 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 use the maximum configured threads in the 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. In addition, relying on the dynamic thread pool can also 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.

39
Using Timeout with Cache Operations

When many threads access Coherence caches concurrently, callers to Coherence cache operations can be blocked on common locks. This blockage may lead to the accumulation of stuck threads. To help avoid these stuck threads, Coherence uses an interruptible lock so operations can be interrupted after a specified timeout interval. You can use the Timeout API to interrupt and error out the blocking operation after the amount of blocking time exceeds the specified timeout interval.
Here is an example of how to use the Timeout API to interrupt the `getCache` operation if it takes longer than the given two minute timeout interval:

```
try (Timeout t = Timeout.after(2, TimeUnit.MINUTES))
     {
     NamedCache cache = CacheFactory.getCache("myCache");
     }
   catch (InterruptedException e)
     {
     // error out getCache() due to timed out.
     }
```

40

Using the Repository API

Coherence Repository API provides a higher-level, DDD-friendly way to access data managed in Coherence.

Coherence Repository API makes the implementation of data access layer within the applications easier, regardless of which framework you use to implement applications that use Coherence as a data store. It works equally well for plain Java applications and applications that use CDI, where you can simply create your own repository implementations.

It is also the foundation for Micronaut Data (see Micronaut Data with Coherence) and Spring Data repository implementations. All the functionalities described in this chapter are available when using these frameworks as well. The only difference is how you define your own repositories, which is framework-specific and documented separately.

The Repository API is implemented on top of the `NamedMap` API and provides a number of features that make it easier to use for many typical use cases where Coherence is used as a Key-Value data store.

This chapter includes the following sections:

- Features and Benefits
  In addition to the basic CRUD (create, read, update, delete) functionality, the Repository API provides many features that simplify the common data management tasks.
- Implementing a Repository
  Coherence provides an abstract base class `com.oracle.coherence.repository.AbstractRepository`, which your custom repository implementation needs to extend and provide implementation of three abstract methods.
- Performing the Basic CRUD Operations
  You can use the basic operations to add, remove, update, and query the repository
- Performing Server-Side Projections
- Making In-Place Updates
- Using the Stream API and the Data Aggregation API
- Creating Event Listeners
  Coherence not only allows you to store, modify, query, and aggregate your data entities efficiently, but also register to receive event notifications whenever any entity in the repository changes.
- Using the Asynchronous Repository API

Features and Benefits

In addition to the basic CRUD (create, read, update, delete) functionality, the Repository API provides many features that simplify the common data management tasks.

A comprehensive list of its features are provided below:

- Powerful projection features
- Flexible in-place entity updates

- First-class data aggregation support
- Stream API support
- Asynchronous API support
- Event listener support
- Declarative Acceleration and Index Creation
- CDI (Contexts and Dependency Injection) support

Implementing a Repository

Coherence provides an abstract base class `com.oracle.coherence.repository.AbstractRepository`, which your custom repository implementation needs to extend and provide implementation of three abstract methods.

```
/**
 * Return the identifier of the specified entity instance.
 *
 * @param entity  the entity to get the identifier from
 *
 * @return the identifier of the specified entity instance
 */
protected abstract ID getId(T entity);

/**
 * Return the type of entities in this repository.
 *
 * @return the type of entities in this repository
 */
protected abstract Class<? extends T> getEntityType();

/**
 * Return the map that is used as the underlying entity store.
 *
 * @return the map that is used as the underlying entity store
 */
protected abstract M getMap();
```

For example, a repository implementation that can be used to store `Person` entities, with `String` identifiers, can be as simple as:

```
public class PeopleRepository
        extends AbstractRepository<String, Person>
    {
    private NamedMap<String, Person> people;

    public PeopleRepository(NamedMap<String, Person> people)
        {
        this.people = people;
        }

    protected NamedMap<String, Person> getMap()
        {
        return people;
```

```
        }

    protected String getId(Person person)
        {
        return person.getSsn();
        }

    protected Class<? extends Person> getEntityType()
        {
        return Person.class;
        }
    }
```

Where:

- The `getMap` method returns the `NamedMap` that should be used as a backing data store for the repository, which is in this case provided using the constructor argument. However, it can just as easily be injected using CDI (Contexts and Dependency Injection).
- The `getId` method returns an identifier for a given entity.
- The `getEntityType` method returns the class of the entities stored in the repository.

A trivial repository implementation as the above example allows you to access all the Repository API features, which are provided by the `AbstractRepository` class you extended.

However, you are free to add additional business methods to the repository class above that will make it easier to use within your application. The most common example of such methods are the various "finder" methods that your application needs. For example, if your application needs to frequently query the repository to find people based on their name, you may want to add a method for that purpose, as shown below:

```
public Collection<Person> findByName(String name)
        {
        Filter<Person> filter = Filters.like(Person::getFirstName, name)
                                       .or(Filters.like(Person::getLastName,
name));
        return getAll(filter);
        }
```

You can then invoke the `findByName` method directly within the application to find all the people whose first or the last name starts with the letter 'A', for example:

```
for (Person p : people.findByName("A%"))
    {
    // processing
    }
```

Performing the Basic CRUD Operations

You can use the basic operations to add, remove, update, and query the repository

To add new entities to the repository, or replace the existing ones, you can use either the `save` or the `saveAll` method.

The `save` method takes a single entity as an argument and stores it in the backing `NamedMap`:

```
people.save(new Person("555-11-2222", "Aleks", 46));
```

The `saveAll` method allows you to store a batch of entities by passing either a collection or a stream of entities as an argument. After you have some entities stored in a repository, you can query the repository using the `get` and `getAll` methods.

```
// gets a single person by identifier
Person person =
people.get("555-11-2222");

assert person.getName().equals("Aleks");
assert person.getAge() == 46;

// fetches all the people from the repository
Collection<Person> allPeople = people.getAll();

// fetches all the people from the repository, that are aged 18 years or
above.
Collection<Person> allAdults =
people.getAll(Filters.greaterOrEqual(Person::getAge, 18));
```

You can retrieve sorted results by calling `getAllOrderedBy` method and specifying a `Comparable` property by using a method reference. The result of the following example will contain the names of all the people from the repository, sorted by age from the youngest to the oldest:

```
Collection<Person> peopleOrderedByAge = people.getAllOrderedBy(Person::getAge)
```

For more complex use cases, you can specify a `Comparator` to use instead. For example, if you want to sort the results of the `findByName` method used earlier (see Implementing a Repository), first by last name, and then by first name, you can re-implement it as follows:

```
public Collection<Person> findByName(String name)
        {
        Filter<Person> filter = Filters.like(Person::getFirstName, name)
                                       .or(Filters.like(Person::getLastName,
name));
        return getAllOrderedBy(filter,
                               Remote.comparator(Person::getLastName)
                                     .thenComparing(Person::getFirstName));
```

This example uses the Coherence `Remote.comparator` instead of the standard Java `Comparator` to ensure that the specified comparator is serializable and can be sent to the remote cluster members.

Finally, to remove entities from a repository, you can use one of the several `remove` methods:

```
// removes specified entity from the repository
boolean fRemoved = people.remove(person);


// removes entity with the specified identifier from the repository
boolean fRemoved = people.removeById("111-22-3333");
```

In both examples above, the result will be a boolean indicating whether the entity was actually removed from the backing `NamedMap`, and it may be `false` if the entity was not present in the repository.

If you are interested in the removed value itself, you can use the overloads of the methods above that allow you to express that:

```
// removes specified entity from the repository and returns it as the result
Person removed = people.remove(person, true);

// removes entity with the specified identifier from the repository and
returns it as the result
Person removed = people.removeById("111-22-3333", true);
```

> **Note:**
>
> This removal results in additional network traffic. Therefore, unless you really need the removed entity, it is probably best not to ask for it.

The examples above are useful when you want to remove a single entity from the repository. In cases when you want to remove multiple entities as part of a single network call, you should use one of the `removeAll` methods instead. These methods enable you to remove a set of entities by specifying either their identifiers explicitly or the criteria for removal through the `Filter`.

```
// removes all men from the repository and returns 'true' if any entity has
been removed
boolean fChanged = people.removeAll(Filters.equal(Person::getGender,
Gender.MALE));


// removes entities with the specified identifiers from the repository and
returns
// 'true' if any entity has been removed
boolean fChanged = people.removeAllById(Set.of("111-22-3333", "222-33-4444"));
```

Just as with single-entity removal operations, you can also use overloads that allow you to return the removed entities as the result:

```
// removes the names of all men from the repository and returns the map of
removed
// entities, keyed by the identifier
```

```
Map<String, Person> mapRemoved =
        people.removeAll(Filters.equal(Person::getGender, Gender.MALE), true);


// removes the entities with the specified identifiers from the repository
and returns
// the map of removed entities, keyed by the identifier
Map<String, Person> mapRemoved =
        people.removeAllById(Set.of("111-22-3333", "222-33-4444"), true);
```

Performing Server-Side Projections

While querying a repository for a collection of entities that satisfy some criteria is certainly a common and useful operation, sometimes you do not need all the attributes within the entity.For example, if you only need a person's name, querying for and then discarding all the information contained within the `Person` instance is unnecessary and waste of time.
It is the equivalent of executing the following against a relational database:

```
SELECT * FROM PEOPLE
```

The following simple command would suffice:

```
SELECT name FROM PEOPLE
```

Coherence Repository API allows you to limit the amount of data collected by performing server-side projection of the entity attributes you are interested in. For example, if you only need a person's name, you can get just the name:

```
// returns the name of the person with a specified identifier
String name  = people.get("111-22-3333", Person::getName);

// returns the map of names of all the people younger than 18, keyed by the
person's identifier
Map<String, String> mapNames =
        people.getAll(Filters.less(Person::getAge, 18), Person::getName);
```

Obviously, returning either the whole entity or a single attribute from an entity are two ends of the spectrum, and more often than not you need something in between. For example, you may need the person's name and age. For situations such as these, Coherence allows you to use *fragments*:

```
// returns a fragment containing the name and age of the person with a
specified identifier
Fragment<Person> fragment = people.get("111-22-3333",
                                       Extractors.fragment(Person::getName,
Person::getAge));

// retrieves the person's name from a fragment
String name = fragment.get(Person::getName);

// retrieves the person's age from a fragment
int age = fragment.get(Person::getAge);
```

You can perform the same projection across multiple entities using one of the `getAll` methods:

```
//returns a map of fragments containing the name and age of all the people
younger than 18, keyed by the person's identifier
Map<String, Fragment<Person>> fragments = people.getAll(
        Filters.less(Person::getAge, 18),
        Extractors.fragment(Person::getName, Person::getAge));
```

Unlike the relational database, which contains a set of columns for each row in the table, Coherence stores each entity as a full object graph, which means that the attributes can be other object graphs and can be nested to any level. This means that you should also be able to project attributes of the nested objects. For example, the `Person` class may have a nested `Address` object as an attribute, which in turn has `street`, `city`, and `country` attributes. If you want to retrieve the name and the country of a person in a repository, see the following example:

```
// returns a fragment containing the name and the 'Address' fragment of the
person with a specified identifier
Fragment<Person> person = people.get(
        "111-22-3333",
        Extractors.fragment(Person::getName,
                            Extractors.fragment(Person::getAddress,
Address::getCountry)));

// retrieves the person's name from the 'Person' fragment
String name = person.get(Person::getName);

// retrievea the 'Address' fragment from the 'Person' fragment
Fragment<Address> address = person.getFragment(Person::getAddress);

// retrieves the person's country from the 'Address' fragment
String country = address.get(Address::getCountry);
```

Making In-Place Updates

The most common approach for updating data in modern applications is the read-modify-write pattern.

For example, a typical code to update an attribute of a `Person` looks similar to the following:

```
Person person = people.get("111-22-3333");
person.setAge(55);
people.save(person);
```

This is true regardless of whether the underlying data store provides a better, more efficient way of updating data. For example, RDBMS provides stored procedures for that purpose, but very few developers use them because they are not as convenient to use, and do not fit well into popular application frameworks, such as JPA, Spring Data, or Micronaut Data. They also fragment the code base to some extent, splitting the business logic across the application and the data store, and require that some application code is written in SQL.

However, the approach above is suboptimal for the following reasons:

- It doubles the number of network calls the application makes to the data store, increasing the overall latency of the operation.
- It moves (potentially a lot) more data over the network than absolutely necessary.
- It may require expensive construction of a complex entity in order to perform a very simple update operation of a single attribute (this is particularly true with JPA and RDBMS back ends).
- It puts additional, unnecessary load on the data store, which is typically the hardest component of the application to scale.
- It introduces concurrency issues (that is, what should happen if the entity in the data store changes between the initial read and subsequent write), which typically requires that both the read and the write happen within the same transaction.

A much better, more efficient way to perform the updates is to send the update *function* to the data store, and execute it locally, within the data store itself (which is pretty much what stored procedures are for).

Coherence has always had support for these types of updates through *entry processors*, but the Repository API makes it even simpler to do so. For example, the code shown earlier can be rewritten as:

```
people.update("111-22-3333", Person::setAge, 55);
```

The above code is telling Coherence to update the `Person` instance with a given identifier by calling the `setAge` method on it with the number `55` as an argument. This code is not only significantly more efficient, but also shorter and easier to write and read.

Note that it is not important to know where in the cluster a `Person` instance with a given identifier is located. Coherence guarantees that it will invoke the `setAge` method on the entity with a specified ID, on a primary owner, and automatically create a backup of the modified entity for fault tolerance.

It is also worth pointing out that the approach above provides the same benefits stored procedures do in RDBMS, but without the downsides, that is, you are still writing all your code in Java, and keeping it in the same place. This approach allows you to implement rich domain models for your data, and execute business logic on your entities remotely, which works exceptionally well with DDD applications.

Calling a setter on an entity remotely is not sufficient for all data mutation needs. For example, the conventional JavaBean setter returns `void`, but you often want to know what the entity value is after the update. The solution to that problem is simple. Coherence will return the result of the specified method invocation, so all you need to do is change the `setAge` method to implement a fluent API:

```
public Person setAge(int age)
    {
    this.age = age;
    return this;
    }
```

You will now get the modified `Person` instance as the result of the `update` call:

```
Person person = people.update("111-22-3333", Person::setAge, 55);
assert person.getAge() == 55;
```

Sometimes you need to perform more complex updates, or update multiple attributes at the same time. While you can accomplish these by making multiple update calls, it is inefficient because each update will result in a separate network call. Instead, you can use the `update` overload that allows you to specify the function to execute in that situation:

```
Person person = people.update("111-22-3333", p ->
    {
    p.setAge(55);
    p.setGender(Gender.MALE);
    return p;
    });

assert person.getAge() == 55;
assert person.getGender() == Gender.MALE;
```

This way you have full control of the update logic that will be executed, and the return value.

You may sometimes want to update an entity that does not exist in the repository. In this case, you should create a new instance. For example, you may want to create a shopping cart entity for a customer when they add the first item to the cart. While you could implement the code to check whether the `Cart` for a given customer exists, and create a new one if the cart does not exist, it results in network calls that can be avoided if you simply create the `Cart` instance as part of the `Cart::addItem` call. The Repository API allows you to accomplish that by using the optional `EntityFactory` argument:

```
carts.update(customerId,          // the cart/customer identifier
             Cart::addItem,       // the method to invoke on a target 'Cart'
instance
             item,                // the 'CartItem' to add to the cart
             Cart::new            // the 'EntityFactory' to use to create a
new 'Cart' instance if the
                                  // cart with the specified identifier does
not exist
             );
```

The `EntityFactory` interface is simple:

```
@FunctionalInterface
public interface EntityFactory<ID, T>
        extends Serializable
    {
    /**
     * Create an entity instance with the specified identity.
     *
     * @param id identifier to create entity instance with
     *
     * @return a created entity instance
     */
    T create(ID id);
    }
```

It has a single `create` method that accepts entity identifier and returns a new instance of the entity with a given identifier. In the example above, it implies that the `Cart` class has a constructor similar to the following:

```
public Cart(Long cartId)
    {
    this.cartId = cartId;
    }
```

Just like with projections and other operations, in addition to the `update` methods that you can use to modify a single entity, there are also a number of `updateAll` methods that you can use to modify multiple entities in a single call. An example where this may be useful is when you want to apply the same exact function to multiple entities, as is the case when performing a stock split:

```
positions.updateAll(
        Filters.equal(Position::getSymbol, "AAPL"),
        Position::split, 5);
```

In the above code example:

- The first argument is the `Filter` used to determine the set of positions to update.
- The second argument is the function to apply to each position; in this case `split(5)` will be called on each `Position` entity with the `AAPL` symbol.

As with single-entity updates, the result of each function invocation will be returned to the client, this time in the form of a `Map` containing the identifiers of the processed entities as keys, and the result of the function applied to that entity as the value.

Using the Stream API and the Data Aggregation API

You can query the repository to retrieve a subset of entities using a `getAll` method and a `Filter`, but sometimes you do not need the entities themselves, but a result of some computation applied to a subset of entities in the repository.For example, you may need to calculate the average salary of all the employees in a department, or the total value of all equity positions in a portfolio.
While you could certainly query the repository for the entities that need to be processed and perform processing itself on the client, this is a very inefficient way to accomplish the task because you may end up moving significant amount of data over the network, just to discard it after the client-side processing.

Coherence provides a number of features that allow you to perform various types of distributed processing efficiently. Just as the in-place updates leverage Coherence Entry Processor API to perform data mutation on cluster members that store the data, Repository API's support for data aggregation leverages Coherence Remote Stream API and the Aggregation API to perform read-only distributed computations efficiently. This feature allows you to move processing to the data, instead of the other way around, and to perform computation in parallel across as many CPU cores as your cluster has, instead of a handful of (or in many cases only one) cores on the client.

The first option is to use the Stream API, which you are probably already familiar with because it is a standard Java API introduced in Java 8. For example, you can calculate the average salary of all employees as follows:

```
double avgSalary = employees.stream()
         .collect(RemoteCollectors.averagingDouble(Employee::getSalary));
```

If you want to calculate the average salary only for the employees in a specific department instead, you could filter the names of employees to process:

```
double avgSalary = employees.stream()
         .filter(e -> e.getDepartmentId == departmentId)
         .collect(RemoteCollectors.averagingDouble(Employee::getSalary));
```

However, while it works, the code above is not ideal because it will process and potentially deserialize the names of all the employees in the repository to determine whether they belong to a specified department.

A better way to accomplish the same task is to use Coherence-specific `stream` method overload which allows you to specify the `Filter` to create a stream based on:

```
double avgSalary = employees.stream(Filters.equal(Employee::getDepartmentId,
departmentId))
         .collect(RemoteCollectors.averagingDouble(Employee::getSalary));
```

The difference is subtle, but important. Unlike previous example, this example allows Coherence to perform query before creating the stream, and leverage any indexes you may have in the process. This can significantly reduce the overhead when dealing with large data sets.

However, there is also an easier way to accomplish the same thing:

```
double avgSalary = employees.average(Employee::getSalary);
```

or, for a specific department:

```
double avgSalary = employees.average(
        Filters.equal(Employee::getDepartmentId, departmentId),
        Employee::getSalary);
```

These are the examples of using the repository aggregation methods directly, which make the common tasks such as finding `min`, `max`, `average`, and `sum` of any entity attribute simple.

There are also more advanced aggregations, such as `groupBy` and `top`:

```
Map<Gender, Set<Person>> peopleByGender = people.groupBy(Person::getGender);

Map<Long, Double> avgSalaryByDept =
    employees.groupBy(Employee::getDepartmentId,
averagingDouble(Employee::getSalary));

List<Double> top5salaries = employees.top(Employee::getSalary, 5);
```

The simpler ones are `count` and `distinct`.

In many cases, you may not only require the `min`, `max`, or `top` values of an attribute, but also want the entities to which these values belong. For such situations, you can use the `minBy`, `maxBy`, and `topBy` methods, which return the entities containing the minimum, maximum and top values of an attribute, respectively:

```
Optional<Person> oldestPerson   = people.maxBy(Person::getAge);
Optional<Person> youngestPerson = people.minBy(Person::getAge);

List<Employee> highestPaidEmployees = employees.topBy(Employee::getSalary, 5);
```

- Using Declarative Acceleration and Index Creation

Using Declarative Acceleration and Index Creation

Coherence uses indexes to optimize queries and aggregations. The indexes allow you to avoid deserializing entities stored across the cluster, which is an expensive operation when you have large data sets with complex entity classes. The indexes themselves can also be sorted, which is helpful when executing range-based queries, such as `less`, `greater`, or `between`.

The standard way to create indexes is by calling the `NamedMap.addIndex` method. However, the Repository API introduces a simpler, declarative way of index creation.

To define an index, annotate the accessor for the entity attribute(s) for which you want to create an index, with the `@Indexed` annotation:

```
public class Person
    {
    //defines an unordered index on 'Person::getName', which is suitable for
filters such as 'equal', 'like', and 'regex'
    @Indexed
    public String getName()
        {
        return name;
        }
    //defines an ordered index on 'Person::getAge', which is better suited
for filters such as 'less', 'greater', and 'between'
    @Indexed(ordered = true)
    public int getAge()
        {
        return age;
        }
    }
```

When the repository is created, it will introspect the entity class for the `@Indexed` annotation and automatically create an index for each attribute that has this annotation. The created index will then be used whenever that attribute is referenced within the query expression.

In some cases, you may want to keep the deserialized entity instances instead of discarding them. Retaining the instances can be beneficial if you make frequent queries, aggregations, use the Stream API, or make in-place updates or projections because the cost of maintaining individual indexes on all the attributes may be greater than to retain the deserialized entity instances.

For situations such as these, Coherence provides a special index type: `DeserializationAccelerator`. However, if you are use the Repository API, you have an easier way of configuring. Annotate either the entity class, or the repository class itself with the `@Accelerated` annotation:

```
@Accelerated
public class Person
    {
    }
```

You will require additional storage capacity in the cluster to store both the serialized and deserialized copy of all the entities, but in some situations the performance benefits can significantly outweigh the cost. In other words, acceleration is a classic example of a time–space tradeoff, and it is entirely up to you to decide when it makes sense to use it.

Creating Event Listeners

Coherence not only allows you to store, modify, query, and aggregate your data entities efficiently, but also register to receive event notifications whenever any entity in the repository changes.

You can create and register a listener that will be notified whenever an entity is inserted, updated, or removed:

```
public static class PeopleListener
          implements PeopleRepository.Listener<Person>
        {
        public void onInserted(Person personNew)
            {
            // handle INSERT event
            }

        public void onUpdated(Person personOld, Person personNew)
            {
            // handle UPDATE event
            }

        public void onRemoved(Person personOld)
            {
            // handle REMOVE event
            }
        }


people.addListener(new PeopleListener());
people.addListener("111-22-3333", new PeopleListener());
people.addListener(Filters.greater(Person::getAge, 17), new PeopleListener());
```

Where:

- `people.addListener` registers a listener that will be notified whenever any entity in the repository is inserted, updated, or removed.
- The second line registers a listener that will be notified when an entity with the specified identifier is inserted, updated, or removed.

- The third line registers a listener that will be notified when any `Person` older than 17 is inserted, updated, or removed.

As shown in the example above, there are several ways to register only for the events you are interested in, to reduce the number of events received and the amount of data sent over the network.

> **Note:**
>
> All of the listener methods used in the above example have a default no-op implementation. Therefore, you only need to implement the ones you actually want to handle.

However, having to implement a separate class each time you want to register a listener is a bit cumbersome, so the Repository API also provides a default listener implementation, and a fluent builder for it that make the task easier:

```
people.addListener(
        people.listener()
              .onInsert(personNew -> { /* handle INSERT event */ })
              .onUpdate((personOld, personNew) -> { /* handle UPDATE event
with old value */ })
              .onUpdate(personNew -> { /* handle UPDATE event without old
value */ })
              .onRemove(personOld -> { /* handle REMOVE event */ })
              .build()
);
```

> **Note:**
>
> When using the Listener Builder API, you have the option of omitting the old entity value from the `onUpdate` event handler arguments list. You can also specify multiple handlers for the same event type, in which case they will be composed and invoked in the specified order.

There is also an option of providing a single event handler that will receive all the events, regardless of the event type:

```
people.addListener(
        people.listener()
              .onEvent(person -> { /* handle all events */ })
              .build()
);
```

Just as when implementing the listener class explicitly, you can pass the entity identifier or a Filter as the first argument to the `addListener` method to limit the scope of the events received.

Using the Asynchronous Repository API

In addition to the synchronous repository, `AbstractRepository<ID, T>`, there is an asynchronous version: `AbstractAsyncRepository<ID, T>`. The same abstract methods as described in Implementing a Repository, should be implemented.
The main differences between the two APIs is that the asynchronous API returns `java.util.CompletableFuture` of the return type. For example, `Collection<T> getAll()` in the blocking version would be `CompletableFuture<Collection<T>>` in the asynchronous version of the Repository API.

The asynchronous API also offers call backs to which the results of the operation will be passed, as they become available, instead of buffering the result into a collection prior to returning. This feature enables you to stream and process very large result sets without paying the cost of accumulating all the results in memory, which is not possible with the blocking API.

Example 40-1 AbstractAsyncRepository Examples

```
public class AsyncPeopleRepository
        extends AbstractAsyncRepository<String, Person>
    {
    private AsyncNamedMap<String, Person> people;

    public AsyncPeopleRepository(AsyncNamedMap<String, Person> people)
        {
        this.people = people;
        }

    protected AsyncNamedMap<String, Person> getMap()
        {
        return people;
        }

    protected String getId(Person entity)
        {
        return entity.getSsn();
        }

    protected Class<? extends Person> getEntityType()
        {
        return Person.class;
        }
    }
```

- The `getMap` method returns `AsyncNamedMap` that should be used as a backing data store for the repository, which is in this case, provided through the constructor argument. However, it can also be easily be injected through CDI (Contexts and Dependency Injection).
- The `getId` method returns an identifier for a given entity.
- The `getEntityType` method returns the class of the entities stored in the repository.

The following example uses `AsyncPersonRepository` to make a simple query for an entity:

```
String upercaseName = asyncPeople.get("111-22-3333")
                                 .thenApply(Person::getName)
```

```
.thenApply(String::toUpperCase)
 .get()
```

Where:

- The first line gets a `CompletableFuture<Person` based on the ID.
- When the future is completed, the second line obtains the person's name from the `Person` instance.
- The third line converts the name to uppercase.
- The third line blocks and returns the upper-cased name.

This usage pattern will be similar across all the methods that return `CompletableFuture`.

- Using Asynchronous Callbacks

Using Asynchronous Callbacks

Instead of dealing with an entire collection being realized for the results, it is possible to define a callback that will be invoked as results become available. These APIs will return `CompletableFuture<Void>` to signal that all the results have been processed.

For example, if you want to print the names of people as they are streamed back from the server, without accumulating result set on the client, do the following:

```
asyncPeople.getAll(person -> System.out.println(person.getName()))
    .thenApply(done -> System.out.println("DONE!"))
```

In the above code example:

- The first line of code prints the name of each `Person` within the repository.
- The second line prints `DONE!` when the names of all the people have been processed.

You can also extract the name attribute by providing `ValueExtractor` for it as the first argument. In this case, to move less data over the network, you can rewrite the code in the above example as follows:

```
asyncPeople.getAll(Person::getName, (id, name) -> System.out.println(name))
     .thenApply(done -> System.out.println("DONE!"))
```

Where:

- The first line of code prints the name of each `Person` within the repository.
- The second line prints `DONE!` when the names of all the people have been processed.

In the example above, the callback is implemented as a `BiConsumer` that receives the entity identifier and the extracted value as arguments. You can also use the fragment extractor as the first argument to the `getAll` method above, in which case the second argument to the callback will be `Fragment<Person>` instead of just the name attribute.

41

Implementing Concurrency in a Distributed Environment

The Coherence Concurrent module provides distributed implementations of the concurrency primitives from the `java.util.concurrent` package such as executors, atomics, locks, semaphores, and latches.

You can implement concurrent applications using the constructs you are already familiar with and also expand the "scope" of concurrency from a single process to potentially hundreds of processes within a Coherence cluster. You can use executors to submit tasks to be executed somewhere in the cluster; you can use locks, latches, and semaphores to synchronize execution across many cluster members; you can use atomics to implement global counters across many processes, and so on.

While these features are extremely powerful and enable you to reuse the knowledge you already have, they may have detrimental effect on scalability and/or performance. Whenever you synchronize execution through locks, latches, or semaphores, you are introducing a potential bottleneck into the architecture. Whenever you use a distributed atomic to implement a global counter, you are turning very simple operations that take mere nanoseconds locally, such as increment and decrement, into fairly expensive network calls that could take milliseconds (and potentially block even longer under heavy load).

So, use these features sparingly. In many cases, there is a better, faster, and a more scalable way to accomplish the same goal using Coherence primitives such as entry processors, aggregators, and events. These primitives are designed to perform and scale well in a distributed environment.

> **Note:**
>
> - To use the concurrency features, Oracle recommends using the Bootstrap API to start the Coherence cluster members. See Using the Bootstrap API.
> - Coherence concurrent features do not support, and cannot be configured to use federation. Coherence Federation is asynchronous. Therefore, it would not make sense to federate data that is inherently atomic in nature.

This chapter includes the following sections:

- Using Factory Classes
  Each feature (executors, atomics, locks, semaphores, and latches) is backed by one or more Coherence caches, possibly with preconfigured interceptors. All interaction with lower level Coherence primitives is hidden behind various factory classes that allow you to get the instances of the classes you need.
- Using Local and Remote Instances
  In many cases, the factory classes allow you to get both the local and the remote instances of various constructs. For example, `Locks.localLock` will give you an instance of a standard `java.util.concurrent.locks.ReentrantLock`, while `Locks.remoteLock` will return an instance of a `RemoteLock`.

- Using Serialization
  Coherence Concurrent supports both Java serialization and POF out-of-the-box serialization, with Java serialization being the default.
- Using Persistence
  Coherence Concurrent supports both active and on-demand persistence, but just like in the rest of Coherence it is set to `on-demand` by default.
- Using the Coherence Concurrent Features
  You can use the Coherence Concurrent features after declaring the features as a dependency in the `pom.xml` file.
- Using Executors
- Using Atomics
- Using Locks
  Coherence Concurrent provides distributed implementations of `Lock` and `ReadWriteLock` interfaces from the `java.util.concurrent.locks` package, enabling you to implement lock-based concurrency control across cluster members when necessary.
- Using Latches and Semaphores
- Using Blocking Queues

Using Factory Classes

Each feature (executors, atomics, locks, semaphores, and latches) is backed by one or more Coherence caches, possibly with preconfigured interceptors. All interaction with lower level Coherence primitives is hidden behind various factory classes that allow you to get the instances of the classes you need.

For example, you will use factory methods within the `Atomics` class to get instances of various atomic types, `Locks` to get lock instances, `Latches` and `Semaphores` to get latches and semaphores.

Using Local and Remote Instances

In many cases, the factory classes allow you to get both the local and the remote instances of various constructs. For example, `Locks.localLock` will give you an instance of a standard `java.util.concurrent.locks.ReentrantLock`, while `Locks.remoteLock` will return an instance of a `RemoteLock`.

In cases where JDK does not provide a standard interface, which is the case with atomics, latches, and semaphores, the interface from the existing JDK class has been extracted to create a thin wrapper around the corresponding JDK implementation. For example, Coherence Concurrent provides a `Semaphore` interface and a `LocalSemaphore` class that wraps `java.util.concurrent.Semaphore`. The same is true for `CountDownLatch` and all atomic types.

The main advantage of using factory classes to construct both the local and the remote instances is that it allows you to name local locks the same way you name the remote locks: calling `Locks.localLock("foo")` always returns the same `Lock` instance because the `Locks` class internally caches both the local and the remote instances it creates. In the case of remote locks, every locally cached remote lock instance is ultimately backed by a shared lock instance somewhere in the cluster, which is used to synchronize lock state across the processes.

Using Serialization

Coherence Concurrent supports both Java serialization and POF out-of-the-box serialization, with Java serialization being the default.

If you want to use POF instead, you have to set the `coherence.concurrent.serializer` system property to `pof`. You should also include the `coherence-concurrent-pof-config.xml` file into your own POF configuration file to register the built-in Coherence Concurrent types.

Using Persistence

Coherence Concurrent supports both active and on-demand persistence, but just like in the rest of Coherence it is set to `on-demand` by default.

To use active persistence, you should set the `coherence.concurrent.persistence.environment` system property to `default-active`, or use another persistence environment that has active persistence enabled.

> **Note:**
>
> The caches that store lock and semaphore data are configured as transient, and are not persisted when you use active or on-demand persistence.

Using the Coherence Concurrent Features

You can use the Coherence Concurrent features after declaring the features as a dependency in the `pom.xml` file.

To declare, make the following entry in the `pom.xml` file.

```
<dependency>
    <groupId>${coherence.groupId}</groupId>
    <artifactId>coherence-concurrent</artifactId>
    <version>${coherence.version}</version>
</dependency>
```

Using Executors

Coherence Concurrent provides a facility to dispatch tasks, either a `Runnable`, `Callable`, or `Task` to a Coherence cluster for execution. Executors that will run the submitted tasks are configured on each cluster member by defining one or more named executors within a cache configuration resource.
This section includes the following topics:

- Using Executors - Examples
- Advanced Orchestration
- Advanced Orchestration - Examples
- Configuring Executors

- Managing Executors
- Managing Executors Over REST
- Using CDI

Using Executors - Examples

By default, each Coherence cluster with the `coherence-concurrent` module on the classpath includes a single-threaded executor that may be used to run the dispatched tasks.

Given this, the simplest example would be:

```
RemoteExecutor remoteExecutor = RemoteExecutor.getDefault();

Future<Void> result = remoteExecutor.submit(() ->
System.out.println("Executed"));

result.get(); // block until completion
```

If an executor has been configured with the name of `Fixed5`, then a reference to the executor may be obtained with::

```
RemoteExecutor remoteExecutor = RemoteExecutor.get("Fixed5");
```

If no executor has been configured with the given name, the `RemoteExecutor` will throw the following exception:

```
RejectedExecutionException
```

Each `RemoteExecutor` instance may hold local resources that should be released when the `RemoteExecutor` is no longer required. Like an `ExecutorService`, a `RemoteExecutor` has similar methods to shut down the executor. Calling these methods has no impact on the executors registered within the cluster.

Advanced Orchestration

While the `RemoteExecutor` does provide functionality similar to the standard `ExecutorService` included in the JDK, this may not be enough in the context of Coherence. A task might need to run across multiple Coherence members, produce intermediate results, and remain durable in case a cluster member running the task fails.

In such cases, task orchestration can be used. Before diving into the details of orchestration, the following concepts should be understood.

Table 41-1 Task Orchestration Interfaces

Interface	Description
`Task`	Tasks are like `Callable` and `Runnable` classes in that they are designed to be potentially run by one or more threads. Unlike `Callable` and `Runnable` classes, their execution may occur in different Java Virtual Machines, fail and/or recover between different Java Virtual Machine processes.
`Task.Context`	Provides contextual information for a `Task` as it is executed, including the ability to access and update intermediate results for the `Executor` executing the said `Task`.

Table 41-1 (Cont.) Task Orchestration Interfaces

Interface	Description
`Task.Orchestration`	Defines information concerning the orchestration of a `Task` across a set of executors, defined across multiple Coherence members for a given `RemoteExecutor`.
`Task.Coordinator`	A publisher of collected `Task` results that additionally permits coordination of the submitted `Task`.
`Task.Subscriber`	A receiver of items produced by a `Task.Coordinator`.
`Task.Properties`	State sharing mechanism for tasks.
`Task.Collector`	A mutable reduction operation that accumulates results into a mutable result container, optionally transforming the accumulated result into a final representation after all results have been processed.

Tasks

`Task` implementations define a single method called `execute(Context)` that performs the task, possibly yielding execution to some later point. After the method has completed execution, by returning a result or throwing an exception (but not a `Yield` exception), the task is considered completed for the assigned `Executor`.

A `Task` may yield execution for a given time by throwing a `Yield` exception. This exception type signals the execution of a `Task` by an `Executor` is to be suspended and resumed at some later point in time, typically by the same `Executor`.

Task Context

When a `Task` is executed, a `Context` instance will be passed as an execution argument.

The `Context` provides access to task properties allowing shared state between tasks running in multiple Java Virtual Machines.

The `Context` provides details on overall execution status.

Table 41-2 Execution Status

Execution State	Method	Description
`Complete`	`Context.isDone()`	Allows a `Task` to determine if the task is complete. Completion may be due to normal termination, an exception, or cancellation. In all of these cases, this method will return `true`.

Table 41-2 (Cont.) Execution Status

Execution State	Method	Description
Cancelled	`Context.isCancelled()`	Allows a `Task` to determine if the task is effectively cancelled.
Resuming	`Context.isResuming()`	Determines if a `Task` execution by an `Executor` is resuming after being recovered (for example, failover) or due to resumption after a task had previously thrown a `Yield` exception.

Task Orchestration

Orchestrations begin by calling `RemoteExecutor.orchestrate(Task)`, which will return a `Task.Orchestration` instance for the given `Task`. With the `Task.Orchestration`, it's possible to configure the aspects of where the task will be run.

Table 41-3 Task Orchestration Methods

Method	Description
`concurrently()`	Tasks will be run, concurrently, across all Java Virtual Machines, where the named executor is defined/configured. This is the default.
`sequentially()`	Tasks will be run, in sequence, across all Java Virtual Machines, where the named executor is defined/configured.
`limit(int)`	Limit the task to `n` executors. Use this to limit the number of executors that will be considered for task execution. If not set, the default behavior is to run the task on all Java Virtual Machines where the named executor is defined/configured.
`filter(Predicate)`	Filtering provides an additional way to constrain where a task may be run. The predicates will be applied against metadata associated with each executor on each Java Virtual Machine. Some examples of metadata would be the member in which the executor is running, or the role of a member. Predicates may be chained to provide Boolean logic in determining an appropriate executor.
`define(String, <V>)`	Define initial state that will be available to all tasks no matter which Java Virtual Machine that task is running on.
`retain(Duration)`	When specified, the task will be retained allowing new subscribers to be notified of the final result of a task computation after it has completed.
`collect(Collector)`	This is the terminal of the orchestration builder returning a `Task.Collectable`, which defines how results are to be collected and ultimately submits the task to the grid.

Task Collector and Collectable

The `Task.Collector` passed to the orchestration will collect results from tasks and optionally transform the collected results into a final format. Collectors are best illustrated by using examples of Collectors that are available in the `TaskCollector` class.

Table 41-4 Task Collector Methods

Method	Description
`count()`	The count of non-null results that have been collected from the executing tasks.

Table 41-4 (Cont.) Task Collector Methods

Method	Description
`firstOf()`	Collects and returns the first result provided by the executing tasks.
`lastOf()`	Collects and returns the last result returned by the executing tasks.
`setOf()`	Collects and returns all non-null results as a Set.
`listOf()`	Collects and returns all non-null results as a List.

The `Task.Collectable` instance returned by calling `collect` on the orchestration allows, among other things, setting the condition under which no more results will be collected or published by any registered subscribers. Calling `submit()` on the `Task.Collectable` will begin the orchestration of the task.

Task Coordinator

Upon calling `submit()` on the orchestration `Collectable`, a `Task.Coordinator` is returned. Like the `Task.Collectable`, the `Task.Coordinator` allows for the registration of subscribers. Additionally, it provides the ability to cancel or check the completion status of the orchestration.

Task Subscriber

The `Task.Subscriber` receives various events pertaining to the execution status of the orchestration.

Table 41-5 Task Subscriber Events

Method	Description
`onComplete()`	Signals the completion of the orchestration.
`onError(Throwable)`	Called when an unrecoverable error (given as the argument) has occurred.
`onNext(<T>)`	Called when the `Task.Coordinator` has produced a result.
`onSubscribe(Task.Subscription)`	Called prior to any calls to `onComplete()`, `onError(Throwable)`, or `onNext(<T>)` are called. The `Task.Subscription` provided gives access to cancelling the subscription or obtaining a reference to the `Task.Coordinator`.

Advanced Orchestration - Examples

To begin, consider the following code common to orchestration examples:

```
// demonstrate orchestration using the default RemoteExecutor
RemoteExecutor executor = RemoteExecutor.getDefault();

// WaitingSubscriber is an implementation of the
// com.oracle.coherence.concurrent.executor.Task.Subscriber interface
// that has a get() method that blocks until Subscriber.onComplete() is
// called and will return the results received by onNext()
WaitingSubscriber subscriber = new WaitingSubscriber();

// ValueTask is an implementation of the
// com.oracle.coherence.concurrent.executor.Task interface
```

```
// that returns the value provided at construction time
ValueTask task = new ValueTask("Hello World");
```

Given the previous example, the simplest example of an orchestration is:

```
// orchestrate the task, subscribe, and submit
executor.orchestrate(task)
        .subscribe(subscriber)
        .submit();

// wait for the task to complete
// if this was run on four cluster members running the default executor service,
// the returned Collection will have four results
Collection<String> results = subscriber.get();
```

Building on the previous example, assume a cluster with two storage and two proxy members. The cluster members are configured with the roles of `storage` and `proxy`, respectively. Let's say the task needs to run on `storage` members only, then the orchestration could look like:

```
// orchestrate the task, filtering by a role, subscribe, and submit
executor.orchestrate(task)
        .filter(Predicates.role("storage"))
        .subscribe(subscriber)
        .submit();

// wait for the task to complete
// as there are only two storage members in this hypothetical, only two
// results will be returned
Collection<String> results = subscriber.get();
```

There are several predicates available for use in `com.oracle.coherence.concurrent.executor.function.Predicates`, however, in case none apply to the target use case, simply implement the `Remote.Predicate` interface.

You can customize the collection of results and how they are presented to the subscriber by using `collect(Collector)` and `until(Predicate)`:

```
// orchestrate the task, collecting the first non-null result,
// subscribe, and submit
executor.orchestrate(new MayReturnNullTask())
        .collect(TaskCollectors.firstOf())
        .until(Predicates.nonNullValue())
        .subscribe(subscriber)
        .submit();

// wait for the task to complete
// the first non-result returned will be the one provided to the
// subscriber
Collection<String> results = subscriber.get();
```

Several collectors are provided in `com.oracle.coherence.concurrent.executor.TaskCollectors`, however, in case none apply to the target use case, implement the `Task.Collector` interface.

Configuring Executors

Several executor types are available for configuration.

Table 41-6 Types of Executors

ExecutorService Type	Description
Single thread	Creates an `ExecutorService` with a single thread.
Fixed thread	Creates an `ExecutorService` with a fixed number of threads.
Cached	Creates an `ExecutorService` that creates new threads as needed and reuses existing threads when possible.
Work stealing	Creates a work-stealing thread pool by using the number of available processors as its target parallelism level.
Custom	Allows the creation of non-standard executors.
VirtualThread	Creates a VirtualThread-per-task ExecutorService. Requires JDK 21 or later.

Table 41-7 Configuration Elements

Element Name	Required	Expected Type	Description
`single`	no	N/A	Defines a single-thread executor.
`fixed`	no	N/A	Defines a fixed-thread pool executor.
`cached`	no	N/A	Defines a cached-thread-pool executor
`work-stealing`	no	N/A	Defines a work-stealing pool executor.
`custom-executor`	no	`java.util.concurrent.ExecutorService`	Defines a custom executor.
`virtual-per-task`	no	N/A	Defines a VirtualThread-per-task executor.
`name`	yes	`java.lang.String`	Defines the logical `name` of the executor.
`thread-count`	yes	`java.lang.Integer`	Defines the thread count for a fixed-thread pool executor.
`parallelism`	no	`java.lang.Integer`	Defines the parallelism of a work-stealing-thread pool executor. If not defined, it defaults to the number of processors available on the system.
`thread-factory`	no	N/A	Defines a `java.util.concurrent.ThreadFactory`. Used by `single`, `fixed`, and `cached` executors.
`instance`	yes	`java.util.concurrent.ThreadFactory`	Defines how the `ThreadFactory` will be instantiated. For information about the instance element, see instance. This element must be a child of the `thread-factory` element.

For complete details, see schema.

Configuration Examples

To define executors, the `cache-config` root element should include the `coherence-concurrent` NamespaceHandler to recognize the configuration elements.

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

> **Note:**
>
> Executors defined by the configuration must precede any other elements in the document. Failing to do so, will prevent the document from being validated.

The following examples assume that the `xml` namespace defined for the NamespaceHandler is `c`:

```
<!-- creates a single-threaded executor named <em>Single</em> -->
<c:single>
  <c:name>Single</c:name>
</c:single>


<!-- creates a single-threaded executor named <em>Single</em> with a thread
factory-->
<c:single>
  <c:name>SingleTF</c:name>
  <c:thread-factory>
    <c:instance>
      <c:class-name>my.custom.ThreadFactory</c:class-name>
    </c:instance>
  </c:thread-factory>
</c:single>


<!-- creates a fixed-thread executor named <em>Fixed5</em> -->
<c:fixed>
  <c:name>Single</c:name>
  <c:thread-count>5</c:thread-count>
</c:fixed>
```

Managing Executors

There are various ways to manage and monitor executors. You can use one of the following options:

- MBeans: See ExecutorMBean.
- Reporter: See Understanding the Executor Report.
- Coherence VisualVM plug-in: See coherence-visualvm.
- REST API: See Managing Executors Over REST.
- Distributed Tracing: See Distributed Tracing .

Managing Executors Over REST

Coherence Management over REST exposes endpoints to query and invoke actions against `ExecutorMBean` instances.

Table 41-8 REST Endpoints

Description	Method	Path	Produces
View all Executors	`GET`	`/management/coherence/cluster/executors`	JSON
View all Executors with matching name	`GET`	`/management/coherence/cluster/executors/{name}`	JSON
Reset Executor statistics by name	`POST`	`/management/coherence/cluster/executors/{name}/resetStatistics`	JSON

Using CDI

You can inject RemoteExecutors through CDI.

For example:

```
@Inject
private RemoteExecutor single;    // injects a RemoteExecutor named 'single'.

@Inject
@Name("Fixed5")
private RemoteExecutor fixedPoolRemoteExecutor;    // injects a
RemoteExecutor named 'Fixed5'.
```

Using Atomics

Coherence Concurrent provides distributed implementations of atomic types, such as `AtomicInteger`, `AtomicLong`, and `AtomicReference`. It also provides local implementations of the same types. The local implementations are just thin wrappers around the existing `java.util.concurrent.atomic` types, which implement the same interface as their distributed variants, to be interchangeable.
To create instances of atomic types, you should call the appropriate factory method on the `Atomics` class:

```
// Creates a local, in-process instance of named 'AtomicInteger' with an
implicit initial value of 0.
AtomicInteger localFoo  = Atomics.localAtomicInteger("foo");

// Creates a remote, distributed instance of named 'AtomicInteger', distinct
from the local instance 'foo',
// with an implicit initial value of '0'.
AtomicInteger remoteFoo = Atomics.remoteAtomicInteger("foo");

// Creates a remote, distributed instance of named 'AtomicLong', with an
initial value of '5'.
AtomicLong    remoteBar = Atomics.remoteAtomicLong("bar", 5L);
```

> **Note:**
>
> The `AtomicInteger` and `AtomicLong` types used in the code above are not types from the `java.util.concurrent.atomic` package. They are actually interfaces defined within the `com.oracle.coherence.concurrent.atomic` package that both `LocalAtomicXyz` and `RemoteAtomicXyz` classes implement, which are the instances that are actually returned by the above methods.

Therefore, you can rewrite the above code as:

```
LocalAtomicInteger  localFoo  = Atomics.localAtomicInteger("foo");
RemoteAtomicInteger remoteFoo = Atomics.remoteAtomicInteger("foo");
RemoteAtomicLong    remoteBar = Atomics.remoteAtomicLong("bar", 5L);
```

However, Oracle strongly recommends that you use interfaces instead of concrete types because interfaces make it easy to switch between local and distributed implementations when necessary.

After the instances are created, you can use them the same way you would use any of the corresponding `java.util.concurrent.atomic` types:

```
int  counter1 = remoteFoo.incrementAndGet();
long counter5 = remoteBar.addAndGet(5L);
```

This section includes the following topics:

- Asynchronous Implementations of Atomic Types

- Using CDI

Asynchronous Implementations of Atomic Types

The instances of numeric atomic types, such as `AtomicInteger` and `AtomicLong`, are frequently used to represent various counters in the application where a client may need to increment the value, but does not necessarily need to know what the new value is.

When working with the local atomics, you can use the same API shown earlier (see Using Atomics) and simply ignore the return value. However, when using distributed atomics that would introduce unnecessary blocking on the client while waiting for the response from the server, which would then simply be discarded. Obviously, this will have a negative impact on both performance and throughput of the atomics.

To reduce the impact of remote calls in those situations, Coherence Concurrent also provides non-blocking, asynchronous implementations of all atomic types it supports.

To obtain a non-blocking instance of any supported atomic type, simply call the `async` method on the blocking instance of that type:

```
// Creates a remote, distributed instance of named, non-blocking
'AsyncAtomicInteger', with an implicit initial value of 0.
AsyncAtomicInteger asyncFoo =
Atomics.remoteAtomicInteger("foo").async();

// Creates a remote, distributed instance of named, non-blocking
'AsyncAtomicLong', with an initial value of 5.
AsyncAtomicLong    asyncBar = Atomics.remoteAtomicLong("bar", 5L).async();
```

After you create these instances, you can use them the same way you would use any of the corresponding blocking types. The only difference is that the non-blocking instances will simply return a `CompletableFuture` for the result, and will not block:

```
CompletableFuture<Integer> futureCounter1 = asyncFoo.incrementAndGet();
CompletableFuture<Long>    futureCounter5 = asyncBar.addAndGet(5L);
```

Both the blocking and non-blocking instances of any distributed atomic type, with the same name, are backed by the same cluster-side atomic instance state, so they can be used interchangeably.

Using CDI

Atomic types from Coherence Concurrent can also be injected using CDI, which eliminates the need for explicit factory method calls on the Atomics class.

```
// Injects a local, in-process instance of an 'AtomicInteger' named 'foo',
with an implicit initial value of '0'.
@Inject
@Name("foo")
private AtomicInteger localFoo;

// Injects a remote, distributed instance of an 'AtomicInteger' named 'foo',
distinct from
// the local instance 'foo', with an implicit initial value of '0'.
```

```
@Inject
@Remote
@Name("foo")
private AtomicInteger remoteFoo;

// Injects a remote, distributed instance of non-blocking 'AsyncAtomicLong',
with an implicit name of 'asyncBar'.
@Inject
@Remote
private AsyncAtomicLong asyncBar
```

After you obtain an instance of an atomic type through a CDI injection, you can use it the same way you would use an instance obtained directly from the `Atomics` factory class.

Using Locks

Coherence Concurrent provides distributed implementations of `Lock` and `ReadWriteLock` interfaces from the `java.util.concurrent.locks` package, enabling you to implement lock-based concurrency control across cluster members when necessary.

Unlike local JDK implementations, the classes in this package use cluster member/process ID and thread ID to identify lock owner, and store shared lock state within a Coherence `NamedMap`. However, this also implies that the calls to acquire and release locks are remote, network calls, because they need to update shared state that is likely stored on a different cluster member. This update may impact the performance of `lock` and `unlock` operations.

This section includes the following topics:

- Using Exclusive Locks
- Using Read/Write Locks
- Using CDI

Using Exclusive Locks

A `RemoteLock` class provides an implementation of a `Lock` interface and enables you to ensure that only one thread on one member is running a critical section guarded by the lock at any given time.

To obtain an instance of a `RemoteLock`, call the `Locks.remoteLock factory` method:

```
Lock foo = Locks.remoteLock("foo");
```

As seen with `Atomics`, you can obtain a local `Lock` instance from the `Locks` class, which will simply return an instance of a standard `java.util.concurrent.locks.ReentrantLock`, by calling the `localLock` factory method:

```
Lock foo = Locks.localLock("foo");
```

After you create a `Lock` instance, you can use it as you normally would:

```
foo.lock();
try {
    // critical section guarded by the exclusive lock `foo`
```

```
}
finally {
    foo.unlock();
}
```

Using Read/Write Locks

A `RemoteReadWriteLock` class provides an implementation of a `ReadWriteLock` interface and enables you to ensure that only one thread on one member is running a critical section guarded by the `write` lock at any given time, while allowing multiple concurrent readers.

To obtain an instance of a `RemoteReadWriteLock`, call the `Locks.remoteReadWriteLock` factory method:

```
ReadWriteLock bar = Locks.remoteReadWriteLock("bar");
```

As seen with `Atomics`, you can obtain a local `ReadWriteLock` instance from the `Locks` class, which will simply return an instance of a standard `java.util.concurrent.locks.ReentrantReadWriteLock`, by calling the `localReadWriteLock` factory method:

```
ReadWriteLock bar = Locks.localReadWriteLock("bar");
```

After you create a `ReadWriteLock` instance, you can use it as you normally would:

```
bar.writeLock().lock()
try {
    // critical section guarded by the exclusive write lock `bar`
}
finally {
    bar.writeLock().unlock();
}
```

Or:

```
bar.readLock().lock()
try {
    // critical section guarded by the shared read lock `bar`
}
finally {
    bar.readLock().unlock();
}
```

Using CDI

You can also use CDI to inject both the exclusive and read/write lock instances into objects that need them:

```
// Injects distributed exclusive lock named 'foo' into the 'lock' field.
@Inject
@Remote
@Name("foo")
```

```
private Lock lock;

// Injects distributed read/write lock named 'bar' into the 'bar' field.
@Inject
@Remote
@Name("bar")
private ReadWriteLock bar;
```

After you obtain an instance of lock through a CDI injection, you can use it the same way you would use an instance obtained directly from the `Locks` factory class.

Using Latches and Semaphores

Coherence Concurrent also provides distributed implementations of a `CountDownLatch` and `Semaphore` classes from the `java.util.concurrent` package, enabling you to implement synchronization of execution across multiple Coherence cluster members as easily as you can implement it within a single process using the two JDK classes. It also provides interfaces for those two concurrency primitives, that both remote and local implementations conform to.

As seen with `Atomics`, the local implementations are nothing more than thin wrappers around the corresponding JDK classes.

This section includes the following topics:

- Using the Count Down Latch
- Using a Semaphore
- Using CDI

Using the Count Down Latch

The `RemoteCoundDownLatch` class provides a distributed implementation of a `CountDownLatch`, and enables you to ensure that the execution of the code on any cluster member that is waiting for the latch proceeds only when the latch reaches zero. Any cluster member can both wait for a latch and count down.

To obtain an instance of `RemoteCountDownLatch`, call the `Latches.remoteCountDownLatch` factory method:

```
// Creates an instance of a 'RemoteCountDownLatch' with the initial count of
'5'.
CoundDownLatch foo = Latches.remoteCountDownLatch("foo", 5);
```

As seen with `Atomics` and `Locks`, you can obtain a local `CountDownLatch` instance from the `Latches` class by calling the `remoteCountDownLatch` factory method:

```
// Creates an instance of a 'LocalCountDownLatch' with the initial count of
'10'.
CoundDownLatch foo = Latches.localCountDownLatch("foo", 10);
```

After you have a `RemoteCountDownLatch` instance, you can use it as you normally would, by calling the `countDown` and `await` methods on it.

Using a Semaphore

The `RemoteSemaphore` class provides a distributed implementation of a `Semaphore`, and enables any cluster member to acquire and release permits from the same `semaphore` instance.

To obtain an instance of `RemoteSemaphore`, call the `Semaphores.remoteSemaphore` factory method:

```
// Creates an instance of a 'RemoteSemaphore' with '5' permits.
Semaphore foo = Semaphores.remoteSemaphore("foo", 5);
```

As seen with `Atomics` and `Locks`, you can obtain a local `Semaphore` instance from the `Semaphores` class by calling the `localSemaphore` factory method:

```
// Creates an instance of a 'LocalSemaphore' with '0' permits.
Semaphore foo = Semaphores.localSemaphore("foo");
```

After you create a `Semaphore` instance, you can use it as you normally would, by calling the `release` and `acquire` methods on it.

Using CDI

You can also use CDI to inject both the `CountDownLatch` and `Semaphore` instances into objects that need them:

```
// Injects an instance of 'LocalCountDownLatch' with the initial count of '5'.
@Inject
@Name("foo")
@Count(5)
private CountDownLatch localLatchFoo;

// Injects an instance of 'RemoteCountDownLatch' with the initial count of
'10'.
@Inject
@Name("foo")
@Remote
@Count(10)
private CountDownLatch remoteLatchFoo;

// Inject an instance of 'LocalSemaphore' with '0' (zero) permits available.
@Inject
@Name("bar")
@Remote
private Semaphore localSemaphoreBar;

// Inject an instance of 'RemoteSemaphore' with '1' permit available.
@Inject
@Name("bar")
@Remote
@Permits(1)
private Semaphore remoteSemaphoreBar;
```

After you obtain a `latch` or a `semaphore` instance through a CDI injection, you can use the same way as you would use an instance obtained directly from the `Latches` or `Semaphores` factory classes.

The `@Name` annotation is optional in both cases as long as the member name (in the examples above, the field name) can be obtained from the injection point, but is required otherwise (such as when you use a constructor injection).

The `@Count` annotation specifies the initial latch count, and if omitted, will default to one. The `@Permits` annotation specifies the number of available permits for a semaphore, and if omitted, will default to zero, which means that the first `acquire` call will block until another thread releases one or more permits.

Using Blocking Queues

Coherence supports Queues as data structure from Coherence CE 24.03. The Coherence `NamedQueue` is an implementation of `java.util.Queue` and `NamedDeque` is an implementation of `java.util.Deque`.

Coherence has two implementations of `BlockingQueue`: one is a simple size limited queue, the second is a distributed paged queue that has a much larger capacity. The simple queue is available as both a `BlockingQueue` and a double-ended `BlockingDeque`. The distributed paged queue is available only as a `BlockingQueue` implementation.

> **Note:**
>
> Coherence queues are mapped to caches, which take the same name as the queue. If a cache is being used for a queue, then the same cache must not be used as a normal data cache.

Blocking Queue

The Coherence Concurrent module contains an implementation of `java.util.concurrent.BlockingQueue` called `NamedBlockingQueue` and an implementation of `java.util.concurrent.BlockingDeque` called `NamedBlockingDeque`.

To use a Coherence blocking queue in your application, you must add a dependency on the `coherence-concurrent` module as follows:

```
<dependency>
    <groupId>com.oracle.coherence</groupId>
    <artifactId>coherence-concurrent</artifactId>
    <version>14.1.2-0-0</version>
</dependency>
```

To obtain an instance of a blocking queue use the `com.oracle.coherence.concurrent.Queues` factory class.

To obtain a simple size limited `BlockingQueue` named "my-queue", see the following example:

```
NamedBlockingQueue<String> queue = Queues.queue("my-queue");
```

To obtain a simple size limited `BlockingDeque` named "my-deque", see the following example:

```
NamedBlockingDeque<String> queue = Queues.deque("my-deque");
```

To obtain a distributed paged `BlockingQueue` named "my-queue", see the following example:

```
NamedBlockingQueue<String> queue = Queues.pagedQueue("my-queue");
```

The blocking queue implementations work by using Coherence events. When application code calls a blocking method, the calling thread is blocked but the blocking is not on the server. The application code is unblocked when it receives an event from the server.

For example, if the application code calls the `NamedBlockingQueue take()` method and the queue is empty, this method blocks the calling thread. When an element is put into the queue by another thread (maybe on another JVM) the calling application receives an event. This will retry the `take()` and if successful it returns. If the retry of the `take()` is unsuccessful the calling thread remains blocked. For example, another thread or another JVM was also blocked taking from the same queue and managed to get it retry in the first attempt.

Another example is an application calling the `NamedBlockingQueue put()` method, which gets blocked when the queue is full (2GB size limit). In this case, the calling thread is blocked until a delete event is received to signal that there is now space in the queue. The `put()` is retried and if successful is control returned to the calling thread. If the retry is unsuccessful the thread remains blocked. For example, another thread or JVM is also blocked on a `put()` and its retry is succeeded and refills the queue.

Sizing Queues

It is important to understand how the two Coherence queue implementations store data and how this limits the size of a queue.

- Simple Coherence Queue – The simple queue (and deque) implementation stores data in a single Coherence cache partition. This enforces a size limit of 2 GB because a Coherence cache partition should not exceed 2 GB in size, and in reality, a partition should be a lot smaller than this. Large partitions slow down recovery when a storage enabled member leaves the cluster. With a modern fast network, 300 MB – 500 MB should be a suitable maximum partition size; on a 10 GB network, this could even go as high as 1 GB.
- Distributed Paged Queue - The distributed paged queue stores data in pages that are distributed around the Coherence cluster over multiple partitions, the same as normal cache data. This means that the paged queue can store far more than 2 GB. It is still important to be aware of how partition sizes limit the total queue size.

The absolute hard limit of 2 GB per partition gives the following size:

```
2 GB x 257 = 514 GB
```

But this is far too big to be reliable in production use. If you use a size limit of 500 MB and the default partition count of 257, then you can see how this affects queue size.

```
500 MB x 257 = 128 GB
```

So, by default a realistic limit for a paged queue is around 128 GB. If the partition count is increased to 1087, then the queue size becomes:

```
500 MB x 1087 = 543 GB
```

Of course, all these examples assume that there are enough JVMs with big enough heap sizes in the cluster to store the queue data in memory.

Limitations

The current queue implementation in Coherence has the following limitations:

- As previously mentioned, the simple queue has a hard size limit of 2 GB. When using a simple queue or deque, the Coherence server refuses to accept offers to the queue if its size exceeds 2 GB. The `java.util.Queue` contact allows for queues to reject offers, so this size limitation conforms to the queue contract. Application developers should check the response from offering data to the queue to determine whether the offer has succeeded or not. We use the term "offer" here to cover all queue and deque methods that add data to the queue. An alternative to checking the return Boolean from an `offer()` call would be to use a `NamedBlockingQueue` where the `put()` method gets blocked if the queue is full.
- In a normal operation, queues should not get huge as this would usually mean that the processes reading from the queue are not keeping up with the processes writing to the queue. Application developers should obviously load test their applications using queues to ensure that they are not going to have issues with capacity.
- Queue operations such as offering and polling will contend on certain data structures, and this limits the number of parallel requests and the speed at which requests get processed. To maintain ordering, polling contends on either the head or tail entry, depending on which end of the queue is being polled. This means that poll methods can only be processed sequentially, so even though a poll is efficient and fast, many concurrent poll requests will queue and be processed one at a time. Offer methods do not contend on the head or tail but will contend on the atomic counters used to maintain the head and tail identifiers. Coherence can process multiple offer requests on different worker threads but there are minor contentions on the `AtomicLong` updates.
- Queue operations that work on the head and tail, such as offering and polling, are efficient. Some of the other methods in `java.util.Queue` and `java.util.Deque` are less efficient. For example, iterator methods, `contains()`, and so on. These are not frequently used by applications that require basic queue functionality. Some optional methods on the `java.util.Queue` API that mutate the queue will throw `UnsupportedOperationException` (this is allowed by the Java Queue contract), for example, `retainAll()`, `removeAll()`, and removal using an iterator.

42

Using Contexts and Dependency Injection

Coherence provides support for Contexts and Dependency Injection (CDI) within the Coherence cluster members. This feature enables you to inject Coherence-managed resources, such as `NamedMap`, `NamedCache`, and `Session` instances into CDI managed beans; to inject CDI beans into Coherence-managed resources, such as event interceptors and cache stores; and to handle Coherence server-side events using the CDI observer methods.

In addition, Coherence CDI provides support for automatic injection of transient objects upon deserialization. This feature allows you to inject CDI-managed beans, such as services and repositories [to use Domain-Driven Design (DDD) nomenclature], into transient objects, such as entry processor and data class instances, greatly simplifying implementation of domain-driven applications.

This chapter includes the following sections:

- Using CDI
  You should declare Contexts and Dependency Injection (CDI) as a dependency in the `pom.xml` file to enable its usage.
- Injecting Coherence Objects into CDI Beans
  CDI, and dependency injection in general, make it easy for application classes to declare the dependencies they need and let the runtime provide them when necessary. This feature makes the applications easier to develop, test, and reason about; and the code extremely clean.
- Injecting CDI Beans into Coherence-Managed Objects
- Injecting CDI Beans into Transient Objects
  Using CDI to inject Coherence objects into application classes, and CDI beans into Coherence-managed objects enables you to support many use cases where dependency injection may be useful, but it does not cover an important use case that is specific to Coherence.
- Using the FilterBinding Annotations
  When creating views or subscribing to events, you can modify the view or events using `Filters`. The exact `Filter` implementation injected is determined by the view or event observers qualifiers. Specifically, any qualifier annotation that is itself annotated with the `@FilterBinding` annotation.
- Using ExtractorBinding Annotations
  Extractor bindings are annotations that are themselves annotated with `@ExtractorBinding` and are used in conjunction with an implementation of `com.oracle.coherence.cdi.ExtractorFactory` to produce Coherence `ValueExtractor` instances.
- Using MapEventTransformerBinding Annotations
  Coherence CDI supports event observers that can observe events for cache or map entries. You can annotate the `observer` method with a `MapEventTransformerBinding` annotation to indicate that the observer requires a transformer to be applied to the original event before it is observed.

- Using CDI Response Caching
  CDI response caching allows you to apply caching to Java methods transparently. CDI response caching will be enabled after you add the `coherence-cdi` dependency.

Using CDI

You should declare Contexts and Dependency Injection (CDI) as a dependency in the `pom.xml` file to enable its usage.

To declare CDI as a dependency, add the following content in `pom.xml`:

```
<dependency>
    <groupId>${coherence.groupId}</groupId>
    <artifactId>coherence-cdi-server</artifactId>
    <version>${coherence.version}</version>
</dependency>
```

After the necessary dependency is in place, you can start using CDI to inject Coherence objects into managed CDI beans.

> **Note:**
>
> When using CDI, all Coherence resources used by application code should be injected. Applications should avoid calling Coherence APIs that create or initialize Coherence resources directly, especially static `CacheFactory` methods. If application code calls these Coherence APIs, it may cause Coherence to be initialized too early in the start-up process before the CDI framework has initialized the Coherence extensions. A typical symptom of this would be that Coherence starts without picking up the correct configuration from the CDI framework.

Injecting Coherence Objects into CDI Beans

CDI, and dependency injection in general, make it easy for application classes to declare the dependencies they need and let the runtime provide them when necessary. This feature makes the applications easier to develop, test, and reason about; and the code extremely clean.

Coherence CDI allows you to do the same for Coherence objects, such as `Cluster`, `Session`, `NamedMap`, `NamedCache`, `ContinuousQueryCache`, `ConfigurableCacheFactory`, and so on.

This section includes the following topics:

- Injecting NamedMap, NamedCache, and Related Objects
- Injecting NamedTopic and Related Objects
- Supporting Other Injection Points

Injecting NamedMap, NamedCache, and Related Objects

To inject an instance of a `NamedMap` into the CDI bean, you should define an injection point for it:

```
import javax.inject.Inject;

@Inject
private NamedMap<Long, Person> people;
```

In the example above, it is assumed that the map name you want to inject is the same as the name of the field you are injecting into - `people`. If that is not the case, you can use the `@Name` qualifier to specify the name of the map you want to obtain explicitly:

```
import com.oracle.coherence.cdi.Name;
import javax.inject.Inject;

@Inject
@Name("people")
private NamedMap<Long, Person> m_people;
```

This is also what you have to do if you are using constructor injection or setter injection:

```
import com.oracle.coherence.cdi.Name;
import javax.inject.Inject;

@Inject
public MyClass(@Name("people") NamedMap<Long, Person> people) {
    ...
}

@Inject
public void setPeople(@Name("people") NamedMap<Long, Person> people) {
    ...
}
```

All the examples above assume that you want to use the default scope, which is often, but not always the case. For example, you may have an Extend client that connects to multiple Coherence clusters, in which case you would have multiple scopes.

In this case, you would use the `@SessionName` qualifier to specify the name of the configured `Session`, that will be used to supply the cache or map:

```
import com.oracle.coherence.cdi.SessionName;
import javax.inject.Inject;

@Inject
@SessionName("Products")
private NamedCache<Long, Product> products;

@Inject
```

```
@SessionName("Customers")
private NamedCache<Long, Customer> customers;
```

You can replace `NamedMap` or `NamedCache` in any of the examples above with `AsyncNamedMap` and `AsyncNamedCache`, respectively, to inject asynchronous variant of those APIs:

```
import com.oracle.coherence.cdi.SessionName;
import javax.inject.Inject;

@Inject
private AsyncNamedMap<Long, Person> people;

@Inject
@SessionName("Products")
private AsyncNamedCache<Long, Person> Product;
```

This section includes the following topic:

- Injecting NamedMap or NamedCache Views

Injecting NamedMap or NamedCache Views

You can also inject views by adding the `View` qualifier to either `NamedMap` or `NamedCache`:

```
import com.oracle.coherence.cdi.View;
import javax.inject.Inject;

@Inject
@View
private NamedMap<Long, Person> people;

@Inject
@View
private NamedCache<Long, Product> products;
```

The examples above are equivalent, and both will bring all the data from the backing map into a local view, as they will use `AlwaysFilter` when constructing a view. If you want to limit the data in the view to a subset, you can implement a Custom `FilterBinding` (recommended, see Using the FilterBinding Annotations), or use a built-in `@WhereFilter` for convenience, which allows you to specify a filter using CohQL:

```
import com.oracle.coherence.cdi.Name;
import com.oracle.coherence.cdi.View;
import com.oracle.coherence.cdi.WhereFilter;
import javax.inject.Inject;

@Inject
@View
@WhereFilter("gender = 'MALE'")
@Name("people")
private NamedMap<Long, Person> men;

@Inject
@View
```

```
@WhereFilter("gender = 'FEMALE'")
@Name("people")
private NamedMap<Long, Person> women;
```

The **views** also support transformation of the entry values on the server, to reduce both the amount of data stored locally, and the amount of data transferred over the network. For example, you may have complex `Person` objects in the backing map, but only need their names to populate a drop down on the client UI. In that case, you can implement a custom `ExtractorBinding` (recommended, see Creating the Custom Extractor Annotation), or use a built-in `@PropertyExtractor` for convenience:

```
import com.oracle.coherence.cdi.Name;
import com.oracle.coherence.cdi.View;
import com.oracle.coherence.cdi.PropertyExtractor;
import javax.inject.Inject;

@Inject
@View
@PropertyExtractor("fullName")
@Name("people")
private NamedMap<Long, String> names;
```

> **Note:**
>
> The value type in this example has changed from `Person` to `String`, due to the server-side transformation caused by the specified `@PropertyExtractor`.

Injecting NamedTopic and Related Objects

To inject an instance of a `NamedTopic` into the CDI bean, you should define an injection point for it:

```
import com.tangosol.net.NamedTopic;
import javax.inject.Inject;

@Inject
private NamedTopic<Order> orders;
```

In the example above, it is assumed that the topic name you want to inject is the same as the name of the field you are injecting into, in this case `orders`. If that is not the case, you can use the `@Name` qualifier to specify the name of the topic you want to obtain explicitly:

```
import com.oracle.coherence.cdi.Name;
import com.tangosol.net.NamedTopic;
import javax.inject.Inject;

@Inject
@Name("orders")
private NamedTopic<Order> topic;
```

You have to do the following if you are using 'constructor' or 'setter injection' instead:

```
import com.oracle.coherence.cdi.Name;
import com.tangosol.net.NamedTopic;
import javax.inject.Inject;

@Inject
public MyClass(@Name("orders") NamedTopic<Order> orders) {
    ...
}

@Inject
public void setOrdersTopic(@Name("orders") NamedTopic<Order> orders) {
    ...
}
```

All the examples above assume that you want to use the default scope, which is often, but not always the case. For example, you may have an Extend client that connects to multiple Coherence clusters, in which case you would have multiple scopes.

In this case, you would use the `@SessionName` qualifier to specify the name of the configured `Session`, that will be used to supply the topic:

```
import com.oracle.coherence.cdi.SessionName;
import com.tangosol.net.NamedTopic;
import javax.inject.Inject;

@Inject
@SessionName("Finance")
private NamedTopic<PaymentRequest> payments;

@Inject
@SessionName("Shipping")
private NamedTopic<ShippingRequest> shipments;
```

The examples above allow you to inject a `NamedTopic` instance into the CDI bean, but it is often simpler and more convenient to inject `Publisher` or `Subscriber` for a given topic instead.

This can be easily accomplished by replacing `NamedTopic<T>` in any of the examples above with one of the following:

`Publisher<T>`:

```
import com.oracle.coherence.cdi.Name;
import com.oracle.coherence.cdi.SessionName;
import javax.inject.Inject;

@Inject
private Publisher<Order> orders;

@Inject
@Name("orders")
private Publisher<Order> m_orders;

@Inject
```

```
@SessionName("Finance")
private Publisher<PaymentRequest> payments;
```

OR

`Subscriber<T>`:

```
import com.oracle.coherence.cdi.Name;
import com.oracle.coherence.cdi.SessionName;
import javax.inject.Inject;

@Inject
private Subscriber<Order> orders;

@Inject
@Name("orders")
private Subscriber<Order> m_orders;

@Inject
@SessionName("Finance")
private Subscriber<PaymentRequest> payments;
```

Topic metadata, such as topic name (based on either injection point name or the explicit name from `@Name` annotation), scope, and message type, will be used under the hood to retrieve the `NamedTopic`, and to obtain `Publisher` or `Subscriber` from it.

Additionally, if you want to place your Subscribers into a subscriber group (effectively turning a topic into a queue), you can easily accomplish that by adding the `@SubscriberGroup` qualifier to the injection point:

```
import com.oracle.coherence.cdi.SubscriberGroup;
import javax.inject.Inject;

@Inject
@SubscriberGroup("orders-queue")
private Subscriber<Order> orders;
```

Supporting Other Injection Points

While the injection of a `NamedMap`, `NamedCache`, `NamedTopic`, and related instances is probably the single most used feature of Coherence CDI, it is certainly not the only one.

The following sections describe the other Coherence artifacts that can be injected using Coherence CDI:

- Cluster and OperationalContext Injection
- Named Session Injection
- Serializer Injection
- POF Serializer with a Specific POF Configuration

Cluster and OperationalContext Injection

If you need an instance of a `Cluster` interface somewhere in your application, you can easily obtain it through injection:

```
import com.tangosol.net.Cluster;
import javax.inject.Inject;

@Inject
private Cluster cluster;
```

You can do the same if you need an instance of an `OperationalContext`:

```
import com.tangosol.net.OperationalContext;
import javax.inject.Inject;

@Inject
private OperationalContext ctx;
```

Named Session Injection

On rare occasions, when you need to use a `Session` directly, Coherence CDI makes it trivial to do so.

Coherence creates a default `Session` when the CDI server starts. The session is created using the normal default cache configuration file. Other named sessions can be configured as CDI beans of type `SessionConfiguration`.

For example:

```
import com.oracle.coherence.cdi.SessionInitializer;
import javax.enterprise.context.ApplicationScoped;

@ApplicationScoped
public class MySession
        implements SessionInitializer
    {
    public String getName()
        {
        return "Foo";
        }
    // implement session configuration methods
    }
```

The bean above creates the configuration for a `Session` named `Foo`. The session is created when the CDI server starts, and then is injected into other beans. A simpler way to create a `SessionConfiguration` is to implement the `SessionIntializer` interface and annotate the class.

For example:

```
import com.oracle.coherence.cdi.ConfigUri;
import com.oracle.coherence.cdi.Scope;
```

```
import com.oracle.coherence.cdi.SessionInitializer;
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Named;

@ApplicationScoped
@Named("Foo")
@Scope("Foo")
@ConfigUri("my-coherence-config.xml")
public class MySession
        implements SessionInitializer
    {
    }
```

The above configuration creates a `Session` bean with the name `Foo` with an underlying `ConfigurableCacheFactory` created from the `my-coherence-config.xml` configuration file.

To obtain an instance of the default `Session`, all you need to do is inject the session into the class which needs to use it:

```
import com.tangosol.net.Session;
import javax.inject.Inject;

@Inject
private Session session;
```

If you need a specific named `Session`, you can qualify one using the `@Name` qualifier and specifying the `Session` name:

```
import com.oracle.coherence.cdi.Name;
import javax.inject.Inject;

@Inject
@Name("SessionOne")
private Session sessionOne;

@Inject
@Name("SessionTwo")
private Session sessionTwo;
```

Serializer Injection

While in most cases you dont have to deal with serializers directly, Coherence CDI makes it simple to obtain named serializers (and to register new ones) when you need.

To get a default `Serializer` for the current context class loader, you can inject it:

```
import com.tangosol.io.Serializer;
import javax.inject.Inject;

@Inject
private Serializer defaultSerializer;
```

However, it may be more useful to inject one of the named serializers defined in the operational configuration, which can be easily accomplished using the `@Name` qualifier:

```
import com.oracle.coherence.cdi.Name;
import javax.inject.Inject;

@Inject
@Name("java")
private Serializer javaSerializer;

@Inject
@Name("pof")
private Serializer pofSerializer;
```

In addition to the serializers defined in the operational config, this example will also perform `BeanManager` lookup for a named bean that implements the `Serializer` interface. This means that you can implement a custom `Serializer` bean such as the following:

```
import com.tangosol.io.Serializer;
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Named;

@Named("json")
@ApplicationScoped
public class JsonSerializer implements Serializer {
    ...
}
```

The custom `Serializer` bean would be automatically discovered and registered by the CDI, and you would then be able to inject it easily like the named serializers defined in the operational config:

```
import com.oracle.coherence.cdi.Name;
import javax.inject.Inject;

@Inject
@Name("json")
private Serializer jsonSerializer;
```

POF Serializer with a Specific POF Configuration

You can inject POF serializers by using both the `@Name` and `@ConfigUri` qualifiers to inject a POF serializer that uses a specific POF configuration file.

```
import com.oracle.coherence.cdi.ConfigUri;
import com.oracle.coherence.cdi.Name;
import javax.inject.Inject;

@Inject
@Name("pof")
@ConfigUri("test-pof-config.xml")
private Serializer pofSerializer;
```

The code above will inject a POF serializer that uses the `test-pof-config.xml` file as its configuration file.

Injecting CDI Beans into Coherence-Managed Objects

Coherence has many server-side extension points, which enable you to customize application behavior in different ways, typically by configuring their extensions within various sections of the cache configuration file. For example, you can implement event interceptors and cache stores to handle server-side events and integrate with the external data stores and other services.
Coherence CDI provides a way to inject named CDI beans into these extension points using the custom configuration namespace handler.

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

After you have declared the handler for the `cdi` namespace above, you can specify the `<cdi:bean>` element in any place where you would normally use the `<class-name>` or `<class-factory-name>` elements:

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

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

    <interceptors>
        <interceptor>
            <instance>
                <cdi:bean>registrationListener</cdi:bean>
            </instance>
        </interceptor>
        <interceptor>
            <instance>
                <cdi:bean>activationListener</cdi:bean>
            </instance>
        </interceptor>
    </interceptors>

    <caching-scheme-mapping>
        <cache-mapping>
            <cache-name>*</cache-name>
            <scheme-name>distributed-scheme</scheme-name>
            <interceptors>
                <interceptor>
                    <instance>
                        <cdi:bean>cacheListener</cdi:bean>
```

```
                    </instance>
                </interceptor>
            </interceptors>
        </cache-mapping>
    </caching-scheme-mapping>

    <caching-schemes>
        <distributed-scheme>
            <scheme-name>distributed-scheme</scheme-name>
            <service-name>PartitionedCache</service-name>
            <local-storage system-
property="coherence.distributed.localstorage">true</local-storage>
            <partition-listener>
                <cdi:bean>partitionListener</cdi:bean>
            </partition-listener>
            <member-listener>
                <cdi:bean>memberListener</cdi:bean>
            </member-listener>
            <backing-map-scheme>
                <local-scheme/>
            </backing-map-scheme>
            <autostart>true</autostart>
            <interceptors>
                <interceptor>
                    <instance>
                        <cdi:bean>storageListener</cdi:bean>
                    </instance>
                </interceptor>
            </interceptors>
        </distributed-scheme>
    </caching-schemes>
</cache-config>
```

You can inject the *named* CDI beans (beans with an explicit `@Named` annotations) through the `<cdi:bean>` element. For example, the `cacheListener` interceptor bean used above would look similar to this:

```
@ApplicationScoped
@Named("cacheListener")
@EntryEvents(INSERTING)
public class MyCacheListener
        implements EventInterceptor<EntryEvent<Long, String>> {
    @Override
    public void onEvent(EntryEvent<Long, String> e) {
        // handle INSERTING event
    }
}
```

Also, keep in mind that only `@ApplicationScoped` beans can be injected, which implies that they may be shared. For example, because a wildcard, `*`, is used as a cache name within the cache mapping in the example, the same instance of `cacheListener` will receive events from multiple caches.

This is typically fine, as the event itself provides the details about the context that raised it, including cache name and the service it was raised from. However, it does imply that any

shared state that you may have within the listener class should not be context-specific, and it must be safe for concurrent access from multiple threads. If you cannot guarantee the latter, you may want to declare the `onEvent` method as `synchronized`, to ensure that only one thread at a time can access any shared state you may have.

This section includes the following topic:

- Using CDI Observers to Handle Coherence Server-Side Events

Using CDI Observers to Handle Coherence Server-Side Events

While the above examples show that you can implement any Coherence `EventInterceptor` as a CDI bean and register it using the `<cdi:bean>` element within the cache configuration file, Coherence CDI also provides a much simpler way to accomplish the same goal using the standard CDI Events and Observers.

For example, to observe events raised by a `NamedMap` with the name `people`, with keys of type `Long`, and values of type `Person`, you would define a CDI observer such as the following:

```
private void onMapChange(@Observes @MapName("people") EntryEvent<Long,
Person> event) {
    // handle all events raised by the 'people' map/cache
}
```

This section includes the following topics:

- Observing Specific Event Types
- Filtering the Observed Events
- Transforming the Observed Events
- Observing Events for Maps and Caches in Specific Services and Scopes
- Using Asynchronous Observers

Observing Specific Event Types

The `@Observes` method (see Using CDI Observers to Handle Coherence Server-Side Events) receives all events for the `people` map, but you can also control the types of events received using event qualifiers:

```
private void onUpdate(@Observes @Updated @MapName("people") EntryEvent<Long,
Person> event) {
    // handle UPDATED events raised by the 'people' map/cache
}

private void onChange(@Observes @Inserted @Updated @Removed
@MapName("people") EntryEvent<?, ?> event) {
    // handle INSERTED, UPDATED and REMOVED events raised by the 'people' map/
cache
}
```

Filtering the Observed Events

The events observed can be restricted further by using a Coherence `Filter`. If a filter has been specified, the events are filtered on the server and are never sent to the client. The filter is specified using a qualifier annotation that is itself annotated with `@FilterBinding`.

You can implement a Custom `FilterBinding` (recommended, see Using the FilterBinding Annotations), or use a built-in `@WhereFilter` for convenience, which allows you to specify a filter using CohQL.

For example, to receive all event types in the `people` map, but only for `People` with a `lastName` property value of `Smith`, you can use the built-in `@WhereFilter` annotation:

```
@WhereFilter("lastName = 'Smith'")
private void onMapChange(@Observes @MapName("people") EntryEvent<Long,
Person> event) {
    // handle all events raised by the 'people' map/cache
}
```

Transforming the Observed Events

When an event observer does not want to receive the full cache or map value in an event, you can transform the event into a different value to be observed. This is achieved by using a `MapEventTransformer` that is applied to the observer method using either an `ExtractorBinding` annotation or a `MapEventTransformerBinding` annotation. Transformation of events happens on the server, so you can make observers more efficient as they do not need to receive the original event with the full old and new values.

This section includes the following topics:

- Transforming Events Using ExtractorBinding Annotations
- Transforming Events Using MapEventTransformerBinding Annotations

Transforming Events Using ExtractorBinding Annotations

An `ExtractorBinding` annotation is an annotation that represents a Coherence `ValueExtractor`. When an observer method has been annotated with an `ExtractorBinding` annotation, the resulting `ValueExtractor` is applied to the event's values and a new event is returned to the observer containing just the extracted properties.

For example, an event observer that is observing events from a map named `people`, but only requires the last name, you can use the built-in `@PropertyExtractor` annotation.

```
@PropertyExtractor("lastName")
private void onMapChange(@Observes @MapName("people") EntryEvent<Long,
String> event) {
    // handle all events raised by the 'people' map/cache
}
```

While the earlier examples received events of type `EntryEvent<Long, Person>`, this method receives events of type `EntryEvent<Long, String>` because the property extractor is applied to the `Person` values in the original event to extract the `lastName` property, creating a new event with `String` values.

There are a number of built in `ExtractorBinding` annotations, and it is also possible to create custom `ExtractorBinding` annotation. See Creating the Custom Extractor Annotation.

You can add multiple extractor binding annotations to an injection point, in which case multiple properties are extracted, and the event will contain a `List` of the extracted property values.

For example, if the `Person` also contains an `address` field of type `Address`, which contains a `city` field, this can be extracted with a `@ChainedExtractor` annotation. By combining this with the `@PropertyExtractor` in the earlier example, both the `lastName` and `city` can be observed in the event.

```
@PropertyExtractor("lastName")
@ChainedExtractor({"address", "city"})
private void onMapChange(@Observes @MapName("people") EntryEvent<Long,
List<String>> event) {
    // handle all events raised by the 'people' map/cache
}
```

> **Note:**
>
> Now the event is of type `EntryEvent<Long, List<String>>` because multiple extracted values are returned. The event value is a `List` and in this case both properties are of type `String`, so the value can be `List<String>`.

Transforming Events Using MapEventTransformerBinding Annotations

If more complex event transformations are required than just extracting properties from event values, you can create a custom `MapEventTransformerBinding` that will produce a custom `MapEventTransformer` instance and this instance is applied to the observer's events. See Creating the Custom Extractor Annotation.

Observing Events for Maps and Caches in Specific Services and Scopes

In addition to the `@MapName` qualifier, you can also use the `@ServiceName` and `@ScopeName` qualifiers as a way to limit the events received.

The earlier examples only show how to handle `EntryEvents`, but the same applies to all other server-side event types:

```
private void onActivated(@Observes @Activated LifecycleEvent event) {
    // handle cache factory activation
}

private void onCreatedPeople(@Observes @Created @MapName("people")
CacheLifecycleEvent event) {
    // handle creation of the 'people' map/cache
}

private void onExecuted(@Observes @Executed @MapName("people")
@Processor(Uppercase.class) EntryProcessorEvent event) {
    // intercept 'Uppercase` entry processor execution against 'people' map/
cache
}
```

Using Asynchronous Observers

All the earlier examples used synchronous observers by specifying `@Observes` qualifier for each observer method. However, Coherence CDI fully supports asynchronous CDI observers as well. All you need to do is replace `@Observes` with `@ObservesAsync` in any of the earlier examples.

```
private void onActivated(@ObservesAsync @Activated LifecycleEvent event) {
    // handle cache factory activation
}

private void onCreatedPeople(@ObservesAsync @Created @MapName("people")
CacheLifecycleEvent event) {
    // handle creation of the 'people' map/cache
}

private void onExecuted(@ObservesAsync @Executed @MapName("people")
@Processor(Uppercase.class) EntryProcessorEvent event) {
    // intercept 'Uppercase` entry processor execution against 'people', map/
cache
}
```

> **Note:**
>
> Coherence events fall into two categories: pre- and post-commit events. All the events whose name ends with "`ing`", such as `Inserting`, `Updating`, `Removing`, or `Executing` are pre-commit events. These events can either modify the data or veto the operation by throwing an exception, but to do so they must be synchronous to ensure that they are executed on the same thread that is executing the operation that triggered the event.
>
> This means that you can observe them using asynchronous CDI observers, but if you want to mutate the set of entries that are part of the event payload, or veto the event by throwing an exception, you must use synchronous CDI observer.

Injecting CDI Beans into Transient Objects

Using CDI to inject Coherence objects into application classes, and CDI beans into Coherence-managed objects enables you to support many use cases where dependency injection may be useful, but it does not cover an important use case that is specific to Coherence.

Coherence is a distributed system, and it uses serialization to send both the data and the processing requests from one cluster member (or remote client) to another, as well as to store data, both in memory and on disk.

Processing requests, such as entry processors and aggregators, have to be deserialized on a target cluster member(s) to be executed. In some cases, they could benefit from dependency injection to avoid service lookups.

Similarly, while the data is stored in a serialized, binary format, it may need to be deserialized into user supplied classes for server-side processing, such as when executing entry

processors and aggregators. In this case, data classes can often also benefit from dependency injection [to support Domain-Driven Design (DDD), for example].

While these transient objects are not managed by the CDI container, Coherence CDI supports their injection during deserialization. However, for performance reasons requires that you explicitly opt-in by implementing the `com.oracle.coherence.cdi.Injectable` interface.

This section includes the following topic:

- Making Transient Classes Injectable

Making Transient Classes Injectable

While not technically a true marker interface, `Injectable` can be treated as such for all intents and purposes. All you need to do is add it to the `implements` clause of your class to initiate injection on deserialization:

```
public class InjectableBean
        implements Injectable, Serializable {

    @Inject
    private Converter<String, String> converter;

    private String text;

    InjectableBean() {
    }

    InjectableBean(String text) {
        this.text = text;
    }

    String getConvertedText() {
        return converter.convert(text);
    }
}
```

Assuming that you have the following `Converter` service implementation in your application, it will be injected into `InjectableBean` during deserialization, and the `getConvertedText` method will return the value of the text field converted to upper case:

```
@ApplicationScoped
public class ToUpperConverter
        implements Converter<String, String> {
    @Override
    public String convert(String s) {
        return s.toUpperCase();
    }
}
```

> **Note:**
>
> If your `Injectable` class has the `@PostConstruct` callback method, it will be called after the injection. However, because you have no control over the object's lifecycle after that point, `@PreDestroy` callback will not be called.

This functionality is not dependent on the serialization format and will work with both Java and POF serialization (or any other custom serializer), and for any object that is deserialized on any Coherence member (or even on a remote client).

While the deserialized transient objects are not true CDI managed beans, being able to inject CDI managed dependencies into them upon deserialization will likely satisfy most dependency injection requirements you will ever have in those application components.

Using the FilterBinding Annotations

When creating views or subscribing to events, you can modify the view or events using `Filters`. The exact `Filter` implementation injected is determined by the view or event observers qualifiers. Specifically, any qualifier annotation that is itself annotated with the `@FilterBinding` annotation.

For example, if there is an injection point for a view that is a filtered view of an underlying map, but the filter required is more complex than those provided by the build in qualifiers, or is some custom filter implementation, then you should perform the following steps:

- Create a custom annotation class to represent the required `Filter`.
- Create a bean class implementing `com.oracle.coherence.cdi.FilterFactory` annotated with the custom annotation that will be the factory for producing instances of the custom `Filter`.
- Annotate the view injection point with the custom annotation.

This section includes the following topics:

Creating the Custom Filter Annotation

Creating the filter annotation is same as creating a normal Java annotation class that is annotated with the `@com.oracle.coherence.cdi.FilterBinding` annotation.

In the following example, the most important part is that this new annotation is annotated with `FilterBinding` so that the Coherence CDI extensions can recognize that it represents a `Filter`.

```
@Inherited
@FilterBinding
@Documented
@Retention(RetentionPolicy.RUNTIME)
```

```
public @interface CustomFilter {
}
```

Creating the Custom Filter Factory

After you create the custom annotation, you can create a `FilterFactories` implementation that is responsible for producing instances of the required `Filter`.

```
@ApplicationScoped
@CustomFilter
static class CustomFilterSupplier
        implements FilterFactory<CustomFilter, Object>
    {
    @Override
    public Filter<Object> create(CustomFilter annotation)
        {
        return new CustomComplexFilter();
        }
    }
```

In the above example:

- The `CustomFilterSupplier` class has been annotated with `@ApplicationScoped` to make is discoverable by CDI.
- The `CustomFilterSupplier` class has been annotated with the new filter binding annotation `@CustomFilter` so that the Coherence CDI extension can locate it when it needs to create `Filters`.
- The `CustomFilterSupplier` implements the `FilterFactories` interface's `create` method where it creates the custom `Filter` implementation.

Annotating the Injection Point

Because there is both a custom annotation and an annotated `FilterFactories`, you can annotate the injection point requiring the `Filter` with the new annotation.

```
@Inject
@View
@CustomFilter
private NamedMap<Long, Person> people;
```

Just like views, you can use custom filter binding annotations for event observers. For example, if there is an event observer method that should only receive events matching the same custom `Filter`, then you can annotate the method with the same custom filter annotation.

```
@CustomFilter
private void onPerson(@Observes @MapName("people") EntryEvent<Long, Person>
event) {
```

Using ExtractorBinding Annotations

Extractor bindings are annotations that are themselves annotated with `@ExtractorBinding` and are used in conjunction with an implementation of `com.oracle.coherence.cdi.ExtractorFactory` to produce Coherence `ValueExtractor` instances.

There are a many built-in extractor binding annotations in the Coherence CDI module and it is a simple process to provide custom implementations.

This section includes the following topics:

Using Built-In ExtractorBinding Annotations

The following types of built-in ExtractorBinding annotations are available for use:

PropertyExtractor

You can use the `@PropertyExtractor` annotation to obtain an extractor that extracts a named property from an object. The value field of the `@PropertyExtractor` annotation is the name of the property to extract.

For example, the following `@PropertyExtractor` annotation represents a `ValueExtractor` that will extract the `lastName` property from a value.

```
@PropertyExtractor("lastName")
```

The extractor produced will be an instance of `com.tangosol.util.extractor.UniversalExtractor`, so this example is the same as calling:

```
new UniversalExtractor("lastName");
```

You can apply the `@PropertyExtractor` annotation multiple times to create a `MultiExtractor` that extracts a `List` of properties from a value.

For example, consider a map named `people`, where the map values are instances of `Person` that has a `firstName` and a `lastName` property. The event observer below would observe events on that map, but the event received would only contain the event key and a `List` containing the extracted `firstName` and `lastName` from the original event.

```
@PropertyExtractor("firstName")
@PropertyExtractor("lastName")
private void onPerson(@Observes @MapName("people") EntryEvent<Long,
List<String>> event) {
```

ChainedExtractor

You can use the `@ChainedExtractor` annotation to extract a chain of properties.

For example, a `Person` instance might contain an `address` property that contains a `city` property. The `@ChainedExtractor` takes the chain of fields to be extracted, in this case, extracts the `address` from `Person`, and then extracts the `city` from the `address`.

```
@ChainedExtractor("address", "city")
```

Each of the property names is used to create a `UniversalExtractor` and the array of these extractors is used to create an instance of `com.tangosol.util.extractor.ChainedExtractor`.

The example above would be the same as calling:

```
UniversalExtractor[] chain = new UniversalExtractor[] {
        new UniversalExtractor("address"),
        new UniversalExtractor("city")
};
ChainedExtractor extractor = new ChainedExtractor(chain);
```

PofExtractor

You can use the `@PofExtractor` annotation to produce extractors that can extract properties from POF encoded values. The value passed to the `@PofExtractor` annotation is the POF path to navigate to the property to extract.

For example, if a `Person` value has been serialized using POF with a `lastName` property at index `4`, you can use a `@PofExtractor` annotation as shown below:

```
@PofExtractor(index = 4)
```

The code above creates a Coherence `com.tangosol.util.extractor.PofExtractor`, which is equivalent to calling:

```
com.tangosol.util.extractor.PofExtractor(null, 4);
```

Sometimes (for example when dealing with certain types of `Number`) the `PofExtractor` needs to know the type to be extracted. In this case, you can set the type value in the `@PofExtractor` annotation.

For example, if a `Book` value had a `sales` field of type `Long` at POF index `2`, you can extract the `sales` field using the following `@PofExtractor` annotation:

```
@PofExtractor(index = {2}, type = Long.class)
```

The code above creates a Coherence `com.tangosol.util.extractor.PofExtractor`, which is equivalent to calling:

```
com.tangosol.util.extractor.PofExtractor(Long.class, 2);
```

The `index` value for a `@PofExtractor` annotation is an 'int' array so you can pass multiple POF index values to navigate down a chain of properties to extract. For example, if `Person` contained an `Address` at POF index `5` and `Address` contained a `city` property at POF index `3`, you can extract the city extracted from a `Person` using the `@PofExtractor` annotation as shown below:

```
@PofExtractor(index = {5, 3})
```

Alternatively, if the value that is extracted from is annotated with `com.tangosol.io.pof.schema.annotation.PortableType` and the POF serialization code for the class is generated using the Coherence `com.tangosol.io.pof.generator.PortableTypeGenerator`, then you can pass the property names to the `@PofExtractor` annotation using its `path` field.

For example to extract the `lastName` field from a POF serialized `Person`, you can use the `@PofExtractor` annotation as shown below:

```
@PofExtractor(path = "lastName")
```

The `address city` example would be:

```
@PofExtractor(path = {"address", "city"})
```

The `Book sales` example would be:

```
@PofExtractor(path = "sales", type Long.class)
```

Using Custom ExtractorBinding Annotations

When the built-in extractor bindings are not suitable, or when a custom `ValueExtractor` implementation is required, then you can create a custom extractor binding annotation with a corresponding `com.oracle.coherence.cdi.ExtractorFactory` implementation.

You should perform the following steps to create a custom extractor:

- Create a custom annotation class to represent the required `ValueExtractor`.
- Create a bean class implementing `com.oracle.coherence.cdi.ExtractorFactory` annotated with the custom annotation that will be the factory for producing instances of the custom `ValueExtractor`.
- Annotate the view injection point with the custom annotation.

Creating the Custom Extractor Annotation

Creating the extractor annotation is same as creating a normal Java annotation class which is annotated with the `@com.oracle.coherence.cdi.ExtractorBinding` annotation.

The most important part is that the following new annotation is annotated with `ExtractorBinding` so that the Coherence CDI extensions can recognize that it represents a `ValueExtractor`.

```
@Inherited
@ExtractorBinding
@Documented
@Retention(RetentionPolicy.RUNTIME)
public @interface CustomExtractor {
}
```

Creating the Custom Extractor Factory

After you create the custom annotation, you can create an `ExtractorFactory` implementation that will be responsible for producing instances of the required `ValueExtractor`.

```
@ApplicationScoped
@CustomExtractor
static class CustomExtractorSupplier
        implements ExtractorFactory<CustomExtractor, Object, Object>
    {
    @Override
    public ValueExtractor<Object, Object> create(CustomExtractor annotation)
        {
        return new CustomComplexExtractor();
        }
    }
```

In the above example:

- The `CustomExtractorSupplier` class has been annotated with `@ApplicationScoped` to make is discoverable by CDI.
- The `CustomExtractorSupplier` class has been annotated with the new extractor binding annotation `@CustomExtractor` so that the Coherence CDI extension can locate it when it needs to create `ValueExtractor` instances.
- The `CustomExtractorSupplier` implements the `ExtractorFactory` interface's `create` method where it creates the custom `ValueExtractor` implementation.

Annotating the Injection Point

Because there is both a custom annotation and an annotated `ExtractorFactory`, you can annotate the injection point requiring the `ValueExtractor` with the new annotation.

```
@Inject
@View
@CustomExtractor
private NamedMap<Long, String> people;
```

Like views, you can use custom filter binding annotations for event observers. For example, if there is an event observer method that should only receive transformed events using the

custom extractor to transform the event, then you can use custom filter binding annotations as given below:

```
@CustomExtractor
private void onPerson(@Observes @MapName("people") EntryEvent<Long, String>
event) {
```

Using MapEventTransformerBinding Annotations

Coherence CDI supports event observers that can observe events for cache or map entries. You can annotate the `observer` method with a `MapEventTransformerBinding` annotation to indicate that the observer requires a transformer to be applied to the original event before it is observed.

For more information about Events, see Using CDI Observers to Handle Coherence Server-Side Events.

There are no built-in `MapEventTransformerBinding` annotations; this feature is to support use of custom `MapEventTransformer` implementations.

Perform the following steps to create and use a `MapEventTransformerBinding` annotation:

- Create a custom annotation class to represent the required `MapEventTransformer`.
- Create a bean class implementing `com.oracle.coherence.cdi.MapEventTransformerFactory` annotated with the custom annotation that will be the factory for producing instances of the custom `MapEventTransformer`.
- Annotate the view injection point with the custom annotation.

This section includes the following topics:

- Creating the Custom Extractor Annotation
- Creating the Custom Extractor Factory
- Annotating the Injection Point

Creating the Custom Extractor Annotation

Creating the extractor annotation is same as creating a normal Java annotation class which is annotated with the `@com.oracle.coherence.cdi.MapEventTransformerBinding` annotation.

The most important part is that the following new annotation is annotated with `MapEventTransformerBinding` so that the Coherence CDI extensions can recognize that it represents a `MapEventTransformer`.

```
@Inherited
@MapEventTransformerBinding
@Documented
@Retention(RetentionPolicy.RUNTIME)
public @interface CustomTransformer {
}
```

Creating the Custom Extractor Factory

After you create the custom annotation, you can create a `MapEventTransformerFactory` implementation that will be responsible for producing instances of the required `MapEventTransformer`.

```
@ApplicationScoped
@CustomTransformer
static class CustomExtractorSupplier
        implements ExtractorFactory<CustomExtractor, Object, Object>
    {
    @Override
    public ValueExtractor<Object, Object> create(CustomExtractor annotation)
        {
        return new CustomComplexExtractor();
        }
    }
```

In the above example:

- The `CustomTransformerSupplier` class has been annotated with `@ApplicationScoped` to make is discoverable by CDI.
- The `CustomTransformerSupplier` class has been annotated with the new extractor binding annotation `CustomTransformer` so that the Coherence CDI extension can locate it when it needs to create `MapEventTransformer` instances.
- The `CustomTransformerSupplier` implements the `MapEventTransformerFactory` interface's `create` method where it creates the custom `MapEventTransformer` implementation.

Annotating the Injection Point

Because there is both a custom annotation and an annotated `MapEventTransformerFactory`, you can annotate the observer method requiring the `MapEventTransformer` with the new annotation.

```
@CustomTransformer
private void onPerson(@Observes @MapName("people") EntryEvent<Long, String>
event) {
```

Using CDI Response Caching

CDI response caching allows you to apply caching to Java methods transparently. CDI response caching will be enabled after you add the `coherence-cdi` dependency.

Declare `coherence-cdi` as a dependency in the project's `pom.xml` file, as shown below:

```
<dependency>
    <groupId>${coherence.groupId}</groupId>
    <artifactId>coherence-cdi</artifactId>
```

```
    <version>${coherence.version}</version>
</dependency>
```

This section includes the following topic:

- Response Caching Annotations

Response Caching Annotations

The specific cache to be used for response caching can be declared by the `@CacheName` and `@SessionName` annotations on a class or a method.

The following response caching annotations are supported:

- @CacheAdd
- @CacheGet
- @CacheKey
- @CachePut
- @CacheRemove

@CacheAdd

A method marked with the `@CacheAdd` annotation is always invoked and its execution result is stored in the cache. The key is made up of the values of all parameters (in this case just the string parameter `name`).

```
@Path("{name}")
@POST
@CacheAdd
@CacheName("messages")
public Message addMessage(@PathParam("name") String name)
    {
    return new Message("Hello " + name);
    }
```

@CacheGet

If the return value is present in the cache, it is fetched and returned. Otherwise, the target method is invoked, the invocation result is stored in the cache, and then returned to the caller.

```
@Path("{name}")
@GET
@CacheGet
@CacheName("messages")
public Message getMessage(@PathParam("name") String name)
    {
    return new Message("Hello " + name);
    }
```

@CacheKey

The cache key is assembled from the values of all parameters that are not explicitly annotated with the `@CacheValue` annotation. If one or more parameters are annotated with the `@CacheKey` annotation, only those parameters will be used to create the key.

In this example, only the values of the `lastName` and `firstName` parameters will be used to create the cache key.

```
@Path("{lastName}/{firstName}")
@GET
@CacheGet
public Message getMessage(@PathParam("lastName") @CacheKey String lastName,
                          @PathParam("firstName") @CacheKey String firstName,
                          @HeaderParam("Accept-Language") String
acceptLanguage)
    {
    return new Message("Hello " + firstName + " " + lastName);
    }
```

@CachePut

The value of the `@CacheValue` annotated parameter is stored in the cache, the target method is invoked, and the invocation result is returned to the caller.

In this example, the passed message will be stored in the cache for the key whose value was passed as the `name` parameter.

```
@Path("{name}")
@POST
@Consumes(MediaType.APPLICATION_JSON)
@CachePut
@CacheName("messages")
public Response putMessage(@CacheKey @PathParam("name") String name,
                           @CacheValue Message message)
    {
    return Response.status(Response.Status.CREATED).build();
    }
```

@CacheRemove

This annotation removes the key from the cache and returns the result of the method invocation.

In this example, the key whose value was passed as the `name` parameter will be removed from the cache.

```
@Path("{name}")
@DELETE
@CacheRemove
public Response removeMessage(@PathParam("name") String name)
    {
```

```
return Response.ok().build();
}
```

Part VI

Using the Coherence JCache Implementation

Learn about the Coherence JCache provider; how to use the JCache API; and how to use Coherence JCache features. Try building a Coherence JCache application.
Part VI contains the following chapters:

43
Introduction to Coherence JCache

Learn about the Coherence implementation of the JSR-107 JCACHE - Java Caching API specification. The specification and API is commonly referred to as JCache in this documentation. A JCache overview section is also provided and includes a basic introduction to the API. For complete details about the API, download the JCache specification, Java sources, and JavaDoc from the Java Community Process (JCP) website: https://www.jcp.org/en/jsr/detail?id=107

This chapter includes the following sections:

- Overview of the Coherence JCache Implementation
- Comparison of JCache and NamedCache Features
- Dependencies for Coherence JCache
- Overview of Configuration for the Coherence JCache Provider
- JCache Primer

Overview of the Coherence JCache Implementation

Coherence includes a JCache provider implementation (`COHERENCE_HOME\lib\coherence-jcache.jar`).JCache is a common API for using caching in Java. Application developers use the JCache API (`javax.cache.*`) and Coherence provides the underlying caching capabilities. The provider-based approach guarantees cross-provider portability and allows developers to focus on application logic rather than creating and managing complex caching systems. See JCache Primer.
The Coherence JCache provider uses existing Coherence technology and can be thought of as a wrapper for the Coherence `NamedCache` API. This allows Coherence to reuse and expose many of its best-in-class technologies through JCache interfaces.

Supported Cache Types

The Coherence JCache provider offers the following cache types:

- Local Cache – A cache that is local to an application process. Entries that are stored in local caches do not persist after an application process ends. A local cache is similar to a `NamedCache` cache that is configured using a local-scheme cache definition. The local cache implementation is defined in the `com.tangosol.coherence.jcache.localcache` package. See Creating Local Caches.
- Partitioned Cache – A cache that is partitioned (distributed) among multiple processes in a Coherence cluster. Entries that are stored in partitioned caches are backed up and persist on the cluster after an application process ends. A partitioned cache is similar to a `NamedCache` cache that is configured using a distributed-scheme cache definition. The partitioned cache implementation is defined in the `com.tangosol.coherence.jcache.partitionedcache` package. See Creating Partitioned Caches and Understanding Distributed Caches.
- Pass-Through Cache – A cache that delegates to an existing `NamedCache` cache. Pass-through caches offer applications the ability to use all Coherence native features from a

JCache interface. Pass-through caches are ideal for applications that want to migrate to JCache but also want to reuse their existing Coherence application components. The pass-through cache implementation is defined in the `com.tangosol.coherence.jcache.passthroughcache` package. See Creating Pass-Through Caches.

- Remote Cache – A cache that delegates to a partitioned cache through a proxy service. A remote cache is similar to a partitioned cache except that access to the cluster is through Coherence*Extend. Remote caches are ideal for applications that want to use a Coherence cache, but do not want to become members of the cluster. The remote cache implementation is defined in the `com.tangosol.coherence.jcache.remote` package. See Creating Remote Caches.

Coherence JCache Events

The Coherence JCache provider uses the native Coherence event APIs to implement JCache events. The local and partitioned cache implementations leverage the Coherence `MapListener` API. The pass-through cache implementation only supports the Coherence `MapEvents` that map directly to the JCache events and there is no support for JCache expired events. Each cache type is responsible for registering map listeners and for dispatching map events. Event classes are located in the `com.tangosol.coherence.jcache.common` package as well as in each cache type package. See Using JCache Events .

> **Note:**
>
> - There is no equivalent of the `javax.cache.event.CacheEntryExpiredListener` API in the Coherence `MapListener` API.
> - The `NamedCache.clear` method results in a Coherence `MapListener.entryDeleted` event; however, the JCache `Cache.clear` method does not result in any events.

Coherence JCache Entry Processors

The Coherence JCache provider uses the native Coherence `InvocableMap.EntryProcessor` API to implement JCache entry processors. Each cache type includes an `InvokeProcessor` class in their respective `processors` package that is responsible for executing JCache entry processors when the `invoke` or `invokeAll` methods are called from a cache. See Processing JCache Entries .

The `processors` package for each cache type also includes many native Coherence entry processors that are used to perform cache operations. For example, when using a partitioned cache, the use of the `put` method results in the use of the `PutProcessor` class.

Coherence Serialization for JCache

The Coherence JCache provider makes use of Coherence Portable Object Format (POF) serialization. POF is a proven binary format within Coherence and is efficient in both space and time. For partitioned and pass-through caches, many cache operations utilize POF. Cache configuration, entry processors, event listeners and filters, and JCache statistics also make use of POF.

POF is also used to provide serialization as required by parts of the JCache specification. The `com.tangosol.coherence.jcache.serialization` package includes POF serializer

implementations to support JCache factory builders, expiry policies, and cache entry listener configuration.

Lastly, applications can choose to use POF for serialization as required; however, it is not a requirement when using the Coherence JCache provider. In use cases where portability between cache providers is a requirement, applications should use Java serialization.

Coherence JCache Management

The Coherence JCache provider implements the JCache `CacheMXBean` and `CacheStatisticsMXBean` MXBean interfaces. The implementation are located in the `com.tangosol.coherence.jcache.common` package. Management information for local and partitioned caches are registered to the default MBean server and are found under the `javax.cache` namespace. Management information for pass-through caches are reported using the native Coherence JMX management implementation. See Viewing JCache Management Information.

Comparison of JCache and NamedCache Features

The Coherence JCache provider offers support for many features that are also available with the Coherence native `NamedCache` API.However, not all features are available through JCache. Table 43-1 shows a comparison of the two APIs.

Table 43-1 Comparison of JCache and NamedCache Features

Feature	JCache	NamedCache
Local Cache	X	X
Partitioned (Distributed) Cache	X	X
Replicated Cache		X
Optimistic Cache		X
Near Cache		X
Read Through	X	X
Write Through	X	X
Events	X	X
Query Filters		X
Indexes		X
Entry Processors	X	X
Aggregation		X

Dependencies for Coherence JCache

Applications that use JCache and the Coherence JCache provider must include addition libraries on the application classpath. The libraries include:

- `COHERENCE_HOME\lib\cache-api.jar` – The standard JCache API.
- `COHERENCE_HOME\lib\coherence-jcache.jar` – The Coherence JCache provider implementation.
- `COHERENCE_HOME\lib\coherence.jar` – The core Coherence library.

The Coherence JCache provider includes a service definition in the `META-INF/services` directory of the `coherence-jcache.jar` library. The definition allows Coherence to be automatically loaded and used as the default caching provider by applications that use the `javax.cache.Caching` bootstrap class. See Specifying Coherence as the JCache Provider.

Overview of Configuration for the Coherence JCache Provider

The JCache provider utilizes the same configuration files as the native Coherence `NamedCache` API. However, the need to customize the configuration files has been simplified and in some cases not required at all. The following lists the configuration file used by the JCache implementation:

- `tangosol-coherence-override.xml` – A Coherence operational override file is used when configuring a Coherence cluster for partitioned caches and when using pass-through caches. The override file is not required for local caches. See Specifying an Operational Configuration File.
- `coherence-jcache-cache-config.xml` – A Coherence JCache provider-specific cache configuration file that is called `coherence-jcache-cache-cofig.xml` is included in the provider JAR file and used to create local and partitioned caches; however, applications are not expected to edit the configuration. The included cache configuration file defines a `JCacheNamespace` handler class that is used to programmatically define local and partitioned caches for use by JCache applications.
- `coherence-jcache-pof-config.xml` – A Coherence JCache provider-specific POF configuration file that is called `coherence-jcache-pof-cofig.xml` is included in the provider JAR file and is used to define JCache POF types. See Configuring a JCache POF Configuration file.

JCache Primer

Read an overview of the JCache specification and API that is intended for those that are new to JCache.The overview includes basic concepts that are used when completing the instructions in this book. If you are familiar with JCache, you can skip this section.
This section is not intended to replace the specification or the API documentation. For complete details, download the JCache specification, Java sources, and JavaDoc from the Java Community Process (JCP) website:

`https://www.jcp.org/en/jsr/detail?id=107`

This section includes the following topics:

- What is JCache
- JCache Caching Providers and Cache Managers
- JCache Caches
- JCache Cache Configuration
- JCache Custom Programming
- JCache Management

What is JCache

The JCache specification defines an API for creating and using caches in Java programs. Applications often use caches to store and reuse resources that require a significant cost to

create. Applications can then quickly access the resources in memory without having to incur the cost associated with recreating the resources. Applications commonly use caching to increase application performance, availability, and scalability.

The JCache API defines a provider-based model for caching. The provider model separates the cache client API from the cache implementation. Applications use a well-defined client API and cache providers are responsible for the actual cache implementation. The provider-based model frees application developers from having to create and manage complex caching sub-systems and ensures portability between cache providers that implement the specification.

JCache Caching Providers and Cache Managers

The JCache API defines the `CachingProvider` and `CacheManager` interfaces. Applications use the `CachingProvider` interface to get and use a cache manager. Applications use the `CacheManager` interface to create and use caches. Applications are free to use multiple cache providers. However, a cache manager can only be associated with a single cache provider.

JCache offers several ways to get cache providers and access cache managers. A common access pattern that is used throughout this documentation is to use the `Caching` bootstrap class. The class provides a convenient way to get a `CachingPovider` implementation and automatically discovers providers that include a standard Java service definition.

The following example gets a default `CachingProvider` implementation and then creates a cache manager:

```
CachingProvider cachingProvider = Caching.getCachingProvider();
CacheManager cacheManager = cachingProvider.getCacheManager();
```

JCache Caches

The JCache API defines a `Cache` interface. The `Cache` interface creates a data structure that stores key and value pairs. Each key and value pair is referred to an entry and is defined by the `Cache.Entry` interface. A cache is similar to a Java `Map` data structure; However, there are some key differences:

- Keys or values cannot be null.
- `Cache.put` operations do not return an entry's previous value. The `Cache.getAndPut` operation is functionally equivalent to the `Map.put` operation.
- Cache Entries expire and can be evicted.
- Values can be automatically loaded from, and written to, an external source to support read-through and write-through caching.
- Cache entry changes can be observed.
- Cache statistics can be collected.

The `Cache` interface includes methods for putting, getting, replacing, and removing cache entries. Methods are also provided for loading a cache, registering cache listeners, and invoking entry processors.

The following example demonstrates using a cache manager to create a `Cache` instance called `MyCache`:

```
Cache<String, Integer> cache = cacheManager.createCache("MyCache", config);
```

JCache makes use of generics to support both compile and runtime type checking. The example creates a cache that requires keys to be of type `String` and values to be of type `Integer`. Type checking is not required but is often a best practice.

Operations are performed on the cache instance:

```
String key = "k";
Integer value = 1;

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

JCache Cache Configuration

The JCache API defines the `CompleteConfiguration` interface that is used to configure a cache. The `MutableConfiguration` class is a default implementation of the interface. The configuration options include:

- setting store-by semantics (by value or by reference)
- setting cache entry types
- setting cache expiry
- enabling read-through and write-through caching
- enabling management and statistics

Caches are configured when the cache is created. In the previous cache example, the `createCache` method required both a name and configuration for the cache. The following example demonstrates creating a configuration object to be used by the `createCache` method:

```
MutableConfiguration<String, Integer> config =
   new MutableConfiguration<String, Integer>();
config.setStoreByValue(true).setTypes(String.class, Integer.class)
   .setReadThrough(true)
   .setWriteThrough(true)
   .setManagementEnabled(true)
   .setStatisticsEnabled(true)
   .setExpiryPolicyFactory(CreatedExpiryPolicy.factoryOf(Duration.FIVE_MINUTES));

Cache<String, Integer> cache = cacheManager.createCache("MyCache", config);
```

JCache Custom Programming

The JCache API provides multiple interfaces that give applications the ability to customize caching. To use an implementation, you must use the `Factory` interface to create the instance. This ensures that the instance is serializable. The following interfaces can be implemented by an application:

- `ExpiryPolicy` – This interface is used to define when (`Duration`) cache entries expire based on entry creation, access, and modification operations.
- `CacheLoader` – This interface is used to load data into a cache from an external resource as is required when using read-through caching.
- `CacheWriter` – This interface is used to write data to an external resource as is required when using read-through caching.
- `CacheEntryListener` – A set of subinterfaces (`CacheEntryCreatedListener`, `CacheEntryUpdatedListener`, `CacheEntryRemovedListener`, and `CacheEntryExpiredListener`) that are used to receive and react to Cache events.

- `EntryProcessor` – This interface is used to perform compound operations on cache entries in an atomic, lock-free manner. Unlike the other interfaces, the `EntryProcessor` interface does not require the use of the `Factory` interface to create an instance; however, the implementation will need to be serialized if the intention is to use distributed caching.

JCache Management

The JCache API defines two dynamic MBeans that are used to manage caches. The `CacheMXBean` MXBean reports cache configuration information. The `CacheStatisticsMXBean` MXBean reports cache performance statistics that are used to troubleshoot potential issues. The dynamic MBeans are registered with an implementation-specific MBean server and are obtained using standard JMX; including, the use of any JMX MBean-compliant browser.

44

Building Your First Coherence JCache Application

You can follow step-by-step instructions for building and running a basic Coherence JCache example that demonstrates fundamental concepts of both JCache and the Coherence JCache provider.The sample application is a simple application that stores a domain object (the `Person` object) into a cache. The example demonstrates using a local cache, a partitioned cache, and a pass-through cache.
If you are new to Coherence, you may consider also running the native Coherence `NamedCache` example. See Building Your First Coherence Application.

This chapter includes the following sections:

Task 1: Create a Simple Object

Create a simple `Person` object to be cached using the Coherence JCache provider.The `Person` object contains a constructor and three fields for a first name, last name, and age. The `Person` object implements the `Serializable` interface. Serialization is required when the object is stored in a partitioned cache.

Example 44-1 A Simple Person Object

```
package com.examples;

import java.io.Serializable;

public class Person implements Serializable {
   private String m_sFirstName;
   private String m_sLastName;
   private int m_nAge;
   private static final long serialVersionUID = 99L;

   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 + ")";
   }
}
```

Task 2: Store the Object in a Local Cache

Applications use the JCache API to access and interact with a cache.The API provides methods for creating and using a cache. The default cache type that Coherence uses if no cache type is defined is a local cache (local to the application process).
This section includes the following topics:

- Create the Sample JCache Application
- Run the Sample JCache Application

Create the Sample JCache Application

The following application stores a single `Person` object to a local cache. The application demonstrates getting a cache provider, creating a cache manager, configuring and creating a cache, and using the cache.

Example 44-2 An Example JCache Application

```
package com.examples;

import javax.cache.Cache;
import javax.cache.CacheManager;
import javax.cache.Caching;
import javax.cache.configuration.MutableConfiguration;
import javax.cache.spi.CachingProvider;

public class JCacheExample {
   public static void main(String[] args)
   {
      CachingProvider cachingProvider = Caching.getCachingProvider();
      CacheManager cacheManager = cachingProvider.getCacheManager();

      MutableConfiguration<String, Object> config =
         new MutableConfiguration<String, Object>();
      config.setStoreByValue(true).setTypes(String.class, Object.class);

      Cache<String, Object> cache = cacheManager.createCache("MyCache", config);

      Person p = new Person("John","Doe",24);

      String key = "k";
      Person value = p;
```

```
        cache.put(key, value);
        System.out.println("\n Cache: " + cache.getName() + " contains: " +
              cache.get(key) + "\n");

        cacheManager.close();
    }
}
```

Run the Sample JCache Application

To run the standalone application example:

1. From a command prompt, compile the `Person.java` and `JCacheExample.java` files. The following example assumes that the files are in a single directory that is referred to as *`APPLICATION_HOME`* for the remainder of the tasks in this chapter:

   ```
   cd APPLICATION_HOME
   javac -cp COHERENCE_HOME\lib\cache-api.jar com\examples\*
   ```

2. Run the `JCacheExample` class and include the location of the `coherence.jar`, `coherence-jcache.jar`, and `cache-api.jar` libraries on the classpath using the Java `-cp` option. For example:

   ```
   java -cp .;COHERENCE_HOME\lib\cache-api.jar;
   COHERENCE_HOME\lib\coherence-jcache.jar;COHERENCE_HOME\lib\coherence.jar
   com.examples.JCacheExample
   ```

   Coherence log messages are emitted that indicate the Coherence configuration resources that are being used and the Coherence cache factory being created. The application emits the entry that is in the cache and then the application exits.

Task 3: Configure an Example Cluster

Partitioned caches and pass-through caches use a Coherence cluster to distribute cached data.This task creates 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 JVM processes do not attempt to join an existing Coherence cluster that may be running on the network.
To configure an example cluster:

1. Create a file named `tangosol-coherence-override.xml`.
2. Add the following override configuration and replace *`cluster_name`* with a value that is unique for this cluster. For example, use your name for the 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>cluster_name</cluster-name>
         </member-identity>
      </cluster-config>
   </coherence>
   ```

3. Save the file to the *`APPLICATION_HOME`*`\config` directory.

Task 4: Store the Object in a Partitioned Cache

A partitioned cache is a cache that distributes cache entries among any number of cache servers in a Coherence cluster.Entries that are stored in partitioned caches are backed up and persist on the cluster after the application process ends. See Creating Partitioned Caches. In this task, two separate Java processes form the cluster: a cache server process and the `JCacheExample` application process. For simplicity, the two processes are collocated on a single computer. The cache server, by default, is configured to store cache data. Lastly, a Coherence `CacheFactory` is used to verify that the `JCacheExample` application successfully created and loaded the cache on the cluster.

This section includes the following topics:

- Start the Example Cache Server
- Run The Application
- Verify the Cache

Start the Example Cache Server

From a command prompt, start a cache server instance using the `Coherence` class and use the Java `-cp` option to include the *`APPLICATION_HOME`*`\config` directory. The classpath must also include the `cache-api.jar`, `coherence-jcache.jar`, and `coherence.jar` libraries. Make sure that the operational override file and the `coherence-jcache.jar` are loaded on the classpath before the `coherence.jar` library. Lastly, use the `coherence.cacheconfig` system property to explicitly use the JCache `coherence-jcache-cache-config.xml` cache configuration file that is located in the `coherence-jcache.jar`. For example:

```
java -Dcoherence.cacheconfig=coherence-jcache-cache-config.xml
-cp APPLICATION_HOME\config;COHERENCE_HOME\lib\cache-api.jar;
COHERENCE_HOME\lib\coherence-jcache.jar;COHERENCE_HOME\lib\coherence.jar
com.tangosol.net.Coherence
```

From the cache server output, notice that a distributed cache service that is called `jcache-partitioned-service` is created and is the senior member of the cluster:

```
(thread=DistributedCache:jcache-partitioned-service, member=1): Service
jcache-partitioned-service joined the cluster with senior service member 1
```

Run The Application

The `coherence.jcache.configuration.classname` system property configures the Coherence JCache provider to use a partitioned cache instead of a local cache. The application code does not need to be modified in any way, which allows portability between JCache providers. In addition, Coherence manages the application scope and cache configuration.

To store the `Person` object in a partitioned cache:

- From a command prompt, run the application that was created in Example 44-2 and set the `coherence.jcache.configuration.classname` system property to `partitioned` and the `coherence.distributed.localstorage` system property to `false`. Use the Java `-cp` option to include the *`APPLICATION_HOME`*`\config` directory. The classpath must also include the `cache-api.jar`, `coherence-jcache.jar`, and `coherence.jar` libraries. Make sure that the operational override file and the `coherence-jcache.jar` library are loaded on the classpath before the `coherence.jar` library. For example:

```
java -Dcoherence.jcache.configuration.classname=partitioned
-Dcoherence.distributed.localstorage=false
-cp .;APPLICATION_HOME\config;COHERENCE_HOME\lib\cache-api.jar;
COHERENCE_HOME\lib\coherence-jcache.jar;COHERENCE_HOME\lib\coherence.jar
com.examples.JCacheExample
```

Coherence log messages are emitted that indicate the Coherence configuration resources that are being used. Notice that the `tangosol-coherence-override.xml` file was loaded. Lastly, notice that the application process joins the cluster and that the `jcache-partitioned-service` instance joins with the senior service on the cache server:

```
(thread=DistributedCache:jcache-partitioned-service, member=2): Service
jcache-partitioned-service joined the cluster with senior service member 1
```

Verify the 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 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 cache and list all items in the cache.

To verify the cache:

1. From a command prompt, start a standalone cache factory instance using the `CacheFactory` class. Use the Java `-cp` option to include the *`APPLICATION_HOME`*`\config` directory. The classpath must also include the `Person` object, `cache-api.jar`, `coherence-jcache.jar`, and `coherence.jar` libraries. Make sure that the operational override file and the `coherence-jcache.jar` are loaded on the classpath before the `coherence.jar` library. Lastly, set the `coherence.cacheconfig` system property to `coherence-jcache-cache-config.xml` and the `coherence.distributed.localstorage` system property to `false`. For example:

   ```
   java -Dcoherence.cacheconfig=coherence-jcache-cache-config.xml
   -Dcoherence.distributed.localstorage=false
   -cp APPLICATION_HOME\config;APPLICATION_HOME\person.jar;
   COHERENCE_HOME\lib\cache-api.jar;COHERENCE_HOME\lib\coherence-jcache.jar;
   COHERENCE_HOME\lib\coherence.jar 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 `MyCache` cache using the `cache` command:

   ```
   cache jcache-partitioned-coherence-jcache-cache-config.xml$MyCache
   ```

   > **Note:**
   >
   > For the purpose of this example, the cache name includes the application scope. Applications do not need to explicitly include the application scope when using a cache.

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

```
list
```

The command returns and displays:

```
k = Person( John Doe : 24)
```

4. Shutdown all processes.

Task 5: Store the Object in a Pass-Through Cache

A pass-through cache is a cache that delegates to a pre-existing Coherence cache (a cache that is defined in a Coherence cache configuration file).Pass-through caches allow you to use all of the native features of Coherence and provide greater control over cache configuration. In this task, two separate Java processes form the cluster: a cache server process and the `JCacheExample` application process. For simplicity, the two processes are collocated on a single computer. The cache server, by default, is configured to store cache data. Lastly, a Coherence `CacheFactory` is used to verify that the `JCacheExample` application successfully created and loaded the cache on the cluster.

This section includes the following topics:

Define the Example Cache

For this example, a cache configuration is created that defines a distributed cache that is explicitly mapped to the `MyCache` name.

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>MyCache</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 the file to the *APPLICATION_HOME*\config directory.

Start the Example Cache Server

From a command prompt, start a cache server instance using the Coherence class and use the Java -cp option to include the *APPLICATION_HOME*\config directory and the coherence.jar library. Make sure that the operational override file is loaded on the classpath before the coherence.jar library. Lastly, use the coherence.cacheconfig system property to explicitly define the example-config.xml cache configuration file. For example:

```
java -Dcoherence.cacheconfig=example-config.xml
-cp APPLICATION_HOME\config;COHERENCE_HOME\lib\coherence.jar com.tangosol.net.Coherence
```

Run the Application

The coherence.jcache.configuration.classname system property configures the Coherence JCache provider to use a pass-through cache. The application code does not need to be modified in any way.

From a command prompt, run the JCacheExample class and set the coherence.jcache.configuration.classname system property to passthrough, the coherence.cacheconfig system property to example-config, and the coherence.distributed.localstorage system property to false. Use the Java -cp option to include the *APPLICATION_HOME*\config directory. The classpath must also include the cache-api.jar, coherence-jcache.jar, and coherence.jar libraries. Make sure that the operational override file is loaded on the classpath before the coherence.jar library. For example:

```
java -Dcoherence.jcache.configuration.classname=passthrough
-Dcoherence.cacheconfig=example-config.xml
-Dcoherence.distributed.localstorage=false
-cp .;APPLICATION_HOME\config;COHERENCE_HOME\lib\cache-api.jar;
COHERENCE_HOME\lib\coherence-jcache.jar;COHERENCE_HOME\lib\coherence.jar
com.examples.JCacheExample
```

Coherence log messages are emitted that indicate the Coherence configuration resources that are being used. Notice that the tangosol-coherence-override.xml file and example-config.xml file were loaded. The application process connects to the cluster that contains the cache server process and both processes are running the DistributedCache service. As before, the application emits the entry that is in the cache and then the application exits.

Verify the 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 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 MyCache cache and list all items in the cache.

To verify the cache:

1. From a command prompt, start a standalone cache factory instance using the CacheFactory class. Use the Java -cp option to include the *APPLICATION_HOME*\config

directory. The classpath must also include the `Person` object and the `coherence.jar` library. Make sure that the operational override file is loaded on the classpath before the `coherence.jar` library. Lastly, set the `coherence.cacheconfig` system property to `example-config.xml` and the `coherence.distributed.localstorage` system property to `false`. For example:

```
java -Dcoherence.cacheconfig=example-config.xml
-Dcoherence.distributed.localstorage=false
-cp APPLICATION_HOME\config;APPLICATION_HOME\person.jar;
COHERENCE_HOME\lib\coherence.jar 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 `MyCache` cache using the `cache` command:

```
cache MyCache
```

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

```
list
```

The command returns and displays:

```
k = Person( John Doe : 24)
```

45
Performing Basic Coherence JCache Tasks

Learn basic tasks that are typical of application development when using the Coherence JCache provider.The instructions demonstrate the JCache API and include details that are specific to the Coherence JCache provider implementation.
This chapter includes the following sections:

Specifying Coherence as the JCache Provider

The `META-INF/services/javax.cache.spi.CachingProvider` service definition assures that applications using the `javax.cache.Caching` bootstrap class use the Coherence JCache provider by default.

The provider definition is located in the `coherence-jcache.jar` library. The instructions and examples in this chapter assume that the Coherence JCache provider is the default JCache provider. However, applications are able to register and use multiple cache providers. In such cases, applications have several options to select the Coherence JCache provider.

> **Note:**
>
> If multiple JCache providers are registered, then the use of the `getCachingProvider()` or `getCachingProvider(ClassLoader)` methods result in a cache exception.

The first option is to override the default cache provider using the `javax.cache.spi.cachingprovider` system property and specifying the fully qualified name of the Coherence JCache provider implementation class. For example:

```
System.setProperty("javax.cache.spi.cachingprovider",
   "com.tangosol.coherence.jcache.CoherenceBasedCachingProvider");
```

The system property can also be specified on the command line at runtime. For example,

```
-Djavax.cache.spi.cachingprovider=com.tangosol.coherence.jcache.
   CoherenceBasedCachingProvider
```

The second option is to use the `Caching.getCachingProvider(String)` or `Caching.getCachingProvider(String, ClassLoader)` methods to explicitly request the Coherence JCache provider implementation. For example:

```
CachingProvider cachingProvider = Caching.getCachingProvider(
   "com.tangosol.coherence.jcache.CoherenceBasedCachingProvider");
CacheManager cacheManager = cachingProvider.getCacheManager();
...
```

Use the `getCachingProviders` methods to iterate the list of registered providers if multiple caching providers are registered.

Lastly, applications can directly instantiate the Coherence JCache provider. However, this option is not portable across providers. The following example instantiates the Coherence JCache provider:

```
CachingProvider cachingProvider = new CoherenceBasedCachingProvider();
CacheManager cacheManager = cachingProvider.getCacheManager();
```

Creating Coherence JCache Caches

The Coherence JCache provider supports four cache types: local, partitioned, pass-through, and remote.A local cache is a cache that is local to the application process. A partitioned cache distributes cached data across Coherence storage-enabled cluster members. A pass-through cache delegates to any native Coherence cache that is configured within a Coherence cache configuration file. A remote cache is a cache that allows Coherence extend clients to access a cache using the proxy service. This section demonstrates how to create each of these cache types using the JCache API.
This section includes the following topics:

- Creating Local Caches
- Creating Partitioned Caches
- Creating Pass-Through Caches
- Creating Remote Caches
- Using Native Coherence Functionality from JCache

Creating Local Caches

Applications can cache data using a local cache. Local caches are the default cache type when using the Coherence JCache provider and are similar to a Coherence local cache scheme. Local caches do not provide data backup and the data does not persist after the application process exits. The following example creates a local cache using the JCache API.

```
CachingProvider cachingProvider = Caching.getCachingProvider();
CacheManager cacheManager = cachingProvider.getCacheManager();

MutableConfiguration<String, String> config =
   new MutableConfiguration<String, String>();
config.setStoreByValue(true).setTypes(String.class, String.class);

Cache<String, String> cache = cacheManager.createCache("MyCache", config);
```

A local cache results in the creation of a Coherence `NamedCache` instance that use `jcache-local-*` cache names that are mapped to a local cache scheme that is named `jcache-local-scheme`, which is managed by the `jcache-local-service` service.

Creating Partitioned Caches

Applications can cache data to a Coherence partitioned cache. The application process automatically joins a Coherence cluster and Coherence manages the distribution and backup of the data across the cluster. Partitioned caches do not require any application code changes; therefore, existing JCache applications can easily be migrated to use Coherence partitioned caches.

> **Note:**
>
> Partitioned caches do not support store-by-reference semantics.

To create a partitioned cache, use the `tangosol.coherence.jcache.configuration.classname` property and set it to `partitioned`. For example:

```
-Dcoherence.jcache.configuration.classname=partitioned
```

The system property also supports a value of `local`, which is the default value if no value is specified and results in a local cache being created.

A partitioned cache results in the creation of a Coherence `NamedCache` instance that use `jcache-partitioned-*` cache names that are mapped to a partitioned cache scheme that is named `jcache-partitioned-scheme`, which is managed by the `jcache-partitioned-service` service.

Operational Configuration

JCache applications must use a `tangosol-coherence-override.xml` operational override file or Coherence system properties, or both, to configure Coherence operational settings. The operational settings, among other things, allow applications to join an existing cluster. See Specifying an Operational Configuration File. For example, an application can specify the following system properties at runtime to join a cluster that is named `Cluster1` and has a multicast address of `231.1.1.1` and port `7574`.

```
-Dcoherence.cluster=Cluster1
-Dcoherence.clusteraddress=231.1.1.1
-Dcoherence.clusterport=7574
```

Cache Configuration

Partitioned caches for JCache automatically use a default Coherence cache configuration file that is included in the `coherence-jcache.jar` library. The configuration file is called `coherence-jcache-cache-config.xml`. Any cache servers in the cluster must also use the `coherence-jcache-cache-config.xml` file. For example, a cache server can explicitly set the cache configuration file using the `coherence.cacheconfig` system property:

```
-Dcoherence.cacheconfig=coherence-jcache-cache-config.xml
```

You can also use an existing Coherence cache configuration file. See Using Native Coherence Functionality from JCache.

Creating Pass-Through Caches

Applications can use a pass-through cache to delegate all cache operations to pre-existing Coherence caches. A pass-through cache results in the use of a native Coherence `NamedCache` instance from a JCache interface. A pass-through cache allows an application to take full advantage of all Coherence native features and configuration.

To create a pass-through cache, an application can use the Coherence `PassThroughCacheConfiguration` JCache configuration object and specify the name of an existing cache mapping that is defined in a Coherence cache configuration file. For example:

```
...
CachingProvider cachingProvider = Caching.getCachingProvider();
CacheManager cacheManager = cachingProvider.getCacheManager();

PassThroughCacheConfiguration<String, Object> config =
   new PassThroughCacheConfiguration<String, Object>();
config.setTypes(String.class, Object.class);

Cache<String, Object> cache = cacheManager.createCache("MyCache", config);
```

As an alternative, existing JCache applications can continue to use a `MutableConfiguration` object and create a pass-through cache by specifying the `coherence.jcache.configuration.classname` property and setting the value to `passthrough`. For example:

```
-Dcoherence.jcache.configuration.classname=passthrough
```

The system property allows JCache applications to use Coherence native features without having to change any application code. However, most `MutableConfiguration` object properties are ignored. A warning message is emitted if the system property is used to create a pass-through cache.

```
WARNING: Lossy conversion of configuration
javax.cache.configuration.MutableConfiguration to a PassThroughConfiguration.
Most properties from class javax.cache.configuration.MutableConfiguration are
ignored. Configure PassThroughCache using native Coherence configuration
methodologies.
```

Operational Configuration

Pass-through caches for JCache use a `tangosol-coherence-override.xml` operational override file or Coherence system properties, or both, to configure Coherence operational settings. The operational settings, among other things, allow applications to join an existing cluster. See Specifying an Operational Configuration File.

Cache Configuration

Pass-through caches for JCache use an existing Coherence cache configuration file. Applications can specify the location of the file at runtime using the `coherence.cacheconfig` property. For example:

```
-Dcoherence.cacheconfig=my-cache-config.xml
```

The above technique changes the default URI a cache manager uses to get a specified cache configuration file. An alternative way to specify a cache configuration that uses the JCache API and does not rely on setting a system property is to specify the URI when creating a cache manager. For example:

```
Caching.getCacheManager(new URI(my-cache-config.xml), null, null);
```

Any cache created from the above returned cache manager uses the specified cache configuration file to the `getCacheManager` call. See Specifying a Cache Configuration File.

Creating Remote Caches

Applications can cache data to a remote Coherence partitioned cache. The application process connects to a Coherence proxy service on the cluster and Coherence manages the distribution and backup of the data across the cluster. Remote caches rely on Coherence*Extend. See Overview of Coherence*Extend in *Developing Remote Clients for Oracle Coherence*. Remote caches do not require any application code changes; therefore, existing JCache applications can easily be migrated to use Coherence remote caches.

To create a remote cache, use the `tangosol.coherence.jcache.configuration.classname` property and set it to `remote`. For example:

```
-Dcoherence.jcache.configuration.classname=remote
```

A remote cache results in the creation of a Coherence `NamedCache` instance that use `jcache-extend-*` cache names that are mapped to a remote cache scheme that is named `jcache-extend-tcp`. The cache is configured to connect to the cluster using a proxy service that is named `TCPProxyService`.

Operational Configuration

JCache applications must use a `tangosol-coherence-override.xml` operational override file or Coherence system properties, or both, to configure Coherence operational settings if required. See Specifying an Operational Configuration File.

Cache Configuration

Remote caches for JCache automatically use a default Coherence cache configuration file that is included in the `coherence-jcache.jar` library. The configuration file is called `coherence-jcache-extendclient-cache-config.xml`. In addition, at least one cache servers in the cluster must use the `coherence-jcache-server-proxy-cache-config.xml` file, which defines the proxy service. For example, a cache server can explicitly set the cache configuration file using the `coherence.cacheconfig` system property:

```
-Dcoherence.cacheconfig=coherence-jcache-server-proxy-cache-config.xml
```

You can also choose to use an existing Coherence cache configuration file. See Using Native Coherence Functionality from JCache.

Using Native Coherence Functionality from JCache

The Coherence JCache provider allows applications to use native Coherence functionality. Applications that use native Coherence functionality are not portable among JCache providers. Applications typically use native functionality to reuse existing Coherence implementations and configuration.

This section includes the following topics:

- Accessing NamedCache Instances from JCache
- Using Coherence Configuration with JCache

Accessing NamedCache Instances from JCache

JCache applications can directly access the underlying Coherence `NamedCache` instance for a cache. A cache cannot be accessed directly using the `NamedCache` and JCache API interchangeably. In addition, using the `NamedCache` instance to access cache keys and values bypasses additional behavior provided by the Coherence JCache provider implementation.

The following example directly accesses the underlying `NamedCache` instance for a JCache cache and uses the instance to put and get an entry:

```
CachingProvider cachingProvider = Caching.getCachingProvider();
CacheManager cacheManager = cachingProvider.getCacheManager();

MutableConfiguration<String, String> config =
   new MutableConfiguration<String, String>();
config.setStoreByValue(true).setTypes(String.class, String.class);

Cache<String, String> cache = cacheManager.createCache("MyCache", config);

String key = "k";
String value = "Hello World";

NamedCache nc = cache.unwrap(NamedCache.class);

nc.put(key, value);

System.out.println("The value is " + nc.get(key) + "\n");

nc.destroy();
cacheManager.close();
```

Using Coherence Configuration with JCache

Any native Coherence concept that does not have an equivalent JCache concept (such as eviction) can be configured using a Coherence cache configuration file, operational override file, or Coherence system properties.

> **Note:**
>
> Applications that are designed to be portable across JCache provider implementations should only rely on the configuration that is provided through the JCache `CompleteConfiguration` API.

When using an existing Coherence cache configuration file for JCache partitioned and local caches, then the file must include a `jcache` namespace handler definition. For example:

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

The JCache namespace handler generates any missing elements that are required for the JCache-specific cache configuration file and allows an applications to extend the standard

cache configuration file. The JCache namespace handler does not generate an element if it is already defined in the cache configuration file.

Configuring Coherence JCache Caches

Coherence JCache-based caches are configured programmatically using the `MutableConfiguration` class during cache creation.This is a standard JCache class and should be used to maintain portability among JCache providers. For example:

```
MutableConfiguration<String, String> config =
   new MutableConfiguration<String, String>();
config.setStoreByValue(true).setTypes(String.class, String.class);

Cache<String, String> cache = cacheManager.createCache("MyCache", config);
```

Applications that use local and partitioned caches can optionally use the Coherence JCache provider-specific implementations of the JCache `CompleteConfiguration` API. The implementations are `LocalCacheConfiguration` and `PartitionedCacheConfiguration`, respectively. However, the use of these classes is not portable among JCache providers. For example:

```
PartitionedCacheConfiguration<String, String> config =
   new PartitionedCacheConfiguration<String, String>();
config.setStoreByValue(true).setTypes(String.class, String.class);
```

Application that use pass-through caches, require the Coherence JCache provider `PassThroughCacheConfiguration` configuration class. Pass-through caches are configured using native Coherence configuration and therefore do not support using JCache-specific configuration options.

```
PassThroughCacheConfiguration<String, String> config =
   new PassThroughCacheConfiguration<String, String>();
```

This section includes the following topics:

- Setting Store-By Semantics
- Setting Cache Entry Types
- Setting Cache Expiry
- Enabling Read-Through and Write-Through Caching
- Enabling Management

Setting Store-By Semantics

JCache provides the option to specify whether a cache uses store-by-reference or store-by-value semantics. The `setStoreByValue` method configures store-by semantics for both keys and values. A value of `true` indicates store-by-value semantics and a value of `false` indicates store-by-reference semantics. The Coherence JCache provider uses store-by-value semantics by default for all cache types and only supports store-by-reference for local caches. Partitioned caches do not support store-by-reference. In addition, the provider relies on either Java Serialization or POF to implement store-by-value semantics for all Coherence JCache cache types. Applications that use local caches and configure store-by-reference do not require that keys and values be serializable or POF-enabled.

The following example configures a local cache to use store-by-reference semantics.

```
CachingProvider cachingProvider = Caching.getCachingProvider();
CacheManager cacheManager = cachingProvider.getCacheManager();

MutableConfiguration<String, Integer> config =
   new MutableConfiguration<String, Integer>();
config.setStoreByValue(false);

Cache<String, Integer> cache = cacheManager.createCache("MyCache", config);
```

> **Note:**
>
> If store-by-reference is enabled, the cache can be mutated by any threads holding a reference to the cache. Keys that are mutated may not be retrievable. In addition, values that are mutated may be observed by all threads in the JVM if Java permissions are not properly set. Beyond the local heap, entries need to be transformed into a representation and any mutations that occur after the transformation may not be reflected in the cache.

Setting Cache Entry Types

JCache allows you to set the expected key and value types of an entry. Any entry that is placed in the cache must then adhere to the declared types; otherwise, a runtime exception occurs. The `setTypes` method is used to configure the entry types, which cannot be set to `null`. The following example sets the key type to `String` and the value type to `Integer`:

```
CachingProvider cachingProvider = Caching.getCachingProvider();
CacheManager cacheManager = cachingProvider.getCacheManager();

MutableConfiguration<String, Integer> config =
   new MutableConfiguration<String, Integer>();
config.setTypes(String.class, Integer.class);

Cache<String, Integer> cache = cacheManager.createCache("MyCache", config);
```

> **Note:**
>
> The Coherence JCache implementation does not perform runtime checks to ensure that objects passed into mutative cache operations match with what was configured through the `javax.cache.configuration.Configuration` API. Applications should use generics to operate on the cache and take advantage of static compile time checking.

The default key and value type, if no type is configured, is `Object.class`. Using the default type or explicitly configuring the `Object.class` type disables runtime type checking and allows and application to use any key or value type. However, it is then the responsibility of the application to ensure type safety. The following example disables type checking by explicitly setting the key and value types to `Object.class` and omitting the type parameters when getting the cache:

```
CachingProvider cachingProvider = Caching.getCachingProvider();
CacheManager cacheManager = cachingProvider.getCacheManager();

MutableConfiguration<Object, Object> config =
```

```
   new MutableConfiguration<Object, Object>();
config.setStoreByValue(true).setTypes(Object.class, Object.class);

Cache<Object, Object> cache = cacheManager.getCache("MyCache");
```

Although it is not required, the above examples also use compile-time type checking. When omitted, the compiler will not check for type safety. The following example disables both compile-time and runtime type checking.

```
MutableConfiguration config = new MutableConfiguration();
cacheManager.createCache("MyCache", config);
```

Setting Cache Expiry

JCache allows you to configure the amount of time an entry is available in a cache. Entries that expire are no longer valid and are not available to an application.

> **Note:**
>
> The Coherence JCache provider may evict entries that are not due to expire if the maximum cache capacity is reached. Cache entries are evicted using a default eviction policy that is based on a combination of how often and recently entries are accessed. Entries that are accessed least frequently and are not accessed for the longest period are evicted first.

Expiry is configured using the `setExpiryPolicyFactory` method to select an expiry policy and to set a time value. The following example configures entries in the cache to expire 5 minutes after they are created:

```
CachingProvider cachingProvider = Caching.getCachingProvider();
CacheManager cacheManager = cachingProvider.getCacheManager();

MutableConfiguration<String, Integer> config =
   new MutableConfiguration<String, Integer>();
config.setExpiryPolicyFactory(CreatedExpiryPolicy.factoryOf(
   Duration.FIVE_MINUTES));

Cache<String, Integer> cache = cacheManager.createCache("MyCache", config);
```

If the policy is set to `null` or if expiry is not configured, then the eternal policy is used by default and entries are never expired. Each policy can specify the duration of time that must pass before a cache entry is considered expired. For convenience, pre-defined durations are provided: `ZERO`, `ONE_MINUTE`, `FIVE_MINUTES`, `TEN_MINUTES`, `TWENETY_MINUTES`, `THIRTY_MINUTES`, `ONE_HOUR`, and `ONE_DAY`. Any non-negative duration value can be configured.

The following expiry policies are supported:

- Created – Cache entries are expired based on when they are created. An update does not reset the expiry time. This policy is implemented in the JCache `CreatedExpiryPolicy` class.
- Accessed – Cache entries are expired based on the last time they accessed. Accessed does not include a cache update. This policy is implemented in the JCache `AccessedExpiryPolicy` class.

- Modified – Cache entries are expired based on the last time it was updated. Updating includes created and changing (updating) an entry. This policy is implemented in the JCache `ModifiedExpiryPolicy` class.
- Touched – Cache Entry based on when it was last touched. A touch includes creation, update or access. This policy is implemented in the JCache `TouchedExpiryPolicy` class.
- Eternal (default) – Cache entries are never expired. This policy is implemented in the JCache `EternalExpiryPolicy` class.

Expiry policies and durations must be serialized to support distributed caching scenarios. The Coherence JCache provider uses Portable Object Format (POF) for serialization and includes POF serializers for expiry types. These types must be configured to be used by the Coherence JCache provider if partitioned caches are used together with pass-through caches. See Configuring a JCache POF Configuration file.

Enabling Read-Through and Write-Through Caching

Read-through and write-through caching allow applications to load data into a cache from a data source and to write data from a cache into a data source. Databases are the most common data source integration, but any data source can be integrated with JCache caches. Read-through and write-through caching are not enabled by default and require application-specific implementations of the JCache `CacheLoader` and `CacheWriter` interfaces, respectively.

To enable read-through or write-through caching, set the `setReadThrough` and `setWriteThrough` methods, respectively, to `true`. For example:

```
CachingProvider cachingProvider = Caching.getCachingProvider();
CacheManager cacheManager = cachingProvider.getCacheManager();

MutableConfiguration<String, Integer> config =
   new MutableConfiguration<String, Integer>();
config.setReadThrough(true).setWriteThrough(true);

Cache<String, Integer> cache = cacheManager.createCache("MyCache", config);
```

To register a `CacheLoader` and `CacheWriter` implementation, use the `setCacheLoaderFactory` and `setCacheWriterFactory` methods, respectively, and enter the fully qualified name of the implementation class. For example:

```
cacheWriter = new MyCacheWriter();
cacheLoader = new MyCacheLoader();

MutableConfiguration<String, Integer> config =
   new MutableConfiguration<String, Integer>();
config.setReadThrough(true).setWriteThrough(true)
   .setCacheWriterFactory(FactoryBuilder.factoryOf(cacheWriter));
   .setCacheLoaderFactory(FactoryBuilder.factoryOf(cacheLoader));

Cache<String, Integer> cache = cacheManager.createCache("MyCache", config);
```

The implementations are automatically used when performing certain cache operations. See Using Read-Through and Write-Through Caching.

Enabling Management

JCache provides two dynamic mbeans: the `CacheMXBean` and `CacheStatisticsMXBean` mbeans. The `CacheMXBean` mbean shows the configuration settings of a cache. The

`CacheStatisticsMXBean` mbean shows performance statistics for a cache. See Viewing JCache Management Information.

Management is disabled by default. To enable management information, set the `setManagementEnabled` and `setStatisticsEnabled` methods to `true` when configuring a cache. For example:

```
CachingProvider cachingProvider = Caching.getCachingProvider();
CacheManager cacheManager = cachingProvider.getCacheManager();

MutableConfiguration config = new MutableConfiguration();
config.setManagementEnabled(true).setStatisticsEnabled(true);

Cache cache = cacheManager.createCache("MyCache", config);
```

Management can also be enabled at runtime using the cache manager:

```
CachingProvider cachingProvider = Caching.getCachingProvider();
CacheManager cacheManager = cachingProvider.getCacheManager();

MutableConfiguration config = new MutableConfiguration();

Cache cache = cacheManager.createCache("MyCache", config);

cacheManager.enableManagement("MyCache", true);
cacheManager.enableStatistics("MyCache", true);
```

Performing Cache Operations

JCache includes many operations for interacting with a cache.The operations are defined in the `javax.cache.Cache` interface. Refer to the `Cache` API documentation for the semantics and details of all the available JCache operations.
A cache must first be created and configured before cache operations can be performed. The following example demonstrates performing simple `put` and `get` operations:

> **Note:**
>
> Key and values cannot be `null`.

```
CachingProvider provider = Caching.getCachingProvider();
CacheManager cacheManager = provider.getCacheManager();

MutableConfiguration config = new MutableConfiguration();

Cache cache = cacheManager.createCache("MyCache", config);

cache.put("k1", "Hello World");
System.out.println("The value is " + cache.get("k1")+ "\n");
```

The above example does not perform compile or runtime entry type checking. See Setting Cache Entry Types.

Understanding Coherence JCache Operations

Cache operations are implemented differently for each cache type: `LocalCache`, `PartitionedCache`, and `PassThroughCache`. For example, partitioned cache operations are

implemented using Coherence entry processors that can take advantage of Coherence data serialization. Each of the cache types provides implementations of cache operations that take advantage of the cache topology being used.

Using Read-Through and Write-Through Caching

JCache supports the use of read-through and write-through caching to integrate caches with external resources.

Read-through caching automatically loads entries from external resources and write-through caching automatically writes entries to external resources. Read-through and write-through caching require an application to implement the JCache `CacheLoader` and `CacheWriter` interfaces, respectively. Read-through and writer-through caching are not enabled by default. See Enabling Read-Through and Write-Through Caching.

> **Note:**
>
> Applications that choose to use a Coherence pass-through cache, can use read-through and write-through caching natively in Coherence and are not required to include JCache specific loader and writer implementations.

This section includes the following topics:

- Providing a Read-Through Implementation
- Pre-Loading a Cache
- Providing a Write-Through Implementation

Providing a Read-Through Implementation

Read-through caching requires an application to implement the `CacheLoader` interface. The implementation details are specific to the external resource that is being used. For details about implementing `CacheLoader` methods, refer to the JCache API documentation.

The `CacheLoader` interface provides the `load` and `loadAll` methods. An implementation uses these methods to include the necessary logic to connect and retrieve a value, or set of values, from an external resource. A cache loader implementation is automatically invoked whenever an entry for a given key is not found as a result of performing a `get`, `getAll`, `getAndRemove`, or `getAndReplace` operation. When using entry processors, the `invoke` and `invokeAll` operations automatically use the cache loader if the entry processor uses the `getValue` method. See Processing JCache Entries .

Pre-Loading a Cache

The `Cache` API includes the `Cache.loadAll` method that can be used to pre-load a cache. The use of this method results in a call to the `loadAll` method of a `CacheLoader` implementation. In this case, the `CacheLoader` implementation must still be configured on the cache, but read-through caching does not need to be enabled. For example:

```
cacheLoader = new MyCacheLoader();

MutableConfiguration<String, String> config =
```

```
   new MutableConfiguration<String, String>();
config.setTypes(String.class, String.class)
   .setReadThrough(false)
   .setCacheLoaderFactory(FactoryBuilder.factoryOf(cacheLoader));
```

In the above example, the `factoryOf` method takes an instance of the `MyCacheLoader` implementation. The method is used for implementations that have state and requires that the implementation class be serializable. If the implementation does not have state, then it is best to use the `factoryOf(Class)` or `factoryOf(String classname)` methods instead.

Loading a cache from an external resource may take a long time to complete. The `CompletionListener` interface allows an application to be notified when the cache has been loaded or if an exception has occurred while loading the cache. The `CompletionListenerFuture` class is a default implementation that can be used as required. For example:

```
cacheLoader = new MyCacheLoader();

MutableConfiguration<String, String> config =
   new MutableConfiguration<String, String>();
config.setTypes(String.class, String.class)
   .setReadThrough(false)
   .setCacheLoaderFactory(FactoryBuilder.factoryOf(cacheLoader));
Cache<String, String> cache = cacheManager.createCache("MyCache", config);

CompletionListenerFuture future = new CompletionListenerFuture();

cache.loadAll(keys, true, future);
```

Providing a Write-Through Implementation

Write-through caching requires an application to implement the `CacheWriter` interface. The implementation details are specific to the external resource that is being used. For details about implementing `CacheWriter` methods, refer to the JCache API documentation.

The `CacheWriter` interface provides methods for writing entries (`write` and `writeAll`) to an external resource and methods for deleting entries (`delete` and `deleteAll`) from an external resource. An implementation uses these methods to include the necessary logic to connect and update data on an external resource. A cache writer implementation is automatically invoked whenever the following cache operations are performed: `put`, `putAll`, `putIfAbsent`, `remove`, `removeAll`, `replace`, `getAndPut`, `getAndRemove`, `getAndReplace`, `invoke`, and `invokeAll`.

Configuring a JCache POF Configuration file

The Coherence JCache provider uses POF for serialization and requires the `coherence-pof-config.xml` file.JCache specific POF types are defined in the `coherence-jacache.jar\coherence-jcache-pof-config.xml` POF configuration file. If you are using partitioned caches together with pass-through caches, the JCache POF types must be included as part of your POF configuration file. 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>coherence-jcache-pof-config.xml</include>
...
```

See Specifying a POF Configuration File.

Viewing JCache Management Information

Learn about managing Coherence JCache and the management data that is collected using the `CacheMXBean` and `CacheStatisticsMXBean` JMX MBeans.
This section includes the following topics:

- Overview of JCache Management Information
- Understanding the JCache CacheConfiguration MBean
- Understanding the JCache CacheStatistics MBean
- Changing the Refresh Interval for Partitioned Cache Statistics

Overview of JCache Management Information

The Coherence JCache provider implements both the `CacheMXBean` and `CacheStatisticsMXBean` Dynamic MBean interfaces. The interfaces provide the following management information:

- `CacheMXBean` – Reports configuration information for a cache. Each configuration option and its current settings are listed as attributes.
- `CacheStatisticsMXBean` – Reports performance information for a cache. The performance statistics are listed as attributes and are used to help troubleshoot possible issues with a cache.

Management information is not enabled by default. See Enabling Management.

> **Note:**
>
> Management information for pass-through caches are reported using the native Coherence JMX management implementation. See Using JMX to Manage Oracle Coherence in *Managing Oracle Coherence*.

Coherence JCache MBeans

You can view management information using any JMX MBean-compliant browser such as the VisualVM console, which is included with the JDK (*`JDK_HOME`*`/bin/jvisualvm`) if you are using JDK1.8. If you use a later JDK, you can install it from https://visualvm.github.io/. Management MBeans for local and partitioned caches are registered to the default MBean server and are found in a JMX browser under `javax.cache`.

Coherence VisualVM Plug-In

JCache management information can be viewed using the Coherence VisualVM plug-in. The plug-in includes a JCache tab that aggregates the management information over time and is used to troubleshoot configuration and performance issues. See Monitor a Coherence Cluster Using the Coherence VisualVM Plug-in in *Managing Oracle Coherence*.

Coherence Reports

JCache management information can also be viewed using Coherence reports. The Coherence JCache reports aggregate management information over time and are used to troubleshoot configuration and performance issues. See Understanding the JCache Configuration Report and Understanding the JCache Statistics Report in *Managing Oracle Coherence*.

Understanding the JCache CacheConfiguration MBean

The `CoherenceCacheMXBean` class implements the `CacheMXBean` interface and provides configuration information for a cache. An MBean object is registered for each cache. The object name of the MBean is:

`javax.cache:type=CacheConfiguration,CacheManager=coherence-jcache-cache-config.xml,Cache=`*`cache_name`*

This section includes the following topics:

- JCache CacheConfiguration MBean Attributes
- JCache CacheConfiguration MBean Operations

JCache CacheConfiguration MBean Attributes

Table 45-1describes the attributes for the `CacheConfiguration` MBean.

Table 45-1 CacheConfiguration MBean Attributes

Attribute	Type	Access	Description
`KeyType`	`String`	read-only	The required key type for the cache. The default value, if no required key type is configured, is `java.lang.object` and indicates that type checking is disabled.
`ManagementEnabled`	`Boolean`	read-only	Specifies whether management is enabled for the cache. The default value is `false`.
`ReadThrough`	`Boolean`	read-only	Specifies whether the cache operates in read-through mode. The default value is `false`.
`StatisticsEnabled`	`Boolean`	read-only	Specifies whether performance statistics are being collected for the cache. The default value is `false`.
`StoreByValue`	`Boolean`	read-only	Specifies whether the cache uses store-by-value or store by-reference semantics. The default value is `true` and indicates that keys and values are stored by value. A value of `false` indicates that keys and values are stored by reference.
`ValueType`	`String`	read-only	The required value type for the cache. The default value, if no required value type is configured, is `java.lang.object` and indicates that type checking is disabled.
`WriteThrough`	`Boolean`	read-only	Specifies whether the cache operates in write-through mode. The default value is `false`.

JCache CacheConfiguration MBean Operations

The `CacheConfiguration` MBean includes a `clear` operation that resets all configuration management information.

Understanding the JCache CacheStatistics MBean

The Coherence JCache provider includes two implementations of the JCache `CacheStatisticsMXBean` interface. The `ContextJCacheStatistics` implementation is used to collect performance statistics for local caches. The `PartitionedJCacheStatistics` is used to collect and aggregate performance statistics for all storage members in a Coherence cluster. An MBean object is registered for each cache. The object name of the MBean is:

`javax.cache:type=CacheStatistics,CacheManager=coherence-jcache-cache-config.xml,Cache=`*`cache_name`*

This section includes the following topics:

- JCache CacheStatistics MBean Attributes
- JCache CacheStatistics MBean Operations

JCache CacheStatistics MBean Attributes

Table 45-2 describes the attributes for the `CacheStatistics` MBean.

Table 45-2 Cache Statistics Attributes

Attribute	Type	Access	Description
`AverageGetTime`	`Float`	read-only	The average time to perform `get` operations. For read-through caches, the time does not include the time that is required to load entries because of a cache miss.
`AveragePutTime`	`Float`	read-only	The average time to perform `put` operations
`AverageRemoveTime`	`Float`	read-only	The average time to perform `remove` operations
`CacheEvictions`	`Long`	read-only	The total number of evictions from the cache. An eviction is a initiated by the cache to free up space. An eviction is not considered a `remove` operation. **Note:** This attribute is not implemented by the Coherence JCache provider.
`CacheGets`	`Long`	read-only	The total number of `get` operations. The value is equal to the sum of hits and misses and does not include operations that check for the existence of a key.
`CacheHitPercentage`	`Float`	read-only	The percentage of cache requests that return an entry. The percentage is reported as a decimal value and is calculated using the value of cache hits divided by cache `get` operations.
`CacheHits`	`Long`	read-only	The number of successful `get` operations

Table 45-2 (Cont.) Cache Statistics Attributes

Attribute	Type	Access	Description
`CacheMissPercentage`	`Float`	read-only	The percentage of cache requests that do not return an entry. The percentage is reported as a decimal value and is calculated using the value of cache misses divided by cache `get` operations.
`CacheMisses`	`Long`	read-only	The number of unsuccessful `get` operations
`CachePuts`	`Long`	read-only	The total number of put operations including operations that replace and existing entry.
`CacheRemovals`	`Long`	read-only	The total number of `remove` operations. The value does not include evictions initiated by the cache to free up space.

JCache CacheStatistics MBean Operations

The `CacheStatistics` MBean includes a `clear` operation that resets all cache statistics.

Changing the Refresh Interval for Partitioned Cache Statistics

The Coherence JCache provider uses a refresh interval to determine when to refresh performance statistics for partitioned caches. The refresh is performed lazily; statistics are refreshed only after the refresh interval is reached and a call to get a statistic is made. Statistics are not refreshed if a call to get a statistic is never made even if the refresh interval is reached.

The default refresh interval is 3 seconds and can be changed to accommodate different cluster sizes and network performance. For example, a cluster with many storage members may require a longer refresh interval to allow statistic aggregation to be performed across all members and to guard against constant updating.

To change the refresh interval, use the `coherence.jcache.statistics.refreshtime` system property and set it to a time entered in milliseconds. The following example configures the refresh interval to 5 seconds.

```
-Dcoherence.jcache.statistics.refreshtime=5000
```

46
Using JCache Events

You can create and register event listeners to handle JCache events.
The JCache event model is defined in the `javax.cache.event` package and described in chapter 8 of the *Java Caching API Specification* .

This chapter includes the following sections:

Overview of Using JCache Events

The JCache event model allows applications to receive and process events that represent observable changes to the entries in a cache.The event model uses standard Java event and event listener conventions that are common in Java applications.
A JCache event is defined in the `CacheEntryEvent` class and is a standard Java event that is specific to cache entries. JCache defines four event types for cache entries:

- `CREATED` – Indicates that the cache entry was created
- `UPDATED` – Indicates that the cache entry was updated
- `REMOVED` – Indicates that the cache entry was removed
- `EXPIRED` – Indicates that the cache entry has expired

The JCache event model uses event listeners to handle events and perform any necessary event processing. Each event type has a corresponding event listener interface that can be implemented as required. In addition, event filters can be used to process events before the events are dispatched to a listener. Event listeners and filters can be statically registered on a cache during configuration or dynamically after the cache has been created.

The Coherence JCache provider supports the use of JCache events for local, partitioned, and pass-through caches. The provider makes use of the native Coherence `MapListener` and `MapEvent` APIs to implement JCache events. Applications can directly access the underlying `NamedCache` instance and use the `MapListener` API; however, doing so directly bypasses additional behavior provided by the JCache Coherence implementation.

Creating Event Listeners

Each event type has a corresponding event listener interface that extends the `CacheEntryListener` interface.The event type interfaces include the `CacheEntryCreatedListener`, `CacheEntryUpdatedListener`, `CacheEntryRemovedListener`, and `CacheEntryExpiredListener` interfaces. A single event listener can choose to implement multiple event type listener interfaces or multiple event listeners can implement the same event type listener interfaces. The following example creates an event listener for `CREATED` events

that emits a log message that includes the event details every time an entry is created in a cache.

Example 46-1 An Example Event Listener Implementation

```
import java.io.Serializable;
import javax.cache.event.CacheEntryCreatedListener;
import javax.cache.event.CacheEntryEvent;
import javax.cache.event.CacheEntryListenerException;

public class MyCacheEntryListener<K, V>
   implements CacheEntryCreatedListener<K, V>, Serializable
   {
      private static final long serialVersionUID = 1L;

   public void onCreated(Iterable<CacheEntryEvent<? extends K, ? extends V>>
      events)
      throws CacheEntryListenerException
      {
         for (CacheEntryEvent<? extends K, ? extends V> event : events)
            {
            System.out.println("Received a " + event);
            }
      }
}
```

Event listeners that require serialization can use POF serialization. However, POF is not portable among cache provider implementations.

Creating Event Filters

Cache entry event filters provide the opportunity to evaluate cache events before the events are dispatched to an entry event listener.Event filters allow additional processing to be performed as required. Event filters must implement the `CacheEntryEventFilter` interface. The following example creates an event filter for `CREATED` events that stops the event from being dispatched and emits a system message.

Example 46-2 An Example Event Filter Implementation

```
import java.io.Serializable;
import javax.cache.event.CacheEntryEvent;
import javax.cache.event.CacheEntryEventFilter;
import javax.cache.event.CacheEntryListenerException;
import javax.cache.event.EventType;

public class MyCacheEntryEventFilter<K, V>
   implements CacheEntryEventFilter<K, V>, Serializable
   {
      private static final long serialVersionUID = 1L;

   public boolean evaluate(CacheEntryEvent<? extends K, ? extends V> event)
      throws CacheEntryListenerException
      {
         boolean result = false;

         if (event.getEventType() == EventType.CREATED)
            {
            System.out.println("filter event=" + event + " filter result=" +
               result);
            }
```

```
            return result;
        }
}
```

Event filters that require serialization can use POF serialization. However, POF is not portable among cache provider implementations.

Registering Event Listeners and Filters

Cache entry listeners and event filters can be registered statically on a cache during configuration or dynamically after a cache instance has been created.Either method requires the use of a listener configuration. The configuration is defined in the `CacheEntryListenerConfiguration` interface. The `MutableCacheEntryListenerConfiguration` class provides a default implementation that can be used as required and is demonstrated in this section. The configuration is used to specify the listener implementation class to use, the filter implementation class to use, whether to send the old value as part of the event, and whether event notification should be synchronous or asynchronous. The order in which listeners are notified is not guaranteed.
This section includes the following topics:

- Registering Event Listeners and Filters During Cache Configuration
- Registering Event Listeners and Filters at Runtime

Registering Event Listeners and Filters During Cache Configuration

To register `CacheEntryListener` and `CacheEntryEventFilter` implementations statically during configuration, use the `addCacheEntryListenerConfiguration` method and include the implementation classes and configure the listener as required. The following example registers the event listener and filter that was created in Example 46-1 and Example 46-2, respectively.

```
CachingProvider cachingProvider = Caching.getCachingProvider();
CacheManager cacheManager = cachingProvider.getCacheManager();

m_listener = new MyCacheEntryListener();

MutableConfiguration<String, String> config =
   new MutableConfiguration<String, String>();
config.setTypes(String.class, String.class).addCacheEntryListenerConfiguration(
      new MutableCacheEntryListenerConfiguration<String, String>
      (FactoryBuilder.factoryOf(m_listener),FactoryBuilder.factoryOf(
      new MyCacheEntryEventFilter<String, String>()),true,true));

Cache<String, String> cache = cacheManager.createCache("MyCache", config);
```

The example configuration sets event notifications to include the old value of the entry and to use synchronous dispatching as indicated by the two `true` properties.

The `removeCacheEntryListenerConfiguration` method removes a cache entry listener that was previously registered.

Registering Event Listeners and Filters at Runtime

To register `CacheEntryListener` and `CacheEntryEventFilter` implementations dynamically on a cache instance at runtime, use the `RegisterCacheEntryListener` method and include the implementation classes and configure the listener as required. The following example registers the event listener and filter that was created in Example 46-1 and Example 46-2, respectively.

```
CachingProvider cachingProvider = Caching.getCachingProvider();
CacheManager cacheManager = cachingProvider.getCacheManager();

m_listener = new MyCacheEntryListener();

MutableConfiguration<String, String> config =
   new MutableConfiguration<String, String>();
config.setTypes(String.class, String.class);

Cache<String, String> cache = cacheManager.createCache("MyCache", config);

cache.registerCacheEntryListener(
   new MutableCacheEntryListenerConfiguration<String, String>
   (FactoryBuilder.factoryOf(m_listener),FactoryBuilder.factoryOf(
   new MyCacheEntryEventFilter<String, String>()),false,false));
```

The example configuration sets event notifications to not include the old value of the entry and to use asynchronous dispatching as indicated by the two `false` properties.

The `deregisterCacheEntryListener` method removes a cache entry listener that was previously registered.

47

Processing JCache Entries

You can create and use JCache entry processors to modify cache entries.Entry processors are defined in the `javax.cache.processor` package and described in Chapter 9 of the *Java Caching API Specification*.
This chapter includes the following sections:

- Overview of Processing JCache Entries
- Creating Entry Processors
- Using Entry Processors

Overview of Processing JCache Entries

JCache supports the use of entry processors to perform updates on cache entries in an atomic, lock-free manner.In distributed cache environments, entries are processed in parallel across each of the different servers that holds cached data.
Entry processors are often used to make multiple changes to a cache in a single operation. For example, the entry processor may evaluate the values for a set of keys and then return different values based on some logic. The application can only view the final result of the processing and does not have access to intermediary results while the entry processor is being invoked. If an error occurs during processing, then an exception is thrown and no changes are made to the cache.

Entry processors must implements the `EntryProcessor` interface. Entry processor implementations are invoked on cache entries at runtime using the `Cache.invoke` and `Cache.invokeAll` methods.

Lastly, the Coherence JCache provider supports the use of JCache entry processors for local, partitioned, and pass-through caches. The provider makes use of the native Coherence `InvocableMap.EntryProcessor` API to implement JCache entry processors.

Creating Entry Processors

JCache entry processors must implement the `EntryProcessor` interface.The interface contains a single `process` method that is used to provide any processing logic for cache entries. Entry processors operate on `MutableEntry` entries. The `MutableEntry` interface ensures that entry processors have exclusive access to entries during processing. The `process` method also allows any arguments to be defined. The arguments can be passed to the processor when the processor is invoked.
Operations that are performed as part of an entry processor are not visible to an application and only take affect after the `process` method has completed. If an error occurs during the `process` method, then no changes are made to the cache entries.

Entry processors are not guaranteed to be executed in-process and therefore should always be serializable. For example, when using a Coherence partitioned cache, an entry processor may be executed on the cache server that contains the data to be processed. For Coherence caches, entry processors have the option of using POF serialization. However, POF is not portable among cache provider implementations.

Example 47-1 creates an entry processor that increments the value of a cache entry by one. The value of the entry is an integer. The key, value, and return types are explicitly defined to ensure type safety as required by the API.

Example 47-1 An Example EntryProcessor Implementation

```
import java.io.Serializable;
import javax.cache.processor.EntryProcessor;
import javax.cache.processor.EntryProcessorException;
import javax.cache.processor.MutableEntry;

public class MyEntryProcessor implements EntryProcessor <String, Integer,
   Integer>, Serializable
   {
      public static final long serialVersionUID = 1L;

   public Integer process(MutableEntry<String, Integer> entry,
      Object... arguments) throws EntryProcessorException
      {

      if (entry.exists())
      {
         Integer current = entry.getValue();
         entry.setValue(current + 1);
         return current;
      }
      else
      {
         entry.setValue(0);
         return -1;
      }
   }
}
```

Using Entry Processors

The `Cache` API provides the `invoke` and `invokeAll` methods that are used to call an entry processor.The `invoke` method operates on a single cache entry and the `invokeAll` method operates on a set of cache entries. Entries are specified using the key name.
This section includes the following topics:

- Invoking Entry Processors for a Single Key
- Invoking Entry Processors for Multiple Keys

Invoking Entry Processors for a Single Key

Entry processors are invoked on a single key using the `invoke` method from a `Cache` instance. The following example demonstrates processing a single key. The examples uses the `MyEntryProcessor` implementation that was created in Example 47-1.

```
CachingProvider cachingProvider = Caching.getCachingProvider();
CacheManager cacheManager = cachingProvider.getCacheManager();

MutableConfiguration<String, Integer> config =
   new MutableConfiguration<String, Integer>();
config.setTypes(String.class, Integer.class);

Cache<String, Integer> cache = cacheManager.createCache("MyCache", config);
```

```
String key = "k";
Integer value = 1;

cache.put(key, value);

System.out.println("The value is " + cache.get(key) + "\n");

cache.invoke(key, new MyEntryProcessor());

System.out.println("The value is now " + cache.get(key) + "\n");
```

Invoking Entry Processors for Multiple Keys

Entry processors are invoked on multiple keys using the `invokeAll` method. The keys must be grouped together using a `Set` implementation. The processor is invoked for each key in the set and the order in which keys are processed is not guaranteed. The following example demonstrates processing multiple keys that are grouped using the `HashSet` class and also uses the `EntryProcessorResult.get` method to retrieve the value returned by the entry processor for a specific key. The example uses the `MyEntryProcessor` implementation that was created in Example 47-1.

```
CachingProvider cachingProvider = Caching.getCachingProvider();
CacheManager cacheManager = cachingProvider.getCacheManager();

MutableConfiguration<String, Integer> config =
   new MutableConfiguration<String, Integer>();
config.setTypes(String.class, Integer.class);

Cache<String, Integer> cache = cacheManager.createCache("MyCache", config);

String key = "k";
Integer value = 1;
String key1 = "k1";
Integer value1 = 1;

cache.put(key, value);
cache.put(key1, value1);

HashSet hs = new HashSet();
hs.add(key);
hs.add(key1);

Map<String, EntryProcessorResult<Integer>> map = cache.invokeAll(hs,
   new MyEntryProcessor());

System.out.println("The value of k is now " + cache.get(key) +
   " the result of invokeAll for k is previous value of " + map.get(key).get() +
   "\n");

System.out.println("The value of k1 is now " + cache.get(key1) +
   " the result of invokeAll for k1 is previous value of " + map.get(key1).get() +
   "\n");
```

Part VII

Developing and Running Polyglot Coherence Applications

Oracle Coherence enables writing server side objects such as `EntryProcessor`, `Filter`, `Aggregator` and so on in JavaScript. You can have both Java and JavaScript objects co-exist in the same Java Virtual Machine.
Part VII contains the following chapter:

48
Developing Polyglot Coherence Applications

This chapter demonstrates Coherence's polyglot capabilities by developing an application that uses server-side JavaScript to implement common Coherence server-side operations. Using this functionality requires that you run Oracle Coherence with Oracle GraalVM Enterprise Edition. See Running Oracle WebLogic Server and Coherence on GraalVM Enterprise Edition.

> **Note:**
>
> To build and run the examples in this chapter, you must have Maven 3.8.6 or later installed.

> **Note:**
>
> The GraalVM Polyglot Runtime does not currently support virtual threads, so make sure that virtual threads are not enabled for the cache service for which you want to use polyglot features.

This chapter includes the following sections:

- Setting Up the Project
- Implementing JavaScript Classes
- Installing and Using Dependencies
- Building a Project
  To make your JavaScript classes available to Coherence Polyglot Framework at runtime, you need to package them in a JAR file.
- Using JavaScript Classes in a Java Application

Setting Up the Project

To demonstrate Coherence's polyglot capabilities, you will create a simple application that populates a Coherence map and then you'll use JavaScript to access and process the map entries you created.
If you already have a Maven Java project for your server-side classes that will be deployed to all Coherence storage members, then you can simply create a `src/main/js` directory and create your JavaScript project within it.

Otherwise, first you must create a Maven project and then create your JavaScript project under the `src/main/js` directory:

1. Change to your Maven project directory.

```
cd coherence-js-app
export PRJ_DIR=`pwd`
mkdir -p src/main/js
cd src/main/js
export JS_DIR=`pwd`
```

This main project directory now can be referred to by the shell environment variable `${PRJ_DIR}`, and the JavaScript project directory can be referred to by using the `${JS_DIR}` environment variable.

2. In the JavaScript project directory, run:

```
npm init -y
```

Running this command will create a `package.json` file, which you will need to edit later.

3. Create aggregator module, `main.mjs`, that will be used to export all of the JavaScript classes defined in other modules.

```
echo > main.mjs
```

4. Modify the main attribute within `package.json` to point to the newly created `main.mjs` module.

```
{
  "name": "coherence-js-app",
  "version": "1.0.0",
  "main": "main.mjs",
  …
}
```

Implementing JavaScript Classes

Now, you will implement JavaScript classes that you can use to access and update the entries in a Coherence map.
For simplicity, assume that the map is populated with `Person` objects with attributes, such as `firstName`, `lastName`, `age`, and `gender`.

```
public class Person
    {
    private String firstName;
    private String lastName;
    private int age;
    private String gender;

    // constructors and accessors omitted for brevity
    }
```

Now, you can implement JavaScript classes that will allow you to query, aggregate, and update.

This section includes the following topics:

- Using Filters
- Using EntryProcessors
- Using Aggregators

Using Filters

`NamedMap` provides the `get()` and `put()` methods that allow you to access entries based on the entries' keys. However, in many cases you may want to retrieve entries based on attributes other than the key. Coherence defines a `Filter` interface for these situations.

Assume that you want to find all those persons who are in their teens, in the age group `13-19`. One way to implement this is to retrieve all entries and pick only those whose age is between 13 and 19, but this method is inefficient. Coherence helps to move the predicate to the storage nodes, which not only avoids huge data movement, but also enables parallel execution of the predicate on the storage nodes.

Coherence provides a number of built in filters to access entries on the storage nodes. See Querying Data in a Cache.

This section includes the following topics:

- Using a Filter Interface
- Implementing a Filter

Using a Filter Interface

The `Filter` interface defines a single method `evaluate(object)`, which takes an object to evaluate as an argument, and returns `true` if the specified object satisfies the criteria defined by the filter, or `false` if the specified object does not satisfy the criteria defined by the filter.

Your filter must adhere to the following:

1. Must define and export a class that has one method, named `evaluate`, that takes one argument.
2. The `evaluate(obj)` must return `true` if `obj` satisfies the criteria represented by the filter. Otherwise, it should return `false` .

Implementing a Filter

To write a filter that checks if the person is in his or her teens, do the following.

Change directory to `${JS_DIR}/src/main/js`. Create the `filters.mjs` file and add the following content:

```
export class IsTeen { // [1]
  evaluate(person) {  // [2]
    return person.age >= 13 && person.age <= 19;
  }
}
```

In the preceding example:

- [1] Defines and exports a class named `IsTeen` .
- [2] Defines a method named `evaluate` that takes a single parameter. This method checks if the `age` attribute is between 13 and 19.

Finally, re-export all classes defined in the `filters.mjs` module from the aggregator module, `main.mjs`:

```
export * from "./filters.mjs"
```

Using EntryProcessors

Coherence provides built in `EntryProcessors` which help to perform parallel updates to cache entries. See Overview of Entry Processor Agents.

If you want to update cache entries (that are filled with `Person` objects as per the example used in this chapter) such that all `lastName` attributes are in uppercase, one way to do this is to retrieve all entries, iterate over them and update them one by one, and finally write them back into the cache.

This is an inefficient method. Coherence provides a more efficient approach of shipping the processing logic where the data resides and thus eliminating the need for data movement. If you need to perform parallel updates to cache entries efficiently, you should use an `EntryProcessor`.

This section includes the following topics:

- Using an EntryProcessor Interface
- Implementing an EntryProcessor

Using an EntryProcessor Interface

An `EntryProcessor` defines a mandatory method called `process(entry)`, which takes a map entry to process as an argument and returns the result of the processing. To implement an `EntryProcessor` in your JavaScript, the class must:

1. Define an exported class with one method named `process` that takes one argument (which will be the map entry).
2. If the `process(entry)` method mutates the map entry 's value, then it must update the entry explicitly.

Implementing an EntryProcessor

To write an `EntryProcessor` that updates `lastName` to its uppercase value, create the `processors.mjs` file and add the following content:

```
export class UpperCaseProcessor { // [1]
    process(entry) { // [2]
      let person = entry.value;
      person.lastName = person.lastName.toUpperCase(); // [3]
      entry.value = person; // [4]
      return person.lastName; // [5]
    }
}
```

In the preceding example:

- [1] Defines and exports a class named `UpperCaseProcessor` .
- [2] Defines a method called `process()` that takes a single parameter. The `NamedMap` entry that needs to be processed is accessible through the `entry` argument that is passed to this method.
- [3] Converts the person's `lastName` to uppercase.
- [4] Updates the `entry` with the new value.
- [5] Returns the updated (uppercase) last name as the result of the processor execution.

Finally, re-export all classes defined in the `processors.mjs` module from the aggregator module, `main.mjs`:

```
export * from "./ processors.mjs"
```

Using Aggregators

You can use a Coherence `Aggregator` to retrive a single aggregated result from a cache, based on certain criteria. For example, retrieving the key of the oldest `Person` in the `Cache` you created in the previous examples. You can retrieve all entries and then find the `Person` with the maximum age, but you will be moving a lot data to the client.

Coherence defines an `Aggregator` interface which lets you compute partial results and then combine those partial results to get a single aggregated result. See Performing Data Grid Aggregation.

This section includes the following topics:

- Using an Aggregator Interface
- Writing an Aggregator

Using an Aggregator Interface

The `Aggregator` interface requires you to implement the following methods:

1. `accumulate(entry)`: Executes in parallel across all members and accumulates a single entry into the partial result for that member. In this method, the partial result is computed by using one or more attributes from `entry`. This method will be called multiple times on an `Aggregator`, once for each entry that was selected for aggregation on a given cluster member.
2. `getPartialResult()`: Returns the partial result of the parallel aggregation from each member.
3. `combine(partialResult)`: Combines partial results returned from each cluster member into the final result. In this method, the partial result is computed by using one or more attributes from `entry`. This method will be called multiple times on the root `Aggregator` instance, once for each cluster member's partial result.
4. `finalizeResult()`: Calculates and returns the final result of the aggregation.

Writing an Aggregator

To define an `Aggregator` that returns the key of the oldest `Person`, create the `main.js` file and add the following content:

```
export class OldestPerson {
  constructor() {
    this.key = -1;
    this.age = 0;
  }

  accumulate(entry) {
    // Compare this entry's age with the result computed so far.
    if (entry.value.age > this.age) {
      this.key = entry.key;
      this.age = entry.value.age;
    }
    return true;
```

```
  }

  getPartialResult() {
    // Return the partial result accumulated / computed so far.
    return JSON.stringify({key:this.key, age:this.age});
  }

  combine(partialResult) {
    // Compute a (possibly) new result from the previously computed
    // partial result.
    let p = JSON.parse(partialResult);

    if (p.age > this.age) {
      this.key = p.key;
      this.age = p.age;
    }
    return true;
  }

  finalizeResult() {
    // Return the final computed result.
    return this.key;
  }
}
```

Finally, re-export the classes defined from the `Aggregator` module, `main.mjs`:

```
export * from "./aggregators.mjs"
```

Installing and Using Dependencies

In this section, you'll see how you can use third-party dependencies when implementing your JavaScript classes, which often may be the reason to use JavaScript in the first place.
You can use the standard approach of using `npm install` to install your dependencies and use the `import` statement function to consume them. For example, to install the `lodash-es` and `@paravano/utils` packages, use the following commands:

```
cd ${JS_DIR}
npm install -s lodash-es
npm install -s @paravano/utils
```

After the dependencies are installed, you can implement an entry processor that uses the `now` function from the `lodash-es` package, and `camel` function from the `@paravano/utils` package:

```
import {now} from 'lodash-es';
import {camel} from "@paravano/utils";

export class CamelCase {
  process(entry) {
    console.log(`> CamelCase: entry=${entry} time=${now()}`)
    entry.value = camel(entry.value);
    return entry.value;
  }
}
```

Building a Project

To make your JavaScript classes available to Coherence Polyglot Framework at runtime, you need to package them in a JAR file.

Coherence Polyglot Framework will automatically load all scripts with an `.mjs` extension from any `scripts/js` directory in the class path, so in the simplest scenario, when you have no external dependencies, you can simply configure your Maven project to copy all of your source files to the correct location within the target directory, which will ensure that they are packaged into the final JAR along with all other classes and resources:

```
<build>
  <resources>
    <resource>
      <directory>${project.basedir}/src/main/resources</directory>
    </resource>
    <resource>
      <directory>${project.basedir}/src/main/js</directory>
      <targetPath>${project.build.outputDirectory}/scripts/js</targetPath>
      <includes>
        <include>**/*.mjs</include>
      </includes>
    </resource>
  </resources>
</build>
```

Note that if you do this, then you also need to redefine the default resource, so all the files from `src/main/resources` get copied to the output directory as well, like they typically would.

However, when you have external dependencies, and often you likely will, you need to use a bundler to bundle both your source code and any code from the external dependencies it depends on, into the output directory.

The bundler we recommend for this task is esbuild, because it is fast and very simple to use. So now, you'll install it and use it to bundle your JavaScript project into the output directory:

1. Install esbuild.

   ```
   npm install --save-dev esbuild
   ```

2. Configure the build script within `package.json` to run esbuild against your main module.

   ```
   {
     "name": "coherence-js-app",
     "version": "1.0.0",
     "main": "main.mjs",
     "scripts": {
       "build": "esbuild main.mjs --bundle --format=esm --charset=utf8 --
   outdir=../../../target/classes/scripts/js/ --out-extension:.js=.mjs"
     },
     "devDependencies": {
       "esbuild": "^0.24.0"
     }
   }
   ```

Finally, when using a bundler and third-party dependencies, you need to configure an excellent Frontend Maven Plugin to:

1. Install Node.js and npm locally.
2. Install dependencies by running `npm install`.
3. Bundle your scripts by running esbuild using the `npm run build` command, which was defined previously.

This all can be easily accomplished by adding the following plug-in configuration to the `pom.xml` file:

```
<plugin>
  <groupId>com.github.eirslett</groupId>
  <artifactId>frontend-maven-plugin</artifactId>
  <executions>
    <execution>
      <id>install node and npm</id>
      <goals>
        <goal>install-node-and-npm</goal>
      </goals>
    </execution>
    <execution>
      <id>npm install</id>
      <goals>
        <goal>npm</goal>
      </goals>
      <configuration>
        <arguments>install</arguments>
      </configuration>
    </execution>
    <execution>
      <id>npm run build</id>
      <goals>
        <goal>npm</goal>
      </goals>
      <configuration>
        <arguments>run build</arguments>
      </configuration>
    </execution>
  </executions>
  <configuration>
    <nodeVersion>v20.17.0</nodeVersion>
    <installDirectory>target</installDirectory>
    <workingDirectory>src/main/js</workingDirectory>
  </configuration>
</plugin>
```

If you now run `mvn install` in your project directory, you should see that Node.js and npm are installed, and that your scripts and their dependencies are bundled, using esbuild, into a single `target/classes/scripts/js/main.mjs` file.

Now, you are ready to consume classes defined in, and exported from, that module in your Java application.

Using JavaScript Classes in a Java Application

In this section, you will complete the polyglot example by writing a Java application that uses the JavaScript classes that you created previously.
This section includes the following topics:

- Adding Runtime Dependencies
- Implementing the Java Application
- Invoking JavaScript Objects from Java

Adding Runtime Dependencies

To use the JavaScript classes that you created previously from your Java application, you need to add dependencies on Coherence itself, and on the GraalVM Polyglot Runtime, and JavaScript language implementation to the `pom.xml` file:

```
<dependencies>
  <dependency>
    <groupId>com.oracle.coherence</groupId>
    <artifactId>coherence</artifactId>
    <version14.1.2-0-0</version>
  </dependency>

  <!-- GraalVM Polyglot support -->
  <dependency>
    <groupId>org.graalvm.polyglot</groupId>
    <artifactId>polyglot</artifactId>
    <version>23.1.4</version>
  </dependency>
  <dependency>
    <groupId>org.graalvm.js</groupId>
    <artifactId>js-language</artifactId>
    <version>23.1.4</version>
  </dependency>
</dependencies>
```

Implementing the Java Application

You are now ready to implement the Java application that will start the Coherence cluster, populate the people map with a few entries, and use the JavaScript classes you created previously to query and modify entries in the people map.

Create a file called `Main.java` within the main project containing the following code:

```
package com.oracle.coherence.example.js;

import com.tangosol.net.Coherence;
import com.tangosol.net.NamedMap;

import com.tangosol.util.Aggregators;
import com.tangosol.util.Filters;
import com.tangosol.util.Processors;

import com.tangosol.util.filter.AlwaysFilter;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;
import java.util.stream.Collectors;

public class Main
    {
    public static void main(String[] args)
        {
        System.setProperty("coherence.log.level", "1");

        try (Coherence coherence = Coherence.clusterMember().start().join())
            {
            NamedMap<Integer, Person> people = coherence.getSession().getMap("people");
            populatePeople(people);

            displayAllTeens(people);
            convertLastNameToUppercase(people);
            displayOldestManAndWoman(people);
            }
        }

    private static void populatePeople(NamedMap<Integer, Person> people)
```

```
    {
    printHeader("populatePeople");

    Map<Integer, Person> map = new HashMap<>();
    map.put(1, new Person("Ashley", "Jackson", "F", 84));
    map.put(2, new Person("John", "Campbell", "M", 36));
    map.put(3, new Person("Jeffry", "Trayton", "M", 95));
    map.put(4, new Person("Florence", "Campbell", "F", 35));
    map.put(5, new Person("Kevin", "Kelvin", "M", 15));
    map.put(6, new Person("Jane", "Doe", "F", 17));

    people.putAll(map);
    print(people);
    }

private static void displayAllTeens(NamedMap<Integer, Person> people)
    {
    }

private static void convertLastNameToUppercase(NamedMap<Integer, Person> people)
    {
    }

private static void displayOldestManAndWoman(NamedMap<Integer, Person> people)
    {
    }

private static void printHeader(String header)
    {
    System.out.println();
    System.out.println(header);
    System.out.println("-".repeat(header.length()));
    }

private static void print(NamedMap<Integer, Person> map)
    {
    print(map.entrySet(AlwaysFilter.INSTANCE));
    }

private static void print(Set<Map.Entry<Integer, Person>> entrySet)
    {
    TreeMap<Integer, Person> map = new TreeMap<>();
    for (Map.Entry<Integer, Person> e : entrySet)
        {
        map.put(e.getKey(), e.getValue());
        }
    for (Map.Entry<Integer, Person> e : map.entrySet())
        {
        System.out.printf("%d: %s%n", e.getKey(), e.getValue());
        }
    }
}
```

Invoking JavaScript Objects from Java

As the last remaining step, you must implement the previous three methods with the empty bodies, which will use JavaScript classes that you implemented earlier.

This section includes the following topics:

- Invoking JavaScript Filters

Invoking JavaScript Filters

A `Filter` written in JavaScript can be instantiated by calling the `script()` method in the `com.tangosol.util.Filters` utility class. It is defined as:

```
public static <V> Filter<V> script(String language,
                                   String filterName,
                                   Object... args);
```

The first argument is used to specify the language in which the script is implemented. For JavaScript, use `js`. The second argument is used to specify the `exported` name of the `Filter`, and the last, optional argument, can be used to pass constructor arguments to the filter, in case it has any.

Add the following method to `Main.java`:

```
private static void displayAllTeens(NamedMap<Integer, Person> people)
    {
    printHeader("displayAllTeens");

    print(people.entrySet(Filters.script("js", "IsTeen")));
    }
```

Making Parallel Updates Using JavaScript EntryProcessor

An `EntryProcessor` written in JavaScript can be instantiated by calling the `script()` method in the `com.tangosol.util.Processors` class.

Add the following methods to `Main.java`:

```
private static void convertLastNameToUppercase(
                                NamedMap<Integer, Person> people)
    {
    printHeader("convertLastNameToUppercase");

    people.invokeAll(Processors.script("js", "UpperCaseProcessor"));
    print(people);
    }
```

Running the JavaScript Aggregator

An `Aggregator` written in JavaScript can be instantiated by calling the `script()` method in the `com.tangosol.util.Aggregators` class.

Use the `OldestPerson` aggregator, along with the built in filters, to find the oldest male and female persons.

Add the following to `Main.java`:

```
private static void displayOldestManAndWoman(NamedMap<Integer, Person> people)
    {
    printHeader("displayOldestManAndWoman");

    Integer oldestManId   = people.aggregate(
                                Filters.equal(Person::getGender, "M"),
                                Aggregators.script("js", "OldestPerson"));
```

```
    Integer oldestWomanId = people.aggregate(
                                Filters.equal(Person::getGender, "F"),
                                Aggregators.script("js", "OldestPerson"));

    System.out.printf("%d: %s%n", oldestManId, people.get(oldestManId));
    System.out.printf("%d: %s%n", oldestWomanId, people.get(oldestWomanId));
    }
```

Now, you can run the Java application, and if everything is implemented correctly, see the following output:

```
Oracle Coherence Version 14.1.2.0.0 (dev-aseovic) Build 0
 Grid Edition: Development mode
Copyright (c) 2000, 2024, Oracle and/or its affiliates. All rights reserved.


populatePeople
--------------
1: Person[lastName=Jackson, firstName=Ashley, age=84, gender=F]
2: Person[lastName=Campbell, firstName=John, age=36, gender=M]
3: Person[lastName=Trayton, firstName=Jeffry, age=95, gender=M]
4: Person[lastName=Campbell, firstName=Florence, age=35, gender=F]
5: Person[lastName=Kelvin, firstName=Kevin, age=15, gender=M]
6: Person[lastName=Doe, firstName=Jane, age=17, gender=F]

displayAllTeens
---------------
5: Person[lastName=Kelvin, firstName=Kevin, age=15, gender=M]
6: Person[lastName=Doe, firstName=Jane, age=17, gender=F]

convertLastNameToUppercase
--------------------------
1: Person[lastName=JACKSON, firstName=Ashley, age=84, gender=F]
2: Person[lastName=CAMPBELL, firstName=John, age=36, gender=M]
3: Person[lastName=TRAYTON, firstName=Jeffry, age=95, gender=M]
4: Person[lastName=CAMPBELL, firstName=Florence, age=35, gender=F]
5: Person[lastName=KELVIN, firstName=Kevin, age=15, gender=M]
6: Person[lastName=DOE, firstName=Jane, age=17, gender=F]

displayOldestManAndWoman
------------------------
3: Person[lastName=TRAYTON, firstName=Jeffry, age=95, gender=M]
1: Person[lastName=JACKSON, firstName=Ashley, age=84, gender=F]
```

If you run into any issues, the example project for this chapter will be available on GitHub.

Part VIII

Building Upgradable Coherence Applications

Successfully upgrading a Coherence application requires careful consideration that must begin during the initial design of the application. It is too late to think about these considerations after the application is in production and the development team is ready to deploy the next version.

In the simplest upgrade scenario, an application consists of a single Coherence cluster where all of the cluster members are shut down and upgraded together and Coherence features such as persistence and federation are not used. In this scenario, you only need to build and test the application before deploying it to production and you do not need to consider backwards compatibility issues.

More complex upgrade scenarios occur when it is not feasible to shutdown and upgrade all parts of the application simultaneously. The complexity of the upgrade process will also depend upon which Coherence features are being used, whether the application is required to support zero-downtime updates, or if parts of the application need to remain on older versions.

> **Note:**
>
> These considerations also apply if you need to downgrade an application to a previous version. If a serious issue is discovered in an in-production application, the only solution may be to revert the changes and downgrade to an earlier version. It may even require a zero-downtime downgrade.

Upgrade Scenarios

Upgrading an application can range from simple single deployments to significantly more complex scenarios, for example:

- A zero downtime upgrade where the application remains running while parts of it are upgraded.
- An application that uses Coherence persistence and will reload data persisted by the old application.
- An application that uses Coherence federation, and the upgraded cluster will federate data to a cluster running the old version of the application, or the new version will receive federated data from a cluster running the old version.
- The application consists of multiple parts that share a Coherence cluster, and these parts have a different development life cycle and are upgraded at different times. These parts could be deployed as other cluster members, as Coherence Extend clients, or as Coherence gRPC clients. In this case, the application APIs and application data must remain compatible between all the parts.

> **Note:**
>
> Coherence does not support rolling upgrades of cluster members on different major Coherence versions. A major Coherence commercial version is where any of the first four parts of the version number differ. For example, you cannot upgrade from 12.2.1.4.x to 14.1.1.0.x, nor from 14.1.1.0 to 14.1.2.0. You can move between patch numbers, that is, 14.1.2.0.x to another 14.1.2.0.x version.
>
> Upgrades that change the Java major version are also not supported for cluster members because Java does not guarantee serialization compatibility between major releases. For example, you can move from Java 17.1 to 17.2 but not from Java 17 to Java 21.

Part VIII contains the following chapters:

- Upgrade Considerations
  Coherence has many features, several of which need to be implemented correctly to ensure an application can be successfully upgraded.

49
Upgrade Considerations

Coherence has many features, several of which need to be implemented correctly to ensure an application can be successfully upgraded.

This chapter includes the following sections:

- Application Recompilation
  Whenever an application upgrades its Coherence version (even for a minor patch update), it is recommended to recompile the application code against the new Coherence version.
- Serialization
  Every class that will be sent over the network between different JVMs in the application must be serializable, using either the default Coherence serialization (Java) or POF serialization. This applies to both cluster members and Coherence client processes.
- Cache Keys and Values
  You must take into consideration how changes to cache keys and values may affect serialization.
- Entry Processors
  If an application uses custom entry processors, then these need to be compatible across versions. This includes serialization evolvability, ideally using POF serialization.
- Filters
  If an application uses custom filter implementations, they must be compatible across versions. This includes serialization evolvability, ideally using POF serialization.
- Aggregators
  If an application uses custom aggregators, then these should be compatible across versions. This includes serialization evolvability, ideally using POF serialization.
- Value Extractors
  If an application uses custom extractors, these should be serialization compatible. Adding or removing fields must be done in an evolvable way, ideally using POF evolvability.
- Use of Lambdas
  Lambdas use in Coherence APIs must be treated like any other class in a Coherence application. Typically, a lambda used as a Coherence API method parameter means the lambda will be serialized and sent to the server to be executed.
- Topics
  Coherence topics, like caches, store serialized data on the server. The classes used for values published to topics should be serialization compatible and evolvable.
- Cache Loaders and Cache Stores
  Applications that use cache loaders or cache stores to access an external data source must still be able to load data from and write data to the external data source.
- Coherence Clients
  Client application code (both Extend and gRPC) must be written to have downward and upward compatible functionality. Although Coherence itself guarantees clients can connect to Coherence clusters on different versions (including different major versions), application code must be written to support this.

- Cache Configuration Changes
  There are a number of things that can be changed in a cache configuration file as part of an upgrade. There are also things that cannot be changed without a full cluster shutdown.
- Operational Configuration Changes
  There are a number of Coherence configuration items that can be changed in the Operational Configuration file (or override file) some of which can be changed in a rolling update and some cannot.
- Security and SSL/TLS
  Security in Coherence has a number of features:
- Persistence
  Applications that use Coherence persistence should ensure that the classes used for cache keys and values are serialization compatible and evolvable.
- Federation
  Federation copies data between different Coherence clusters. The classes used in federated cache keys and values must be evolvable across the versions so that data can successfully be sent between different clusters.
- Executor Service
  The executor service that is part of the Coherence Concurrent module executes tasks remotely in the Coherence cluster. These tasks are serialized and sent to other cluster members for execution, so they must be evolvable like any other class sent over the network in Coherence.

Application Recompilation

Whenever an application upgrades its Coherence version (even for a minor patch update), it is recommended to recompile the application code against the new Coherence version.

Ideally, recompiling the application should also include a full continuous integration build and test run. Coherence APIs are guaranteed to be downward-compatible for patch releases, but they may not be binary compatible. This means that an application should not require code changes for a minor Coherence patch upgrade, but it may require recompilation to work correctly.

Application developers should always consult the release notes when upgrading Coherence, they should especially check for deprecations so that application code can be changed to use alternatives. Deprecated code is not removed in a patch but may be removed in a major Coherence upgrade.

Serialization

Every class that will be sent over the network between different JVMs in the application must be serializable, using either the default Coherence serialization (Java) or POF serialization. This applies to both cluster members and Coherence client processes.

When upgrading to a new version of an application where classes have changed, there are a number of rules to consider depending on how the classes are used, which are detailed in the following sections. For example, Cache Keys and Values.

Although Java serialization is the default serializer used in Coherence, application that require seamless upgrades should be using Coherence POF instead. Java serialization is not downward-compatible without significant effort and is not properly evolvable. POF, on the other hand, can be written to be down- and upward-compatible, and is fully evolvable as well as platform independent.

You cannot change the serializer used by a cache service in a rolling upgrade. For example, an application cannot change from Java serialization to POF serialization.

- Adding or Removing Classes
  Take care when adding or removing classes during upgrades.
- Enum Classes
  If you use enum types in applications, then the same rules that apply to normal classes also apply to enum classes. plus some additional considerations.

Adding or Removing Classes

Take care when adding or removing classes during upgrades.

If a new class is added that can be serialized and sent over the network to another JVM in the application, then you must ensure that this class will not be sent to any JVM that does not have the new class during the rolling upgrade.

If a class is removed that can be serialized and sent over the network to another JVM in the application, then you must ensure that this class will not be sent to any new version JVM that does not have the new class during the rolling upgrade.

A general solution for this is to upgrade in two phases, if possible:

1. In the first upgrade, add the new classes but do not add any functionality that uses those classes.
2. In the second upgrade, add the rest of the code.

For example, in an application where the application code runs in storage disabled cluster members, or Extend clients, and the storage enabled members are separate, you would upgrade the storage enabled members first, because no application code would be currently using the new classes, then you would upgrade the application members.

Enum Classes

If you use enum types in applications, then the same rules that apply to normal classes also apply to enum classes. plus some additional considerations.

Do not change the order of the enum values between versions as some code may rely on the enumeration values ordinal, which will change if the values are re-ordered.

For example, if the Day enum began with SUNDAY (as in the following example):

```
public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
}
```

Then, moving SUNDAY to the end of the values (as in the following example) might result in a breaking change:

```
public enum Day {
    MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY, SUNDAY
}
```

You should also not remove existing values for the enum type because one of those values may be received from a Coherence member still running the old version of the application. Removing a value will also change the ordinals of any remaining values, which will be a breaking change.

You should take care when adding new values to an enum. The new values must always be added to the end of the list of values for the enum. Existing applications will not be able to deserialize the new enum value so it is important that all Coherence members have been upgraded to the new code before the application starts to use the new value.

Cache Keys and Values

You must take into consideration how changes to cache keys and values may affect serialization.

Key Classes

Changes can be made to cache key classes as long as their serialized form remains the same, and their `equals` and `hashCode` methods are compatible. Instances of both the new version of a key class and the old version of that class that represent the same cache key value must serialize to the equivalent Coherence Binary value.

See Using Portable Object Format .

Fields can be added if:

- The new fields are transient (that is, they are not included in the serialized binary representation of the key)
- The new fields are not part of the `equals` and `hashCode` methods.

If using POF, the POF identifier for the key class cannot be changed, that is, the value used for the `type-id` element in the POF configuration must remain the same across versions. The POF identifier is stored as part of the serialized binary value, so changing it would change the serialized form of the key data.

Additionally, if using POF, you also cannot change the POF identifier for fields in the key. The identifier for each field is stored as part of the serialized binary value, so changing it would change the serialized form of the key data.

Methods can be added to or removed from a key class, as these do not affect the serialized form of the key.

Value Classes

Classes used for cache values should be written to be evolvable. Upward and downward serialization evolvability can be achieved by using POF serialization.

Serialization compatibility is not only important for the key and value classes themselves, but also the types of any non-transient fields in keys and values must also be compatible.

Entry Processors

If an application uses custom entry processors, then these need to be compatible across versions. This includes serialization evolvability, ideally using POF serialization.

An entry processor must return the same result type as the previous version (or a compatible sub class of that type). An entry processor may be executed during an upgrade, so the caller on the old version of the application would expect to receive the same result type.

If the new version of an entry processor introduces new fields, the code in the process method must allow for those fields to be null (or some other default set during deserialization). If an old version of the application executed the entry processor, then it will not have been able to set those new fields, so if the processor executes on a storage member with the new version of the class, the fields will be missing when deserialized.

A new version of an entry processor should still populate fields of the old version if those fields are mandatory for the old version to execute without failure.

Introducing a new entry processor class can only be done if it can be guaranteed that the processor will not be executed until all the storage enabled cluster members have been upgraded. If a new version of the application tries to invoke an entry processor and that invoke call targets a storage member running the old version of the code, then the caller will receive an exception with a root cause of `ClassNotFoundException`.

Removing an existing entry processor class from an application should only be done when it can be guaranteed no parts of the application will try to invoke that entry processor.

Filters

If an application uses custom filter implementations, they must be compatible across versions. This includes serialization evolvability, ideally using POF serialization.

Filters can be used to register for events using `MapListener` so they must have a suitable implementation of `equals` and `hashCode`. If these methods change so that a new implementation does not equal an old implementation, then the application may not receive the correct events or may receive no events.

If a new filter implementation introduces new fields, the filter methods must be able to execute when that field is not set. If an old version of the application executes a method using the old version of the filter, when it is deserialized for execution on a cluster member running the new version, those fields will not be set. The new version should allow for this or should set suitable default values in its constructor or when deserialized.

Introducing new filter classes can only be done if it can be guaranteed that the new filter class will not be used until all of the storage enabled cluster members have been upgraded. If a new version of the application tries to execute a cache query, add a `MapListener`, or any other operation involving the new filter class, that operation will fail if it executes on a cluster member that is not yet upgraded.

Custom filter implementations can only be removed once it can be guaranteed that no existing versions of the application will execute methods requiring those filters during the upgrade.

Aggregators

If an application uses custom aggregators, then these should be compatible across versions. This includes serialization evolvability, ideally using POF serialization.

The new version of the aggregator should also return the same result type as previous versions. If an application code executing the old version of the application invokes the aggregator and the aggregator runs on a storage member running the new version, the calling code will expect to receive the same result type (or a compatible sub-class of that result type).

The aggregators partial result type must also remain compatible across versions. As the aggregator runs the reduce phase where the results from different members are collected and reduced to a single result, the member running the reduction could receive partial results from storage members running different versions of the aggregator.

If the new version of an aggregator class introduces new fields, the aggregator must still work if those fields are not set during deserialization, or it should set suitable default values in the constructor or when deserialized. If during upgrade the aggregator is invoked from an old version of the application and executes on a storage member running the new version, the new fields will not be present when deserializing.

Introducing a new aggregator class can only be done if it can be guaranteed that the aggregator will not be executed until all the storage enabled cluster members have been upgraded. If a new version of the application tries to invoke an aggregator and that invoke call targets a storage member running the old version of the code, then the caller will receive an exception with a root cause of `ClassNotFoundException`.

Custom aggregator implementations can only be removed once it can be guaranteed that no existing versions of the application will execute methods requiring those aggregators during the upgrade.

Value Extractors

If an application uses custom extractors, these should be serialization compatible. Adding or removing fields must be done in an evolvable way, ideally using POF evolvability.

Extractors that are used to extract a type to be returned to a calling application should return the same type as the previous version.

Value extractors are used to define indexes in Coherence. A specific index is identified from a Map keyed by the value extractor. If the implementation of a value extractor changes, such that it is not correctly equal to the previous version, then this could cause indexes not to be used or maybe even the wrong index could be used.

Introducing a new value extractor class can only be done if it can be guaranteed that the value extractor will not be executed until all the storage enabled cluster members have been upgraded. If a new version of the application tries to invoke a value extractor and that invoke call targets a storage member running the old version of the code, then the caller will receive an exception with a root cause of `ClassNotFoundException`.

Custom value extractor implementations can only be removed once it can be guaranteed that no existing versions of the application will execute methods requiring those value extractors during the upgrade.

Use of Lambdas

Lambdas use in Coherence APIs must be treated like any other class in a Coherence application. Typically, a lambda used as a Coherence API method parameter means the lambda will be serialized and sent to the server to be executed.

Coherence uses lambdas in two ways, dynamic lambdas and static lambdas. For dynamic lambdas, both the lambda state and the code are serialized and sent to the server to be executed. For static lambdas, only the state is serialized and the lambda code must exist on the server already.

To fully support rolling upgrades, you should use dynamic lambdas. See Processing Entries Using Lambda Expressions.

Topics

Coherence topics, like caches, store serialized data on the server. The classes used for values published to topics should be serialization compatible and evolvable.

During a rolling upgrade, a topic will continue to work without loss of published elements. If the upgrade involves upgrading application JVMs containing topic subscribers, then this will cause uncommitted elements to be resent to new subscribers as the old application members are shutdown. Coherence guarantees at least one delivery, so application code that processes messages should alway be written to be idempotent to allow for messages to be received more than once. See Using Portable Object Format .

Cache Loaders and Cache Stores

Applications that use cache loaders or cache stores to access an external data source must still be able to load data from and write data to the external data source.

Writing evolvable database schemas is far beyond the scope of this documentation, but if a database schema is updated as part of an application upgrade, there may still be cluster members running the old version of the code trying to read from or write to the new database schema.

Cache stores are application code and hence can interact with anything, not just a database, so any changes to the external data sources used by cache stores must be downward and upward compatible.

Coherence Clients

Client application code (both Extend and gRPC) must be written to have downward and upward compatible functionality. Although Coherence itself guarantees clients can connect to Coherence clusters on different versions (including different major versions), application code must be written to support this.

Client application code must also be written to survive disconnection, which will happen during a rolling upgrade. Coherence will reconnect a client automatically on the next request during a rolling upgrade, but any request that was in-flight when the client was disconnected may throw an exception on the client. In-flight requests may actually complete on the server, so applications that re-try failed requests must make sure these requests are idempotent, that is, they can safely be replayed more than once. For example, a `put` request from a client that is in progress on the storage member when the proxy is killed may throw an exception on the client, but the `put` will succeed on the storage member. If the client retries the `put`, the `put` will happen twice. This may be fine, but side-effects of the `put` will occur twice, for example, applications that perform other processing based on events would receive two events for that entry.

Cache Configuration Changes

There are a number of things that can be changed in a cache configuration file as part of an upgrade. There are also things that cannot be changed without a full cluster shutdown.

A few examples to consider:

- Adding Caches - A new version of an application may require a new cache to be created. If the cache configuration file in the old version does not have a mapping for the new cache, the upgrade could fail.
- Removing Caches - If a cache mapping is removed, then this can cause an exception when a cluster member using the new version of the configuration receives a cache `create` request from a member running the old version of the application.
- Change Cache Types - It is not always possible to change the type of a cache in a rolling upgrade. For example, you cannot change from a legacy replicated cache to a distributed cache. Some cache types can be changed, for example, it would be possible to change to use a near cache on a new version of the application, as this is local to the new JVM.

Adding Caches

If a new cache is added to a cache configuration file and that cache mapping maps to a totally new scheme and service name for the new version of the application, then this will work in a rolling upgrade. Only cluster members with the new configuration will start the new service and have the new cache. Existing members will not receive requests for the new cache.

The problem occurs when a new cache is added to the application and that maps to an existing service. When application code requests the new cache, the request to create that cache will go to all members running the same cache service. This will include members running the old version but they will have no mapping for that cache and therefore will throw an exception.

To make adding or removing caches easier, use wildcard names instead of fixed cache names in the cache configuration file.

In Example 49-1, the cache configuration file has mappings that are inflexible and difficult to upgrade. It only supports two cache names: `foo` and `bar`. If an application needs to add another cache where storage enabled members use this configuration, then the upgrade will not work.

Example 49-1 A cache configuration file with fixed mappings

```
<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>foo</cache-name>
      <scheme-name>storage</scheme-name>
    </cache-mapping>

    <cache-mapping>
      <cache-name>bar</cache-name>
      <scheme-name>storage</scheme-name>
    </cache-mapping>
</caching-scheme-mapping>
```

In Example 49-2, the cache configuration file uses wildcard mappings. It has a single mapping that uses a wildcard character `*` to map cache names to the scheme named `storage-scheme`. Storage enabled members with this configuration support any cache name.

Example 49-2 A cache configuration file using wildcard mappings

```
<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>storage-scheme</scheme-name>
    </cache-mapping>
</caching-scheme-mapping>
```

However, in applications where caches have to map to different schemes with different configurations, a single wildcard mapping is not enough. In this case an application would need to use wild card mappings with prefixes.

In Example 49-3, the cache configuration file has two mappings for `foo-*` and `bar-*`. This means any cache name that starts with `foo-` will map to the `foo-storage` scheme, so `foo-`, `foo-1`, `foo-abc`, and so on. The same applies for cache names prefixed with `bar-`. As long as a new version of the application introduces new cache names that match one of the existing mapping names, then the upgrade will work.

Example 49-3 A cache configuration file using wildcard mappings with prefixes

```
<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>foo-*</cache-name>
      <scheme-name>foo-storage</scheme-name>
    </cache-mapping>

    <cache-mapping>
      <cache-name>bar-*</cache-name>
      <scheme-name>foo-storage</scheme-name>
    </cache-mapping>
</caching-scheme-mapping>
```

Removing Caches

Caches cannot easily be removed from a cache configuration during an upgrade.

During the upgrade process, when a cache service on a new members joins the cluster, it receives cache creation messages for all the cache names that exist for that service on the existing cluster senior member. If this includes a cache name that does not have a mapping in the new members cache configuration, then there will be an exception.

If a cache is to be removed from a cache configuration file, that cache must have been destroyed on the existing cluster members before the upgrade starts. The only way to destroy

a cache is in application code. Obviously the destroying the cache on the old cluster members should not cause the existing application to fail, so that version of the application must be able to run without that cache and must not have code that would try to recreate the cache after it is destroyed.

Changing Cache Types

It is not possible to change the basic underlying type of a cache during an upgrade. For example, replicated caches are deprecated, but it is not possible to change a cache from a replicated to a distributed cache without a full cluster shutdown.

The following changes to cache type are allowed in an upgrade:

- NearCache - it should be possible to change a cache from or to a Near Cache, as this is a local configuration on the Coherence JVM.
- ViewScheme - it should be possible to change from or to a view scheme, as this is a local view over a distributed cache.
- ReadWriteBackingMap - changing the backing map for a cache from or to a read/write backing map should be possible with some caveats. If the new version will use a cache store, then only data sent to new members will be written to the external data source during the upgrade, any data changes that happen on existing members will not.
- Backing Map Type Changes - it should be possible to change the type of the backing map configuration in an upgrade. For example, changing from local scheme to Caffeine, or to Elastic Data, or vice versa. The backing map configures how an individual cluster members stores the cache data, so there are no compatibility issues with other members during the upgrade.
- Backing Map Configuration - changing some of the backing map configuration that is only local to that JVM is also possible. For example, changes to high or low units, unit calculator, expiry delay, and so on.

Operational Configuration Changes

There are a number of Coherence configuration items that can be changed in the Operational Configuration file (or override file) some of which can be changed in a rolling update and some cannot.

Security and SSL/TLS

Security in Coherence has a number of features:

- SSL/TLS changes for cluster members
- SSL/TLS changes for Extend and gRPC clients
- Identity assertion
- Storage access authorization

See Introduction to Oracle Coherence Security in *Securing Oracle Coherence*.

SSL/TLS Changes for Cluster Members

Any changes to the SSL/TLS configuration in Coherence must be compatible with the existing cluster members and clients when upgrading.

It is not possible to change cluster communication from non-SSL/TLS to SSL/TLS (or vice-versa) in a rolling upgrade. The new members will not be able to form a cluster with the existing members.

Changing from one-way to two-way authentication is only possible if the existing members can supply a valid certificate to the new members.

Changing hostname verification can only be done if the existing members will still be verified during the upgrade.

See Using SSL to Secure TCMP Communication in *Securing Oracle Coherence*.

SSL/TLS Changes for Extend and gRPC Clients

Similarly to cluster membership, changes to SSL/TLS configuration for Extend or gRPC proxies should be made in such a way that existing clients can still connect to the new proxies. It is not possible to change a proxy from non-SSL/TLS to SSL/TLS (or vice-versa) as existing clients will not be able to connect to the new proxies. One solution would be to introduce new proxies running the changed configuration as well as leaving the existing proxy configurations, so each Coherence server is running two proxies, one SSL/TLS and one non-SSL/TLS. The existing and new clients will be able to connect to the relevant proxy. After upgrade it will be possible to do another upgrade where the old proxy configuration is removed.

Alternatively, if clients can be shut down during the upgrade, then there is no problem with any change to the SSL/TLS configuration on the proxies.

SSL/TLS changes in Extend or gRPC clients should also be done in such a way that the new clients can still connect to the existing proxies, or ensure that new clients are configured to only connect to the new proxies during the upgrade. See Using SSL to Secure Extend Client Communication in *Securing Oracle Coherence*.

Identity Assertion

Coherence Extend can be configured to use identity assertion as a mechanism to pass tokens from the client that are verified on the server when a connection is made. Any changes to the identity assertion code, either on the clients or proxies must be done in a compatible way so that old clients can still connect to new proxies and new clients can connect to old proxies.

The tokens sent by the client are serialized on the client and deserialized on the server. The updated token classes used must be serialization compatible and still be recognized on the existing cluster members.

Removing the use of identity assertion is possible in an upgrade providing it is removed by upgrading the server side proxy servers first.

Alternatively, if clients can be shut down during the upgrade then there is no problem with any change to the identity assertion configuration and code on the proxies. See Securing Extend Client Connections in *Securing Oracle Coherence*.

Storage Access Authorisation

If using the Storage Access Authorization feature to authorize operations on caches on the storage enabled cluster members, any changes must be compatible. See Authorizing Access to Server-Side Operations in *Securing Oracle Coherence*.

Persistence

Applications that use Coherence persistence should ensure that the classes used for cache keys and values are serialization compatible and evolvable.

If a cluster member tries to read persistence files stored by a different version of the application, this may fail if the data is not compatible. The actual issues may not be immediately visible as the serialized binary data will be read from files and loaded into the cache in its serialized binary form, but the application will later fail when it tries to deserialize that data.

For more information, see:

- Using Portable Object Format
- Persisting Caches in *Administering Oracle Coherence*

In particular, consider the following areas when performing a rolling upgrade:

Java Version

If you want servers from different Coherence version to run in the same cluster, then you must use the same major Java version before and after the upgrade.

Serialization Format

The general rules for Java serialization compatibility apply to persistence as well. See Serialization.

If you use Coherence POF serialization and if there are any changes in the cache data classes (keys, values, or both), then you can maintain serialization compatibility through the implementation of the `IEvolvablePortableObject` interface. See Evolvable Portable User Types in *Developing Remote Clients for Oracle Coherence*.

If the serialization format changes, for example, from Java to POF, then you cannot load previously created snapshots or archives.

Changes to Persistence Location

If you want to change the persistence location, then you need to shutdown the server and move or copy all the persistence directories to the new location. Next, upgrade the server and update the persistence configuration to use the new location, before starting the server. The following are possible persistence locations you can change:

- `<active-directory>`
- `<event-directory>`
- `<snapshot-directory>`
- `<trash-directory>`

For example, if you want to change the `oldEnvironment` to the `newEnvironment`, then before starting the upgraded server, you need to copy or move the contents of directory `/oldEnv` to `/newEnv`. Then, update the persistence configuration accordingly.

Example 49-4 shows a persistence configuration where the persistence locations remain set to `/oldEnv`.

Example 49-4 Persistence Locations for `oldEnvironment`

```
<persistence-environments>
   <persistence-environment id="oldEnviornment">
      <persistence-mode>on-demand</persistence-mode>
      <active-directory>/oldEnv</active-directory>
      <snapshot-directory>/oldEnv</snapshot-directory>
      <trash-directory>/oldEnv</trash-directory>
```

```
   </persistence-environment>
</persistence-environments>
```

In Example 49-5, the persistence configuration has been updated to reflect the new persistence location, `/newEnv`.

Example 49-5 Persistence Locations for `newEnvironment`

```
<persistence-environments>
   <persistence-environment id="newEnviornment">
      <persistence-mode>on-demand</persistence-mode>
      <active-directory>/newEnv</active-directory>
      <snapshot-directory>/newEnv</snapshot-directory>
      <trash-directory>/newEnv</trash-directory>
   </persistence-environment>
</persistence-environments>
```

See Creating Persistence Environments in *Administering Oracle Coherence*.

Any distributed cache configuration change rules for rolling upgrade are also applicable to persistence. See Cache Configuration Changes.

Federation

Federation copies data between different Coherence clusters. The classes used in federated cache keys and values must be evolvable across the versions so that data can successfully be sent between different clusters.

For general information on using federation in Coherence, see Federating Caches Across Clusters in *Administering Oracle Coherence*.

In addition to the general considerations for upgrade such as JDK upgrade, Lambda, serialization compatibility, and so on, you must also consider aspects specific to federation.

Upgrading Federated Clusters using a Rolling Update

Where a rolling update is possible, federated clusters can be upgraded on a rolling basis as well. For guidance on performing the upgrade, see Performing a Rolling Restart. Because federation includes multiple Coherence clusters, you should upgrade clusters one at a time, until all clusters are upgraded.

Upgrading Federated Clusters when a Rolling Update is not Feasible

If a rolling update is not feasible, then you must shutdown the old cluster and create a new cluster with the upgrade. Then you can use federation's `replicateAll()` operation to replicate the data from an existing cluster to the upgraded cluster. See FederationManager MBean in *Managing Oracle Coherence*.

Here are the prerequisites for `replicateAll`:

- **Serialization form**: Ensure that the serialization format has not changed and is compatible between the upgrade. See Serialization.
- **Changes to partition count**: Federation tracks changes by partition and can resend changes that may have been missed when a partition migrates from one cluster member to another. While federation between two clusters with differing partition counts is possible, some of the tracking is disabled in this case.

When transitioning a partition count change in federated clusters, use `replicateAll()` to re-populate a restarted cluster, but afterward, take manual steps to verify that all of the cache data was federated to the upgraded cluster. For example, verify the cache sizes after `replicateAll()` has completed.

See Workarounds to Migrate a Persistent Service to a Different Partition Count in *Administering Oracle Coherence*.

Migrating from a Distributed Scheme to a Federated Scheme

Coherence persistence can be used to facilitate migrating a distributed cache service to a federated cache service. See Using Federation in *Administering Oracle Coherence*.

Executor Service

The executor service that is part of the Coherence Concurrent module executes tasks remotely in the Coherence cluster. These tasks are serialized and sent to other cluster members for execution, so they must be evolvable like any other class sent over the network in Coherence.

For more information, see Using Executors.

A

Operational Configuration Elements

The operational configuration reference provides a detailed description of the operational deployment descriptor elements.

> **Note:**
>
> Coherence XML configuration files are validated at runtime using the corresponding XML schemas. It is important that any custom configuration files conform to the schema, because something as simple as having elements in the wrong order will cause validation to fail. As validation errors produced by XML schema validators can be difficult to understand, we recommend that custom configuration files are edited in a development environment capable of validating the XML against the schema as it is being written. This can save a lot of time compared to debugging validation errors at runtime.

This appendix includes the following sections:

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. See Operational Override File.
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/1.3/coherence-operational-config.xsd

The `<coherence>` element is the root element of the operational descriptor and includes the XSD and namespace declarations. 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">
```

Note:

- 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 Operational Configuration Attribute Reference.

Operational Configuration Element Reference

The operational element reference includes all non-terminal operational configuration elements. Each section includes instructions on how to use the element and also includes descriptions for all valid subelements.

- access-controller
- active-active
- active-passive
- address-provider
- address-providers
- authorized-hosts
- cache-factory-builder-config
- callback-handler
- central-replication
- cluster-config
- cluster-quorum-policy
- coherence

- configurable-cache-factory-config
- custom-topology
- federation-config
- flashjournal-manager
- flow-control
- group
- groups
- host-range
- hub-spoke
- identity-asserter
- identity-manager
- identity-transformer
- incoming-message-handler
- init-param
- init-params
- instance
- interceptor
- interceptors
- journaling-config
- key-store
- license-config
- logging-config
- management-config
- mbean
- mbeans
- mbean-filter
- member-identity
- multicast-listener
- name-service-addresses
- notification-queueing
- outgoing-message-handler
- outstanding-packets
- packet-buffer
- packet-bundling
- packet-delivery
- packet-publisher
- packet-size
- packet-speaker

- participant
- participants
- participant-destination
- participant-destinations
- password-provider
- password-providers
- pause-detection
- persistence-environment
- persistence-environments
- provider
- ramjournal-manager
- remote-addresses
- reporter
- security-config
- serializer
- serializers
- service
- service-guardian
- services
- shutdown-listener
- snapshot-archivers
- socket-address
- socket-provider
- socket-providers
- ssl
- storage-authorizer
- storage-authorizers
- tcp-ring-listener
- topology-definitions
- tracing-config
- traffic-jam
- trust-manager
- unicast-listener
- volume-threshold
- well-known-addresses

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-1 describes the subelements of the `access-controller` element.

Table A-1 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 Using an Access Controller in *Securing Oracle Coherence*. The default value is `com.tangosol.net.security.DefaultController`.

Table A-1 (Cont.) access-controller Subelements

Element	Required/ Optional	Description

<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 keystore file, permissions description file, the log flag, and the keystore password specified as follows:
		
		`<init-params>`
		`  <init-param id="1">`
		`  <param-type>java.io.File</param-type>`
		`  <param-value system-property="coherence.security.keystore"></param-value>`
		`  </init-param>`
		`  <init-param id="2">`
		`  <param-type>java.io.File</param-type>`
		`  <param-value system-property="coherence.security.permissions">`
		`  </param-value>`
		`  </init-param>`
		`  <init-param id="3">`
		`  <param-type>java.lang.Boolean</param-type>`
		`  <param-value system-property="coherence.security.log"></param-value>`
		`  </init-param>`
		`  <init-param id="4">`
		`  <param-type>java.lang.String</param-type>`
		`  <param-value system-property="coherence.security.keystore.password"></param-value>`
		`  </init-param>`
		`</init-params>`
		
		The preconfigured system property overrides based on the default AccessController implementation and the default parameters as specified above are coherence.security.keystore, coherence.security.permissions, coherence.security.log, and coherence.security.keystore.password. At a minimum, parameter 1 and 2 should be specified. Parameter 3 and 4 are optional and have a default value of false and null, respectively. See init-param.
		
		In addition, <access-controller> can use the <password-provider> element to supply the keystore password.
		
		Following is an example of <access-controller> using <password-provider>:
		
		`<access-controller>`
		`  <class-name>com.tangosol.net.security.DefaultController</class-name>`
		`  <init-params>`
		`  <init-param id="1">`
		`  <param-type>java.io.File</param-type>`
		`  <param-value>keystoreRsa.p12</param-value>`
		`  </init-param>`
		`  <init-param id="2">`
		` <param-type>java.io.File</param-type>

Table A-1 (Cont.) access-controller Subelements

Element	Required/ Optional	Description
		<param-value>permissions.xml</param-value> </init-param> </init-params> <password-provider> <class-name>pacakge.MyPassswordProvider</class-name> <init-params> <init-param> <param-name>param_1</param-name> <param-value>private</param-value> </init-param> </init-params> </password-provider> </access-controller> For more information about <password-provider>, see password-provider.

active-active

Used in: topology-definitions.

Description

The active-active element specifies an active-active topology where there are one or more active participants that send data to other active participants. The active participants never re-sends the data. For example:

```
<active-active>
   <name>MyTopology</name>
   <active>ClusterA</active>
   <active>ClusterB</active>
</active-active>
```

Elements

Table A-2 describes the subelements of the active-active element.

Table A-2 active-active Subelements

Element	Required/ Optional	Description
<name>	Required	Specifies a user-defined name for the topology. The name is used to refer to this topology.
<active>	Required	Specifies the name of an active participant in the topology. Multiple active participants can be defined.
<interceptor>	Optional	Specifies the name of a custom participant that is implemented as an interceptor for federated change events. An interceptor participant is a receiver participant only.

active-passive

Used in: topology-definitions.

Description

The `active-passive` element specifies an active-passive topology where there are one or more active participants that send data to other passive participants. The active participants never re-sends the data and the passive participants only receive data. For example:

```
<active-passive>
   <name>MyTopology</name>
   <active>ClusterA</active>
   <passive>ClusterB</passive>
</active-passive>
```

Elements

Table A-3 describes the subelements of the `active-passive` element.

Table A-3 active-passive Subelements

Element	Required/ Optional	Description
`<name>`	Required	Specifies a user-defined name for the topology. The name is used to refer to this topology.
`<active>`	Required	Specifies the name of an active participant in the topology. Multiple active participants can be defined.
`<passive>`	Optional	Specifies the name of a passive participant in the topology. Multiple passive participants can be defined.
`<interceptor>`	Optional	Specifies the name of a custom participant that is implemented as an interceptor for federated change events. An interceptor participant is a receiver participant only.

address-provider

Used in: well-known-addresses, address-providers, remote-addresses, and name-service-addresses.

Description

The `address-provider` element specifies either socket address information (IP, or DNS name, and port) or an implementation of the `com.tangosol.net.AddressProvider` interface. The interface offers a programmatic way to define socket addresses. Many elements make use of the `address-provider` element.

The address-provider Element for Well Known Addresses

Within the `well-known-addresses` element, use the `address-provider` element to specify an address factory that implements the `com.tangosol.net.AddressProvider` interface. The interface offers a programmatic way to define Well Known Addresses (WKA) members. See Using Well Known Addresses. The following example demonstrates using the `address-provider` element within the `well-known-addresses` element:

```
<unicast-listener>
   <well-known-addresses>
```

```
      <address-provider>
         <class-name>package.MyAddressProvider</class-name>
      </address-provider>
   </well-known-addresses>
</unicast-listener>
```

The address-provider Element for Socket Addresses

Within the `address-providers` element, use the `address-provider` element to specify a socket address or an address factory that implements the `com.tangosol.net.AddressProvider` interface. The address provider definitions are referenced by TCP/IP acceptors and TCP/IP initiators which are used to setup Coherence*Extend. A TCP/IP acceptor is also available for memcached clients.

The tcp-acceptor, memcached-acceptor, remote-addresses, and name-service-addresses elements in the cache configuration file reference address provider definitions using the specified `id` attribute value. The following example demonstrates defining address provider definitions within the `<address-providers>` element.

```
<address-providers>
   <address-provider id="ap1">
      <class-name>package.MyAddressProvider</class-name>
   </address-provider>
   <address-provider id="ap2">
      <socket-address>
         <address>192.168.1.3</address>
         <port>9999</port>
      </socket-address>
   </address-provider>
   <address-provider id="ap3">
      <socket-address>
         <address>192.168.1.4</address>
         <port>9999</port>
      </socket-address>
   </address-provider>
</address-providers>
```

The address-provider Element for Persistence

Within the `address-providers` element, use the `address-provider` element to specify an address or an address factory that implements the `com.tangosol.net.AddressProvider` interface. The address provider definitions are used when configuring cache persistence. The addresses represent the storage-enabled hosts in the cluster that are required to recover orphaned partition from the persistent storage, or assign empty partitions if the persistent storage is unavailable or lost. For example:

```
<address-providers>
   <address-provider id="persistence-host-list">
      <address>HOST_NAME/address>
      <address>HOST_NAME</address>
   </address-provider>
</address-providers>
```

The address-provider Element for Federation

Within the `name-service-addresses` element, use the `address-provider` element to specify an address or an address factory that implements the `com.tangosol.net.AddressProvider` interface. The address provider definitions are used when configuring cluster federation. The addresses represent cluster nodes in the remote cluster participant.

Elements

Table A-4 describes the subelements of the `address-provider` element.

Table A-4 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 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	Specifies initialization parameters which are accessible by implementations that include a public constructor with a matching signature. Initialization parameters can be specified when using both the `<class-name>` element and the `<class-factory-name>` element.
<socket-address>	Optional	Specifies the address (IP, or DNS name, and port) to which a socket is bound. This element should only be used when defining socket addresses for a TCP/IP acceptor. This element cannot be used if an address provider implementation is defined using the `<class-name>` or `<class-factory-name>` element.
`<address>`	Optional	Specifies an IP address or DNS name. In the case of a bind address, the address can be represented in CIDR format as a subnet and mask (for example, 192.168.1.0/24), allowing runtime resolution against available local IPs. The bind address can also be an external NAT address that routes to a local address; however, both addresses must use the same port.

address-providers

Used in: cluster-config.

Description

The `address-providers` element contains the declarative data for each address provider.

Elements

Table A-5 describes the subelements of the `address-providers` element.

Table A-5 address-providers Subelements

Element	Required/ Optional	Description
<address-provider>	Optional	Specifies either socket address information or an implementation of the `com.tangosol.net.AddressProvider` interface. Multiple `address-provider` elements can be specified.

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-6 describes the subelements of the `authorized-hosts` element.

Table A-6 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. As of release 14.1.2.0.3 and later, this can be a comma-separated list of IP addresses and/or host names. The default operational configuration file allows setting this value with the system property `coherence.authorized.hosts`.
<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.

> **Note:**
>
> If this value is set, both `<host-address>` and `<host-range>` are superceded by this setting. If a host-filter is provided, then setting system property `coherence.authorized.hosts` has no effect.

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-7 describes the subelements of the `cache-factory-builder-config` element.

Table A-7 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 the `<scope-name>` subelement of the <cache-config> element for details on specifying a scope name within a cache configuration file.

callback-handler

Used in: security-config.

Table A-8 describes the subelements of the `callback-handler` element.

Table A-8 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.

central-replication

Used in: topology-definitions.

Description

The central-replication element specifies a central-replication topology where one or more leaf participants send data to the hub participant, and the hub participant re-sends (repeats) the data to all the other leaf participants. Any change that originate at the hub-participant are sent to all the leaf participants.

The following example demonstrates a custom topology:

```
<central-replication>
   <name>central</name>
   <hub>ClusterA</hub>
   <leaf>ClusterB</leaf>
   <leaf>ClusterC</leaf>
</central-replication>
```

Elements

Table A-9 describes the subelements of the central-replication element.

Table A-9 central-replication Subelements

Element	Required/ Optional	Description
<name>	Required	Specifies a user-defined name for the topology. The name is used to refer to this topology.
<hub>	Required	Specifies the name of the hub participant in the topology.
<leaf>	Required	Specifies the name of a leaf participant in the topology. Multiple leaf participants can be defined.
<interceptor>	Optional	Specifies the name of custom participant that is implemented as an interceptor for federated change events. An interceptor participant is a receiver participant only.

cluster-config

Used in: coherence.

Description

Contains the cluster configuration information, including communication and service parameters.

Elements

Table A-10 describes the subelements of the `cluster-config` element.

Table A-10 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`.
`<lambdas-serialization>`	Optional	Specifies the lambdas serialization mode of `dynamic` or `static`.
<persistence-environments>	Optional	Specifies the declarative data for each persistence environment.
<address-providers>	Optional	Specifies any number of address provider definitions.
<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. You cannot change this element during a rolling restart. This also includes changing of any of the elements within `cluster-quorum-policy`.
<journaling-config>	Optional	Specifies configuration for the journaling subsystem.
<storage-authorizers>	Optional	Contains the declarative data for each storage access authorizer implementation.
<password-providers>	Optional	Contains password provider definitions.

cluster-quorum-policy

Used in: cluster-config.

Description

The `cluster-quorum-policy` element contains quorum policy settings for the Cluster service.

Element

Table A-11 describes the subelements of the `cluster-quorum-policy` element.

Table A-11 cluster-quorum-policy-scheme Subelements

Element	Required/ Optional	Description
`<timeout-site-quorum>`	Optional	Specifies the minimum number of sites that must remain in order 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-site-quorum role="Server">2 </timeout-site-quorum>`
`<timeout-machine-quorum>`	Optional	Specifies the minimum number of machines that must remain in order 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-machine-quorum role="Server">4 </timeout-machine-quorum>`
`<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 any of the cluster quorums 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 any of the cluster quorums 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. 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-12 describes the subelements of the `coherence` element.

Table A-12 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 Configuring JMX Management in *Managing Oracle Coherence*.
<security-config>	Optional	Contains the configuration information for the Coherence Security Framework.
<license-config>	Optional	Contains the edition and operational mode configuration.
<federation-config>	Optional	Contains the federation configuration information that is used to synchronize clusters.

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.ExtensibleConfigurableCacheFactory` 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).

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  eccf= new
```

```
   ExtensibleConfigurableCacheFactory("example-config.xml", loader0);
NamedCache cache = eccf.ensureCache("dist-example", loader0);
cache.put(key, value);

//load the second cache configuration and use a cache

ConfigurableCacheFactory  eccf1= new
   ExtensibleConfigurableCacheFactory("example-config1.xml", loader1);
NamedCache cache1 = eccf1.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-13 describes the subelements of the `configurable-cache-factory-config` element.

Table A-13 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.ExtensibleConfigurableCacheFactory`. The preconfigured system property override is `coherence.cachefactory`.

<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 `coherence.cacheconfig`.

custom-topology

Used in: topology-definitions.

Description

The `custom-topology` element is a free-form topology that consists of a group list. A group consist of participants with specific roles: sender, repeater, or receiver. Only sender and repeater participants can send or forward changes to other participants in the group.

Elements

Table A-14 describes the subelements of the `custom-topology` element.

Table A-14 custom-topology Subelements

Element	Required/ Optional	Description
`<name>`	Required	Specifies a user-defined name for the topology. The name is used to refer to this topology.
<groups>	Required	Specifies the configuration information for any number of group definitions.

federation-config

Used in: coherence.

Description

The `federation-config` element contains the federation configuration information that is used synchronize clusters. The configuration includes defining federation participants and defining a synchronization topology.

Elements

Table A-15 describes the subelements of the `federation-config` element.

Table A-15 federation-config Subelements

Element	Required/ Optional	Description
<participants>	Optional	Specifies any number of remote coherence clusters that are participating in the federation.
<topology-definitions>	Optional	Specifies the topology configuration information that is used to synchronize clusters.

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-16 describes the subelements of the `flashjournal-manager` element.

Table A-16 flashjournal-manager Subelements

Element	Required/ Optional	Description
<minimum-load-factor>	Optional	Specifies the factor of live data below which a journal file is eligible for compaction (garbage collection). Higher values will result in more aggressive compaction at the cost of increased processing time spent in compaction. Valid values are decimal numbers between 0.01 and 0.99. The default is 0.25. **Note**: Do not set this element except under advanced use cases.
<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. **Note:** When <maximum-file-size> / 2 < 64M, <maximum-value-size> is limited by <maximum-file-size> / 2.
<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 and must be large enough so that a file is capable of storing at least 2 values. The default value is 2GB. **Note:** • In most cases, setting <maximum-size> instead of <maximum-file-size> is recommended • <maximum-file-size> and <maximum-size> are mutually exclusive. – When <maximum-file-size> is specified, <maximum-size> = <maximum-file-size> * 512. – When <maximum-size> is specified, <maximum-file-size> = <maximum-size> / 512. – If <maximum-file-size> is smaller than 1M, it is set to 1M. • When <maximum-file-size> / 2 < 64M, <maximum-value-size> is limited by <maximum-file-size> / 2.
<collector-timeout>	Optional	Specifies the amount of time that the journal collector can remain unresponsive prior to considering it timed out. The minimum timeout is 30s. Legal values are strings representing time intervals. The default value is 10m. **Note**: Do not set this element except under advanced use cases.

Table A-16 (Cont.) flashjournal-manager Subelements

Element	Required/ Optional	Description
`<maximum-size>`	Optional	Specifies the maximum capacity for the journal as a memory size. A value of 0 disables flash storage and causes journaling to use only RAM storage and not overflow to flash storage. The preconfigured system property override is `coherence.flashjournal.size`. **Note:** `<maximum-file-size>` and `<maximum-size>` are mutually exclusive. If both are set, the larger of <maximum-size>/512 or <maximum-file-size> are used • When `<maximum-file-size>` is specified, `<maximum-size>` = `<maximum-file-size>` * 512. • When `<maximum-size>` is specified, `<maximum-file-size>` = `<maximum-size>` / 512. • If `<maximum-file-size>` is smaller than 1M, it is set to 1M.
`<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`. **Note**: Do not set this element except under advanced use cases.
`<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`. **Note**: Do not set this element except under advanced use cases.
`<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. The preconfigured system property override is `coherence.flashjournal.dir`.

Table A-16 (Cont.) flashjournal-manager Subelements

Element	Required/ Optional	Description
`<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`. **Note**: Do not set this element except under advanced use cases.
`<tmp-purge-delay>`	Optional	Specifies the amount of time to wait before temporary files that were used by the Journaling subsystem from previous journal manager instances are eligible for removal. The delay begins after the file is last used. The default value is 2 hours. This allows for files stored on a shared mount (SAN, NAS, etc.) with some minor variation in the clocks of various servers creating and accessing the files, specifically enough to account for something like daylight savings. The value can be set to 0 for immediate purge when the directory is not shared by multiple members. The purge delay is for deletion of files from previous journal manager instances only and removal is only done at instance startup. If the value does not contain a unit, a unit of milliseconds is assumed. The preconfigured system property override is `coherence.flashjournal.purgedelay`.
`<high-journal-size>`	Optional	Specifies the soft limit, in bytes, on the journal size or a percentage of the flash journal capacity. The soft limit allows the compaction (garbage collection) thread to tune itself to remove stale values and keep the journal within the soft limit. This is not a hard limit and the journal can still grow up to the maximum file count (`512`). Valid values are sizes (e.g. 11GB) or percentages (e.g. 90%). The default value is 11GB. The preconfigured system property override is `coherence.flashjournal.highjournalsize`. **Note**: Do not set this element except under advanced use cases.
`<writer-timeout>`	Optional	Specifies the amount of time that the flash journal writer can remain unresponsive prior to considering it timed out. The minimum timeout is 30s. Legal values are strings representing time intervals. The default value is `8h`. If a write fails (after the `writer-timeout` values is reached), then the journal becomes read-only until the member is restarted. When the write backlog maximum is reached (as defined by the `async-limit` value), the journal responds to subsequent store requests with an exception either until the write is resolved or indefinitely as the `writer-timeout` is surpassed. **Note**: Do not set this element except under advanced use cases.

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-17 describes the subelements of the `flow-control` element.

Table A-17 flow-control Subelements

Element	Required/ Optional	Description
<enabled>	Optional	Specifies if flow control is enabled. Valid values are `true` or `false`. 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.

group

Used in: groups.

Description

The `group` element specifies a set of participants with a well-defined role. All the participants can receive data irrespective of their roles. The following roles are available:

- sender – A sender participant only sends data that originated in its local cluster.
- repeater – A repeater participant can send data that originated locally as well as data that it received from other participants.
- receiver – A receiver participant never sends data.

The following example demonstrates a custom topology:

```
<custom-topology>
   <name>custom</name>
   <groups>
      <group>
         <sender>ClusterA</sender>
         <sender>ClusterB</sender>
      </group>
      <group>
         <repeater>ClusterA</repeater>
         <receiver>ClusterC</receiver>
      </group>
   </groups>
</custom-topology>
```

Elements

Table A-18 describes the subelements of the `group` element.

Table A-18 group Subelements

Element	Required/ Optional	Description
<sender>	Optional	Specifies a participant that acts as a sender. Multiple sender participants can be defined in a group.
<repeater>	Optional	Specifies a participant that acts as a repeater. Multiple repeater participants can be defined in a group.
<receiver>	Optional	Specifies a participant that acts as a receiver. Multiple receiver participants can be defined in a group.

groups

Used in: custom-topology.

Description

The groups element contains the configuration information for any number of group definitions.

Elements

Table A-19 describes the subelements of the groups element.

Table A-19 groups Subelements

Element	Required/ Optional	Description
<group>	Required	Specifies a groups of participants with well-defined roles in the topology. Any number of groups can be defined.

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-20 describes the subelements of each host-range element.

Table A-20 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.

hub-spoke

Used in: topology-definitions.

Description

The `hub-spoke` element specifies a hub and spoke topology where one or more spoke participants only receive data. The hub participant sends data to all the spoke participants.

The following example demonstrates a hub and spoke topology:

```
<hub-spoke>
   <name>hub</name>
   <hub>ClusterA</hub>
   <spoke>ClusterB</spoke>
   <spoke>ClusterC</spoke>
</hub-spoke>
```

Elements

Table A-21 describes the subelements of the `hub-spoke` element.

Table A-21 hub-spoke Subelements

Element	Required/ Optional	Description
`<name>`	Required	Specifies a user-defined name for the topology. The name is used to refer to this topology.
`<hub>`	Required	Specifies the name of the hub participant in the topology.
`<spoke>`	Required	Specifies the name of a spoke participant in the topology. Multiple spoke participants can be defined.
`<interceptor>`	Optional	Specifies the name of custom participant that is implemented as an interceptor for federated change events. An interceptor participant is a receiver participant only.

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-22 describes the subelements of the `<identity-asserter>` element.

Table A-22 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-23 describes the subelements of the identity-manager element.

Table A-23 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. The <key-store> element cannot be specified if the <key>, <key-loader>, <cert>, or <cert-loader> elements are specified.
<key>	Optional	Specifies the URL to load a private key from. The <key> element cannot be specified if the <key-store> or <key-loader> elements are specified.

Table A-23 (Cont.) identity-manager Subelements

Element	Required / Optional	Description
<key-loader>	Optional	Configures a custom implementation of com.tangosol.net.ssl.PrivateKeyLoader that provides a PrivateKey. A <class-name> subelement is used to provide the name of a class that implements the com.tangosol.net.ssl.PrivateKeyLoader interface. As an alternative, use a <class-factory-name> subelement to specify a factory class for creating the PrivateKeyLoader 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. The <key-loader> element cannot be specified if the <key> element is specified.
<cert>	Optional	Specifies the URL to load a certificate from. The <cert> element cannot be specified if the <key-store> element is specified.
<cert-loader>	Optional	Configures a custom implementation of com.tangosol.net.ssl.CertificateLoader that provides a Certificate. A <class-name> subelement is used to provide the name of a class that implements the com.tangosol.net.ssl.CertificateLoader interface. As an alternative, use a <class-factory-name> subelement to specify a factory class for creating the CertificateLoader 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. The <cert-loader> element cannot be specified if the <key-store> element is specified.
<password>	Optional	Specifies the private key password. This element cannot be used with the <password-provider> or <password-url> elements.
<password-url>	Optional	Specifies the file or simple URL to read the private key password from. This element cannot be used with the <password> or <password-provider> elements.
<password-provider>	Optional	Specifies a password provider implementation for retrieving the private key password. This element cannot be used with the <password> or <password-url> elements.

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-24 describes the subelements of the `<identity-transformer>` element.

Table A-24 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 packets into logical messages and dispatches them to the appropriate Coherence service for processing.

Elements

Table A-25 describes the subelements of the `incoming-message-handler` element.

Table A-25 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.
<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. 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 where 10 is the highest priority.

init-param

Used in: init-params.

Description

Defines an initialization parameter. Any number of init-param elements may be specified.

Initialization parameters can be specified by type or name. When using the <param-type> element, an object of the specified type is instantiated and initialized with the value specified in <param-value>. A constructor for <param-type> with the value of <param-value> is called to instantiate the object. When using the <param-name> element, a constructor for <param-name> with the value of <param-value> is called to instantiate the object.

Elements

Table A-26 describes the subelements of the init-param element.

Table A-26 init-param Subelement

Element	Required/ Optional	Description

<param-name>	Optional	Specifies the name of the initialization parameter. For example:
		`<init-param>`
		`  <param-name>sTableName</param-name>`
		`  <param-value>EmployeeTable</param-value>`
		`</init-param>`
		The <param-name> element cannot be specified if the <param-type> element is specified. See Initialization Parameter Settings.

Table A-26 (Cont.) init-param Subelement

Element	Required/ Optional	Description
<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-param> <param-type>java.lang.String</param-type> <param-value>EmployeeTable</param-value> </init-param> 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-27 describes the subelements of the init-params element.

Table A-27 init-params Subelement

Element	Required/ Optional	Description
<init-param>	Optional	Defines an individual initialization parameter.

instance

Used in: service-failure-policy, scope-resolver, partition-assignment-strategy, custom-archiver, and persistence-environment.

Description

The `<instance>` element contains the configuration of an implementation class or class factory that is used to plug in custom functionality. You can initialize parameters by writing XML which nests `<instance>` and `<class-scheme>` (or any other custom namespace) inside of `<param-value>` elements.

For example, given the following Java code:

```
public class MyClass
  {
  public MyClass(String s, OtherClass o, int i) { ... }
  }

public class OtherClass
  {
  public OtherClass(String s) { ... }
  }
```

You can initialize the `MyClass` and `OtherClass` classes by writing the following XML. In the XML, the `MyClass` class is initialized with the string `Hello World` and the integer `42`. The instance of the `OtherClass` class which appears in the `MyClass` class, is initialized with the string `Goodbye World`.

```
<instance>
  <class-name>MyClass</class-name>
    <init-params>
      <init-param>
        <param-value>Hello World</param-value>
      </init-param>
      <init-param>
        <param-value>
          <instance>
            <class-name>OtherClass</class-name>
              <init-params>
                <init-param>
                  <param-value>Goodbye World</param-value>
                </init-param>
              </init-params>
          </instance>
        </param-value>
      </init-param>
      <init-param>
        <param-value>42</param-value>
      </init-param>
    </init-params>
  </instance>
```

Elements

Table A-28 describes the subelements of the `instance` element.

Table A-28 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	Specifies initialization parameters which are accessible by implementations that include a public constructor with a matching signature. Initialization parameters can be specified when using both the <class-name> element and the <class-factory-name> element.

interceptor

Used in: interceptors

Description

The `interceptor` element defines the configuration associated with an event interceptor that is responsible for processing federation change records. Interceptors implements the `com.tangosol.net.events.EventInterceptor` interface for `FederatedChangeEvent` types. See Using Live Events.

> **Note:**
>
> Interceptors that are defined within a participant configuration are applied to all federated cache services that use the participant. To restrict the interceptor to a specific federated cache service, define the interceptor within a federated-scheme in the cache configuration file.

Elements

Table A-29 interceptor Subelements

Element	Required/ Optional	Description
<name>	Optional	Specifies a unique identifier for the interceptor.
<order>	Optional	Specifies whether the interceptor is the first interceptor in a chain of interceptors. The legal values are `LOW` and `HIGH`. A value of `HIGH` indicates that the interceptor is first in the chain of interceptors. A value of `LOW` indicates no order preference. The default value is `LOW`.

Table A-29 (Cont.) interceptor Subelements

Element	Required/ Optional	Description
<instance>	Required	Specifies the interceptor class to instantiate. The interceptor class must implement the `EventInterceptor` interface.

interceptors

Used in: participant

Description

The `interceptors` element contains any number of event interceptor definitions.

Elements

Table A-30 describes the subelements of the `interceptors` element.

Table A-30 interceptors Subelements

Element	Required/ Optional	Description
<interceptor>	Optional	Specifies an event interceptor implementation.

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-28 describes the subelements of the `journaling-config` element.

Table A-31 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 and 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-32 describes the subelements of the `key-store` element.

Table A-32 key-store Subelements

Element	Required/ Optional	Description
`<url>`	Required if the `<key-store-loader>` element is not specified	Specifies the Uniform Resource Locator (URL) to a key store. The `<url>` element cannot be specified if the `<key-store-loader>` element is specified.
`<key-store-loader>`	Required if the `<url>` element is not specified	Configures a custom implementation of `com.tangosol.net.ssl.KeyStoreLoader` that provides a `KeyStore`. A `<class-name>` subelement is used to provide the name of a class that implements the `com.tangosol.net.ssl.KeyStoreLoader` interface. As an alternative, use a `<class-factory-name>` subelement to specify a factory class for creating the `KeyStoreLoader` 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. The `<key-store-loader>` element cannot be specified if the `<url>` element is specified.
`<password>`	Optional	Specifies the password for the key store. This element cannot be used with the `<password-provider>` or `<password-url>` elements.
`<password-url>`	Optional	Specifies the file or a simple URL to read the key store password. This element cannot be used with the `<password>` or `<password-provider>` elements.
<password-provider>	Optional	Specifies a password provider implementation for retrieving the key store password. This element cannot be used with the `<password>` or `<password-url>` elements.
`<type>`	Optional	Specifies the type of a `java.security.KeyStore` instance. The default value is `PKCS12`. **Note**: The default key store type changed as of JDK 9. If you want to continue using `JKS` for backward compatibility, specify `JKS` in the `<type>` element.

license-config

Used in: coherence.

Description

The `license-config` element contains the details of the license that this member uses.

Elements

Table A-33 describes the subelements of the `license-config` element.

Table A-33 license-config Subelements

Element	Required/ Optional	Description
`<edition-name>`	Optional	Specifies the product edition that the member uses. Valid values are: `GE` (Grid Edition), `EE` (Enterprise Edition), `SE` (Standard Edition), `RTC` (Real-Time Client), `DC` (Data Client). The default value is `GE`. **Note:** The edition switches no longer enforce license restrictions. Do not change the default setting (`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 `coherence.mode` on the Java command line. The default value is `dev`. You cannot change this element during a rolling restart.

logging-config

Used in: coherence.

Elements

The following table describes the subelements of the `logging-config` element.

Table A-34 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` • `log4j2` • `slf4j` • *`file name`* If `log4j2` is specified, the Log4j2 API and Core 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/Log4j2` logging libraries. The preconfigured system property override is `coherence.log`.

Table A-34 (Cont.) logging-config Subelements

Element	Required/ Optional	Description
<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 Log4j2 logging systems. The default value is Coherence. The preconfigured system property override is 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 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}, {text}, and {ecid}. 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 coherence.log.limit.

management-config

Used in: coherence.

Elements

Table A-35 describes the subelements of the management-config element.

Table A-35 management-config Subelements

Element	Optional/ Required	Description
<managed-nodes>	Optional	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 – 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. • dynamic – (default) Allow this node to automatically host an MBean server and manage all MBeans. The preconfigured system property override is coherence.management.
<http-managed-nodes>	Optional	Specifies whether a cluster node's JVM has an [in-process] HTTP management server. The HTTP management server uses the Management service to fulfill its requests and is therefore affected by the other management settings such as <managed-nodes>. For example, if <managed-nodes> is set to none, then the HTTP management server does not start. If <managed-nodes> is set to local-only, then the HTTP management server exposes attributes and operations that are limited to local MBeans. Legal values are: • all – Instantiate an HTTP management server on this node. • inherit – Instantiate an HTTP management server on this cluster node if an MBean server will be instantiated on this node. See <managed-nodes> subelement. • none – (default) No HTTP management server is instantiated on this cluster node. The preconfigured system property override is coherence.management.http.
<allow-remote-management>	Optional	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 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 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 coherence.management.refresh.expiry

Table A-35 (Cont.) management-config Subelements

Element	Optional/ Required	Description
<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 coherence.management.refresh.timeout
<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 coherence.management.readonly.
<default-domain-name>	Optional	Specifies the domain name of an existing MBean server that is used to register MBeans exposed by the Coherence management framework. This element is used only if a cluster member has management enabled and the MBean server is located in the same process as the cluster member. If a value is not specified, the first existing MBean server is used. The element should only be used to identify an existing MBean server. This element is also used when implementing the MBeanServerFinder interface. See the <server-factory> element below.
<domain-name-suffix>	Optional	Enables configuring a meaningful application domain name suffix to append to a domain name. When this element's value is a non-empty string, the domain name is "Coherence@" appended with the value. This value is used only by the cluster nodes that have in-process MBeanServer and allow management of local or other node's managed objects. Defaults to an empty string.
<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 an 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. If a domain name is provided in the <default-domain-name> element, then it is used when instantiating the class. 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.
<extended-mbean-name>	Optional	Specifies whether global MBean names that are identified with a nodeId attribute are extended to identify the corresponding member name (if specified). Legal values are: true or false. The default value is false and indicates that the member name is not included in the Global MBean name. The preconfigured system property override is coherence.management.extendedmbeanname.

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-36 describes the subelements of the `mbean` element.

Table A-36 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.
`<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`).

Table A-36 (Cont.) Subelements of mbean

Element	Required/ Optional	Description
<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-37 describes the subelements of the mbeans element.

Table A-37 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="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-60 describes the subelements of the `mbean-filter` element.

Table A-38 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-39 describes the subelements of the `member-identity` element.

Table A-39 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 to prevent accidental cluster discovery among applications. The maximum allowed length for this value is 66 characters. An error is displayed if the value exceeds this limit. The preconfigured system property override is coherence.cluster. You cannot change this element during a rolling restart.
<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. The site name 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 site name also helps determine where to back up data when using distributed caching and the default partition assignment strategy. Lastly, the name is useful for displaying management information (for example, JMX) and interpreting log entries. The maximum allowed length for this value is 66 characters. An error is displayed if the value exceeds this limit. The preconfigured system property override is 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 and is often a cage, rack, or bladeframe identifier. The rack name 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 rack name also helps determine where to back up data when using distributed caching and the default partition assignment strategy. Lastly, the name is useful for displaying management information (for example, JMX) and interpreting log entries. The maximum allowed length for this value is 66 characters. An error is displayed if the value exceeds this limit. The preconfigured system property override is 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 maximum allowed length for this value is 66 characters. An error is displayed if the value exceeds this limit. The preconfigured system property override is coherence.machine.

Table A-39 (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 maximum allowed length for this value is 66 characters. An error is displayed if the value exceeds this limit. The preconfigured system property override is 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 maximum allowed length for this value is 66 characters. An error is displayed if the value exceeds this limit. The preconfigured system property override is 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 maximum allowed length for this value is 66 characters. An error is displayed if the value exceeds this limit. The preconfigured system property override is 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 where 10 is the highest priority. The preconfigured system property override is coherence.priority.

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. If you are having difficulties establishing a cluster when using multicast, see Performing a Multicast Connectivity Test in *Administering Oracle Coherence*.

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, then the Well Known Addresses feature can be used to eliminate the need for multicast traffic. See the well-known-addresses element.

Elements

Table A-40 describes the subelements of the `multicast-listener` element.

Table A-40 multicast-listener Subelements

Element	Required /Optional	Description
`<interface>`	Optional	Specifies the IP or sub-net of the local network interface (NIC) used for multicast traffic. The NIC is automatically selected based on whether multicast is used solely for member discovery or if mutlicast is used for member discovery and data transmission.
`<address>`	Required	Specifies the multicast IP address on which a multicast socket listens or publishes. 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 `224.{major version}.{minor version}.{service}`. The preconfigured system property override is `coherence.clusteraddress`. You cannot change discovery type from multicast to WKA during a rolling restart and visa-versa.
`<port>`	Required	Specifies the port on which the multicast socket listens or publishes. The port value is also used for WKA configurations. Legal values are from 1 to 65535. Select a cluster port that is outside of an operating system's ephemeral port range to avoid other software being randomly assigned the same port. A value between 1024 and 8999 is recommended. The default value is 7574. The preconfigured system property override is `coherence.clusterport`. You cannot change this element during a rolling restart.
`<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 `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.
`<priority>`	Required	Specifies a priority of the multicast listener execution thread. Legal values are from 1 to 10 where 10 is the highest priority.
`<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`.

Table A-40 (Cont.) multicast-listener Subelements

Element	Required /Optional	Description
<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.

name-service-addresses

Used in: participant

Description

The use of name-service-addresses within participant is deprecated. Use remote-addresses instead.

The name-service-addresses element contains the address (IP, or DNS name, and port) of one or more name service TCP/IP acceptors. A federation service initiator uses this information to establish a connection with a remote cluster. The TCP/IP initiator attempts to connect to the addresses in a random order until either the list is exhausted or a connection is established.

For details on using the name-service-address element as a subelement of tcp-initiator, see name-service-addresses in the cache configuration reference.

Elements

Table A-41 describes the subelements of the name-service-addresses element.

Table A-41 name-service-addresses Subelements

Element	Required/ Optional	Description
<socket-address>	Optional	Specifies the address (IP, or DNS name, and port) on which a name service TCP/IP acceptor is listening. Multiple <socket-address> elements can be defined. The <socket-address> element cannot be used together with an <address-provider> element or an <address> element.

Table A-41 (Cont.) name-service-addresses Subelements

Element	Required/ Optional	Description
<address-provider>	Optional	Specifies the address (IP, or DNS name, and port) on which a name service TCP/IP acceptor is listening or the configuration for a `com.tangosol.net.AddressProvider` implementation that supplies the address. The `address-provider` element also supports socket address references. The `<address-provider>` element cannot be used together with a `<socket-address>` element or an `<address>` element.
`<address>`	Optional	Specifies an IP address or DNS name. The bind address can also be an external NAT address that routes to a local address; however, both addresses must use the same port. Multiple `<address>` elements can be defined. The `<address>` element cannot be used together with an `<address-provider>` element or a `<socket-address>` element.

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-42 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-43 describes the subelements of the `outgoing-message-handler` element.

Table A-43 outgoing-message-handler Subelement

Element	Required/ Optional	Description
`<use-filters>`	Optional	Specifies a list of `<filter-name>` elements to be used by this handler.

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-44 describes the subelements of the `outstanding-packets` element.

Table A-44 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, and packet-publisher.

Description

Specifies the size (in packets or bytes) of the operating system buffer for datagram sockets.

Elements

Table A-45 describes the subelements of the `packet-buffer` element.

Table A-45 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 in *Java Platform, Standard Edition API Specification*. 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-46 describes the subelements of the packet-bundling element.

Table A-46 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-47 describes the subelements of the `packet-delivery` element.

Table A-47 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-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-48 describes the subelements of the `packet-publisher` element.

Table A-48 packet-publisher Subelements

Element	Required/ Optional	Description
<packet-size>	Optional	Specifies the packet sizes to use.
<packet-delivery>	Required	Specifies timing parameters related to reliable packet delivery.
<notification-queueing>	Required	Contains the notification queue related configuration info.
<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 where 10 is the highest priority.
`<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. Valid values are `true` and `false`. The default value is `true`. The preconfigured system property override is `coherence.tcmp.enabled`.

packet-size

Used in: packet-publisher.

Description

The packet-size element specifies the maximum and preferred packet sizes. All cluster nodes must use identical maximum packet sizes.

Elements

Table A-49 describes the subelements of the `packet-size` element.

Table A-49 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 than 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. The preferred length should be at least 512. The default value is based on the local nodes MTU.

packet-speaker

Used in: cluster-config.

Description

Specifies configuration information for the packet speaker which is used for network data transmission when packet publisher loads are high.

> **Note:**
>
> The packet speaker is not used for TCMP/TMB, which is the default protocol for data communication.

Elements

Table A-50 describes the subelements of the `packet-speaker` element.

Table A-50 packet-speaker Subelements

Element	Required/ Optional	Description
`<enabled>`	Optional	Specifies whether or not the packet-speaker thread is enabled. Valid values are `true` and `false`. The default value is `false`. The preconfigured system property override is `coherence.speaker.enabled`.
<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 where 10 is the highest priority.

participant

Used in: participants.

Description

The `participant` element specifies the remote coherence cluster that is participating in the federation. Any number of participants can be defined.

Elements

Table A-51 describes the subelements of the `participant` element.

Table A-51 participant Subelements

Element	Required/ Optional	Description
`<name>`	Required	Specifies the cluster name of the participant.

Table A-51 (Cont.) participant Subelements

Element	Required/ Optional	Description
<initial-action>	Optional	Specifies the startup action for the remote participant. The action is taken by the local participant when it starts up. Valid values are: • start – (default) specifies that storage nodes start replicating as soon as there are entries available for replication. • pause – Specifies that replication is paused. All entries that are marked for replication are stored in the federation service internal journal cache until a start operation is performed using the FederationManagerMbean MBean. • stop – Specifies that entries are not marked for replication nor are they stored in the federation service internal journal cache.
<connect-timeout>	Optional	Specifies the timeout of a connection to a destination participant. 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 1m.
<connect-retry-timeout>	Optional	Specifies the total amount of time trying to connect to a destination participant before giving up. 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 0.
<send-timeout>	Optional	Specifies the acknowledgement timeout for a federated replication message. 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 5m.

Table A-51 (Cont.) participant Subelements

Element	Required/ Optional	Description
<max-bandwidth>	Optional	The maximum bandwidth, per member, for sending federated data to a destination participant. This value is loaded from the source member's configuration of the destination participant. The default value is unlimited. Legal values are decimal values with optional factor and unit descriptors. Legal factor descriptor includes: • 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. Legal unit descriptor includes: • bps (bits per second) • Bps (bytes per second) If no unit is specified, a unit of bps is assumed.
<send-old-value>	Optional	Specifies whether the federation service should include the old values when replicating updated cache entries to remote participants. Legal values are true or false. The default value is true.
<batch-size>	Optional	Specifies the maximum number of journal entries that can be selected for replication. The batch size keeps the heap memory usage within limits when replicating. Legal values are integers representing the number of entries. The Default value is 50.
<geo-ip>	Optional	Specifies geographic metadata for the participant. The value is user-defined with no implicit structure and is used to define the participant (for example, latitude, longitude, country-code, and so on).
<participant-destinations>	Optional	Contains the definition of one or more participant-destination elements.
<participant-type>	Optional	Specifies the type of the participant. Valid values are: • cluster – a remote Coherence cluster. • interceptor – a participant that can intercept change records. An interceptor is a receiver participant only. The default value if no value is specified is cluster.
<interceptors>	Optional	Specifies any number of event interceptors for federation change records.
<remote-addresses>	Optional	Specifies socket addresses for a cluster that is participating as part of a federation.
<name-service-addresses>	Optional	The use of name-service-addresses in participant is deprecated. Use remote-addresses instead. Contains addresses of one or more name service TCP/IP acceptors for this cluster that is participating as part of a federation.
<error-on-elastic-data-full>	Optional	Specifies whether federation will put the participant into the ERROR state when Elastic Data or the device used by Elastic Data is full. Legal values are true or false. The default value is true. Available since release 14.1.2.0.3.

participants

Used in: federation-config.

Description

The `participants` element contains the declarative data for each federation participant.

Elements

Table A-51 describes the subelements of the `participants` element.

Table A-52 participants Subelements

Element	Required/ Optional	Description
<participant>	Optional	Specifies the remote coherence clusters that are participating in the federation.

participant-destination

Used in participant-destinations element.

Description

The `participant-destination` element specifies the destination-specific information (for example: for a specific federation service) of a given participant.

Elements

Table A-53 describes the subelements of the `participant-destination` element.

Table A-53 participant-destination Subelements

Element	Required/Optional	Description
<name>	Required	Specifies the name of the participant.
<remote-addresses>	Optional	Specifies the socket addresses of the participant to federate data to.

participant-destinations

Used in participant.

Description

The `participant-destinations` element contains the definition of one or more `participant-destination` elements.

Elements

Table A-54 describes the subelements of the `participant-destinations` element.

Table A-54 participant-destinations Subelements

Element	Required/Optional	Description
<participant-destination>	Optional	Contains the destination-specific information (for example: for a specific federation service) of a given participant.

password-provider

Used in: password-providers, identity-manager, key-store, access-controller.

Description

The `password-provider` element contains the declarative data for a password provider, which must be an instance of the `com.tangosol.net.PasswordProvider` interface. Password providers allow you to plug in a mechanism for determining an appropriate password instead of entering a password in an SSL socket provider configuration as clear text.

The `password-provider` element can be used in-line and also supports the `id` attribute which allows multiple password providers to be defined within the `<password-providers>` element and referenced. For example:

```
<key-store>
   <url>file:trust.jks</url>
   <password-provider>
      <name>trustMgrPass</name>
   </password-provider>
   <type>JKS</type>
</key-store>

<password-providers>
   <password-provider id="trustMgrPass">
      <class-name>pacakge.MyPassswordProvider</class-name>
      <init-params>
         <init-param>
            <param-name>param_1</param-name>
            <param-value>private</param-value>
         </init-param>
      </init-params>
   </password-provider>
   <password-provider id="identityMgrPass">
      <class-name>pacakage.MyPasswordProvider</class-name>
      <init-params>
         <init-param>
            <param-name>param_1</param-name>
            <param-value>private</param-value>
         </init-param>
      </init-params>
   </password-provider>
</password-providers>
```

Elements

Table A-55 describes the subelements of the `password-provider` element.

Table A-55 password-provider Subelements

Element	Required/ Optional	Description
`<class-name>`	Optional	Specifies the fully qualified name of a class that implements the `com.tangosol.net.PasswordProvider` 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 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	Specifies initialization parameters which are accessible by implementations that include a public constructor with a matching signature. Initialization parameters can be specified when using both the `<class-name>` element and the `<class-factory-name>` element.

password-providers

Used in: cluster-config.

Description

The `password-providers` element contains any number of password provider definitions.

Elements

Table A-56 describes the subelements of the `password-providers` element.

Table A-56 password-providers Subelements

Element	Required/ Optional	Description
<password-provider>	Optional	Specifies a password provider definition. The password provider must be a class that implements the `PasswordProvider` interface.

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-57 describes the subelements of the `pause-detection` element.

Table A-57 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`.

persistence-environment

Used in: persistence-environments.

Description

The `persistent-environment` element contains configuration information for a persistence environment implementation. The element supports the `id` attribute which allows multiple environment definitions to be created and referenced by a cache. For example:

```
<persistence-environments>
   <persistence-environment id="environment1">
      <persistence-mode>active</persistence-mode>
      <active-directory>/environment1/active</active-directory>
      <snapshot-directory>/environment1/snapshot</snapshot-directory>
      <trash-directory>/environment1/trash</trash-directory>
   </persistence-environment>
   <persistence-environment id="environment2">
      <persistence-mode>active</persistence-mode>
      <active-directory>/environment2/active</active-directory>
      <snapshot-directory>/environment2/snapshot</snapshot-directory>
      <trash-directory>/environment2/trash</trash-directory>
   </persistence-environment>
</persistence-environments>
```

Directory locations can be on a local disk or a shared disk. Local disk storage requires a list of storage-enabled hosts to be configured. See The address-provider Element for Persistence.

Elements

Table A-58 describes the subelements of the `persistence-environment` element.

Table A-58 persistence-environment Subelements

Element	Required/ Optional	Description
<instance>	Optional	Specifies a custom persistence environment implementation to be used instead of the default persistence environment.

Table A-58 (Cont.) persistence-environment Subelements

Element	Required/ Optional	Description
<persistence-mode>	Optional	Specifies whether the persistence environment is used: • To continually persist cached data (active) • Only when requested (on-demand) • In a custom override (active-async) • Asynchronously persist backups (active-backup) or . Legal values are: active, on-demand, active-async, and active-backup. The default value is on-demand. You cannot change the persistence-mode from on-demand to any of the active-* modes or visa-versa during a rolling restart. You can however change between any of the active-* modes, such as active -> active-async. While changing to active-async you should provide additional time between rolling restarts to ensure that members have sent all backups.
<active-directory>	Optional	Specifies the path to a directory under which cached data is actively persisted by a persistent environment. The default value is *USER_HOME*/coherence/active.
<snapshot-directory>	Optional	Specifies the path to a directory under which copies of cached data are persisted by a persistent environment. The default value is *USER_HOME*/coherence/snapshots.
<trash-directory>	Optional	Specifies the path to a directory under which potentially corrupted persisted data is stored by a persistent environment. The default value is *USER_HOME*/coherence/trash.

persistence-environments

Used in: cluster-config.

Description

The persistence-environments element contains the declarative data for each persistence environment.

Elements

Table A-59 describes the subelements of the persistence-environments element.

Table A-59 persistence-environments Subelements

Element	Required/ Optional	Description
<persistence-environment>	Optional	Contains configuration information for each persistence environment implementation.

provider

Used in: ssl, identity-manager, and trust-manager.

Description

The <provider> element contains the configuration information for a security provider that extends the java.security.Provider class.

Elements

Table A-60 describes the subelements of the provider element.

Table A-60 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 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.

Elements

Table A-61 describes the subelements of the ramjournal-manager element.

Table A-61 ramjournal-manager Subelements

Element	Required/ Optional	Description
<minimum-load-factor>	Optional	Specifies the factor of live data below which a journal file is eligible for compaction (garbage collection). Higher values will result in more aggressive compaction at the cost of increased processing time spent in compaction. Valid values are decimal numbers between 0.01 and 0.99. The default is 0.25. **Note**: Do not set this element except under advanced use cases.
<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. Binary values that exceed the maximum value size are automatically delegated to a flash journal.
<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 2MB and 2GB. The default value is 2MB. The maximum file size should not be changed. **Note**: • In most cases, setting <maximum-size> instead of <maximum-file-size> is recommended • <maximum-file-size> and <maximum-size> are mutually exclusive. – When <maximum-file-size> is specified, <maximum-size> = <maximum-file-size> * 512. – When <maximum-size> is specified, <maximum-file-size> = <maximum-size> / 512. – If <maximum-file-size> is smaller than 1M, it is set to 1M. • When <maximum-file-size> / 2 < 64M, <maximum-value-size> is limited by <maximum-file-size> / 2.
<collector-timeout>	Optional	Specifies the amount of time that the journal collector can remain unresponsive prior to considering it timed out. The minimum timeout is 30s. Legal values are strings representing time intervals. The default value is 10m. **Note**: Do not set this element except under advanced use cases.
<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 for a specific amount of memory is 1GB. The default value for percentage of heap is 25%. That is, the RAM journal resource manager uses a maximum of 25% of the available JVM heap.A RAM journal is, by default, backed by a flash journal and all data in excess of the maximum RAM size is automatically delegated to the flash journal. See the maximum-size element for details on disabling flash journal overflow. The preconfigured system property override is coherence.ramjournal.size.
<off-heap>	Optional	Specifies whether to use in the virtual machine's byte buffer or of-heap NIO buffers.

remote-addresses

Used in: participant, participant-destination.

Description

The `remote-addresses` element contains the addresses of clusters that are participating as part of a federation. The element can be used to specify both `NameService` service addresses by specifying the cluster port of the remote participant and direct connect addresses for environments which cannot use the `NameService` service for address lookup. Specifying the cluster port of the remote participant is recommended.

Elements

Table A-62 describes the subelements of the `remote-addresses` element.

Table A-62 remote-addresses Subelements

Element	Required/ Optional	Description
<socket-address>	Optional	Specifies the address (IP, or DNS name, and port) to which a socket is bound.
<address-provider>	Optional	specifies either socket address information (IP, or DNS name, and port) or an implementation of the `com.tangosol.net.AddressProvider` interface.

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-63 describes the subelements of the `reporter` element.

Table A-63 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-64 describes the subelements of the `security-config` element.

Table A-64 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 `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). See Steps to Implement a LoginModule in *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`.
`<authorizer>`	Optional	Contains the configuration information for a class that implements the `com.tangosol.net.security.Authorizer` interface which represents an environment-specific facility for authorizing callers to perform actions described by the corresponding permission objects. Use the <instance> element to enter the class.

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) and can be referenced by the default serializer for services that do not explicitly define a serializer (see defaults).

Additional serializers can be defined in an operational override file as required.

Elements

Table A-65 describes the subelements of the `serializer` element.

Table A-65 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-66 describes the subelements of the `serializers` element.

Table A-66 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:

- `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
- `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.
- `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.
- `RemoteCache`—A service that routes cache operations from Coherence*Extend clients to a cache on the cluster.
- `RemoteInvocation`— A service that routes cache invocation tasks from Coherence*Extend clients to a cache on the cluster.
- `NameService`—A service that is a specialized TCP acceptor that allows Coherence*Extend clients to connect to a proxy by specifying a proxy service name instead of a proxy service address.
- `RemoteNameService`— A `NameService` implementation that allows a JVM to use a remote `NameService` without having to join the Cluster.
- `PartitionedService.PartitionedCache.FederatedCache` – A specialized partitioned cache service that replicates cached data across clusters.

Elements

Table A-67 describes the subelements of the `services` element.

Table A-67 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 .
`<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: • `PartitionedService.PartitionedCache` (DistributedCache) • `ReplicatedCache` • `ReplicatedCache.Optimistic` • `SimpleCache` • `LocalCache` • `InvocationService` • `ProxyService` • `RemoteCache` • `RemoteInvocation` • `NameService`

`<use-filters>`	Optional	Contains the list of `filter` names to be used by this service. For example, specify `use-filter` as follows:
		`<use-filters>`
		`  <filter-name>gzip</filter-name>`
		`</use-filters>`
		The example activates `gzip` compression for the network messages used by this service, which can help substantially with WAN and low-bandwidth networks.

<init-params>	Optional	Specifies the initialization parameters that are specific to each service. Each parameter is described below.

- Initialization Parameter Settings

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 tables in this 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 **Parameter Value Description** column refers to the possible values for the corresponding `<param-value>` element.

The following sections describe the parameters that can be configured for each of the services:

- DistributedCache Service Parameters
- ReplicatedCache Service Parameters

- OptimisticCache Service Parameters
- Invocation Service Parameters
- LocalCache Service Parameters
- Proxy Service Parameters
- RemoteCache Service Parameters
- RemoteInvocation Service Parameters
- NameService Parameters
- RemoteNameService Parameters
- FederatedCache Service Parameters

DistributedCache Service Parameters

A DistributedCache service supports the parameters listed in Table A-68. 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-68 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 distributed cache partitions. Each storage-enabled cluster member that is running the distributed cache service manages a balanced number of partitions. Valid values are positive integers between 1 and 32767 and should be a prime number. The default value is 257 partitions. See Define the Partition Count. You cannot change this element during a rolling restart.This may appear to work but you will receive a message like the following indicating the change has been ignored: `<Warning> (thread=DistributedCache:PartitionedCache, member=8): This node is configured with a 'partition-count' value of 801, but the service senior is using a value of 257; overriding the local configuration.`
`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 `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 `coherence.distributed.localstorage`.

Table A-68 (Cont.) DistributedCache Service Parameters

Parameter Name	Parameter Value Description
`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 `coherence.distributed.transfer`.
`backup-count`	Specifies the number of members of the partitioned cache service that hold the backup data for each unit of storage in the cache. The default value is 1. The preconfigured system property override is `coherence.distributed.backupcount`. You cannot change this element during a rolling restart. This may appear to work but you will receive a message like the following indicating the change has been ignored: `<Warning> (thread=DistributedCache:PartitionedCache, member=8): This node is configured with a 'backup-count' value of 2, but the service senior is using a value of 1; overriding the local configuration.`
`thread-count`	**Note:** the `thread-count` element is deprecated and is replaced by setting the `thread-count-min` and `thread-count-max` elements to the same value. Specifies the number of daemon threads used by the partitioned cache service. Legal values are positive integers, `0`, or `-1`. The value `0` indicates that all relevant tasks are performed on the service thread. The value `-1` indicates that tasks are performed on the caller's thread where possible. 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 computer. 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 `coherence.distributed.threads`.
`key-associator`	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`	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`	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 preconfigured system property override is `coherence.distributed.task.hung`.

Table A-68 (Cont.) DistributedCache Service Parameters

Parameter Name	Parameter Value Description
`task-timeout`	Specifies the timeout value in milliseconds for requests executing on the service worker threads. This attribute applies only if the thread pool is used and only applies to entry processor implementations that implement the `PriorityTask` interface. 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. Legal values are positive integers or zero. If zero is specified, the default service-guardian `<timeout-milliseconds>` value is used. The preconfigured system property override is `coherence.distributed.task.timeout`.
`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 `0`. The preconfigured system property override is `coherence.distributed.request.timeout`.
`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-68 (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 parameter 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. The preconfigured system property override is `coherence.distributed.guard.timeout`.
`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`	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. You cannot change this element during a rolling restart. This also includes changing of any of the elements within `partitioned-quorum-policy-scheme`.

Table A-68 (Cont.) DistributedCache Service Parameters

Parameter Name	Parameter Value Description
`partition-assignment-strategy`	Specifies the strategy that is used by a partitioned service to manage partition distribution. • `simple` – (default) The simple assignment strategy attempts to balance partition distribution while ensuring machine-safety. • `mirror:<service-name>` – The mirror assignment strategy attempts to co-locate the service's partitions with the partitions of the specified service. This strategy is used to increase the likelihood that key-associated, cross-service cache access remains local to a member. • custom – a class that implements the `com.tangosol.net.partition.PartitionAssignmentStrategy` interface. The preconfigured system property override is `coherence.distributed.assignmentstrategy`. You cannot change this element during a rolling restart.
`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 `coherence.distributed.compressor`. You cannot change this element during a rolling restart.
`service-priority`	Specifies the priority for the service thread. Legal values are from 1 to 10 where 10 is the highest priority. The default value is `10`.
`event-dispatcher-priority`	Specifies the priority for the event dispatcher thread for each service. Legal values are from 1 to 10 where 10 is the highest priority. The default value is `10`.
`worker-priority`	Specifies the priority for the worker threads. Legal values are from 1 to 10 where 10 is the highest priority. The default value is `5`.

Table A-68 (Cont.) DistributedCache Service Parameters

Parameter Name	Parameter Value Description
`reliable-transport`	Specifies the transport protocol used by this service for reliable point-to-point communication. Specifying a value results in the use of a service-specific transport instance rather then the shared transport instance that is defined by the `<unicast-listener>` element. A service-specific transport instance can result in higher performance but at the cost of increased resource consumption and should be used sparingly for select, high priority services. In general, a shared transport instance uses less resource consumption than service-specific transport instances. Valid values are: • `datagram` – UDP protocol. • `tmb` (default) – TCP/IP message bus protocol. • `tmbs` – TCP/IP message bus protocol with SSL support. TMBS requires the use of an SSL socket provider. See socket-provider. • `sdmb` – Socket Direct Protocol (SDP) message bus. • `sdmbs` – SDP message bus with SSL support. SDMBS requires the use of an SSL socket provider. See socket-provider. The default value is the shared transport instance that is configured in the `<reliable-transport>` subelement of the <unicast-listener> element. The preconfigured system property override is `coherence.distributed.transport.reliable`. You cannot change this element from TLS to non-TLS during a rolling restart. You are however able to change between `tmb` or `datagram` during a rolling restart.
`async-backup`	Specifies whether the partitioned cache service should backup changes asynchronously while concurrently responding to the client. Legal values are `true` or `false`. The default value is `false`. The preconfigured system property override is `coherence.distributed.asyncbackup`.
`persistence`	Specifies the persistence-related configuration for a distributed cache service.
`thread-count-max`	Specifies the maximum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers greater or equal to the value of the `thread-count-min` element. The default value is `Integer.MAX_VALUE`. The preconfigured system property override is `coherence.distributed.threads.max`.
`thread-count-min`	Specifies the minimum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers less than or equal to the value of the `thread-count-max` element. The default value is `1`. The preconfigured system property override is `coherence.distributed.threads.min`.

ReplicatedCache Service Parameters

A ReplicatedCache service supports the parameters listed in Table A-69. These settings may also be specified for each service instance as part of the replicated-scheme element in the `coherence-cache-config.xml` descriptor.

Table A-69 ReplicatedCache Service Parameters

Parameter Name	Parameter Value Description
`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`.
`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.
`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`.
`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 `0`. The preconfigured system property override is `coherence.replicated.request.timeout`.
`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.

Table A-69 (Cont.) ReplicatedCache 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. The preconfigured system property override is `coherence.replicated.guard.timeout`.
`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`	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.
`service-priority`	Specifies the priority for the service thread. Legal values are from 1 to 10 where 10 is the highest priority. The default value is `10`.
`event-dispatcher-priority`	Specifies the priority for the event dispatcher thread for each service. Legal values are from 1 to 10 where 10 is the highest priority. The default value is `10`.

Table A-69 (Cont.) ReplicatedCache Service Parameters

Parameter Name	Parameter Value Description
`reliable-transport`	Specifies the transport protocol used by this service for reliable point-to-point communication. Specifying a value results in the use of a service-specific transport instance rather then the shared transport instance that is defined by the `<unicast-listener>` element. A service-specific transport instance can result in higher performance but at the cost of increased resource consumption and should be used sparingly for select, high priority services. In general, a shared transport instance uses less resource consumption than service-specific transport instances. Valid values are: • `datagram` – UDP protocol • `tmb` (default) – TCP/IP message bus protocol • `tmbs` – TCP/IP message bus protocol with SSL support. TMBS requires the use of an SSL socket provider. See socket-provider. • `sdmb` – Socket Direct Protocol (SDP) message bus. • `sdmbs` – SDP message bus with SSL support. SDMBS requires the use of an SSL socket provider. See socket-provider. The default value is the shared transport instance that is configured in the `<reliable-transport>` subelement of the <unicast-listener> element. The preconfigured system property override is `coherence.replicated.transport.reliable`. You cannot change this element from TLS to non-TLS during a rolling restart. You are however able to change between `tmb` or `datagram` during a rolling restart.

OptimisticCache Service Parameters

An OptimisiticCache service supports the parameters listed in Table A-70. These settings may also be specified for each service instance as part of the optimistic-scheme element in the `coherence-cache-config.xml` descriptor.

Table A-70 OptimisiticCache 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 `0`. The preconfigured system property override is `coherence.optimistic.request.timeout`.
`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.
`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. The preconfigured system property override is `coherence.optimistic.guard.timeout`.

Table A-70 (Cont.) OptimisiticCache Service Parameters

Parameter Name	Parameter Value Description
`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`	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.
`service-priority`	Specifies the priority for the service thread. Legal values are from 1 to 10 where 10 is the highest priority. The default value is `10`.
`event-dispatcher-priority`	Specifies the priority for the event dispatcher thread for each service. Legal values are from 1 to 10 where 10 is the highest priority. The default value is `10`.
`reliable-transport`	Specifies the transport protocol used by this service for reliable point-to-point communication. Specifying a value results in the use of a service-specific transport instance rather then the shared transport instance that is defined by the `<unicast-listener>` element. A service-specific transport instance can result in higher performance but at the cost of increased resource consumption and should be used sparingly for select, high priority services. In general, a shared transport instance uses less resource consumption than service-specific transport instances. Valid values are: • `datagram` – UDP protocol • `tmb` (default) – TCP/IP message bus protocol • `tmbs` – TCP/IP message bus protocol with SSL support. TMBS requires the use of an SSL socket provider. See socket-provider. • `sdmb` – Socket Direct Protocol (SDP) message bus. • `sdmbs` – SDP message bus with SSL support. SDMBS requires the use of an SSL socket provider. See socket-provider. The default value is the shared transport instance that is configured in the `<reliable-transport>` subelement of the <unicast-listener> element. The preconfigured system property override is `coherence.optimistic.transport.reliable`. You cannot change this element from TLS to non-TLS during a rolling restart. You are however able to change between `tmb` or `datagram` during a rolling restart.

Invocation Service Parameters

An Invocation service supports the parameters listed in Table A-71. These settings may also be specified for each service instance as part of the invocation-scheme element in the `coherence-cache-config.xml` descriptor.

Table A-71 Invocation Service Parameters

Parameter Name	Parameter Value Description
`thread-count`	**Note:** the `thread-count` element is deprecated and is replaced by setting the `thread-count-min` and `thread-count-max` elements to the same value. Specifies the number of daemon threads used by the service. Legal values are positive integers, `0`, or `-1`. The value `0` indicates that all relevant tasks are performed on the service thread. The value `-1` indicates that tasks are performed on the caller's thread where possible. 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 `coherence.invocation.threads`.
`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 preconfigured system property override is `coherence.invocation.task.hung`.
`task-timeout`	Specifies the default task execution timeout value for requests that can time out (that is, polls and `PriorityTask` implementations), but do not explicitly specify an 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. Legal values are positive integers or zero (indicating no default timeout). The preconfigured system property override is `coherence.invocation.task.timeout`.

Table A-71 (Cont.) Invocation 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 `0`. The preconfigured system property override is `coherence.invocation.request.timeout`.
`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.
`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. The preconfigured system property override is `coherence.invocation.guard.timeout`.

Table A-71 (Cont.) Invocation Service Parameters

Parameter Name	Parameter Value Description
`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`	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.
`service-priority`	Specifies the priority for the service thread. Legal values are from 1 to 10 where 10 is the highest priority. The default value is `10`.
`event-dispatcher-priority`	Specifies the priority for the event dispatcher thread for each service. Legal values are from 1 to 10 where 10 is the highest priority. The default value is `10`.
`worker-priority`	Specifies the priority for the worker threads. Legal values are from 1 to 10 where 10 is the highest priority. The default value is `5`.
`reliable-transport`	Specifies the transport protocol used by this service for reliable point-to-point communication. Specifying a value results in the use of a service-specific transport instance rather then the shared transport instance that is defined by the `<unicast-listener>` element. A service-specific transport instance can result in higher performance but at the cost of increased resource consumption and should be used sparingly for select, high priority services. In general, a shared transport instance uses less resource consumption than service-specific transport instances. Valid values are: • `datagram` – UDP protocol • `tmb` (default) – TCP/IP message bus protocol • `tmbs` – TCP/IP message bus protocol with SSL support. TMBS requires the use of an SSL socket provider. See socket-provider. • `sdmb` – Socket Direct Protocol (SDP) message bus. • `sdmbs` – SDP message bus with SSL support. SDMBS requires the use of an SSL socket provider. See socket-provider. The default value is the shared transport instance that is configured in the `<reliable-transport>` subelement of the `<unicast-listener>` element. The preconfigured system property override is `coherence.invocation.transport.reliable`. You cannot change this element from TLS to non-TLS during a rolling restart. You are however able to change between `tmb` or `datagram` during a rolling restart.

Table A-71 (Cont.) Invocation Service Parameters

Parameter Name	Parameter Value Description
`thread-count-max`	Specifies the maximum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers greater or equal to the value of the `thread-count-min` element. The default value is `Integer.MAX_VALUE`. The preconfigured system property override is `coherence.invocation.threads.max`.
`thread-count-min`	Specifies the minimum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers less than or equal to the value of the `thread-count-max` element. The default value is `1`. The preconfigured system property override is `coherence.invocation.threads.min`.

LocalCache Service Parameters

A LocalCache service supports the parameters listed in Table A-72.

Table A-72 LocalCache Service Parameters

Parameter Name	Parameter Value Description
`lock-enforce`	Specifies whether locking is enforced for put, remove and clear operations. If the value is false, a client is responsible for calling lock and unlock explicitly. The default value is `false`.
`lock-wait`	Specifies the number of milliseconds to continue trying to obtain a lock. This parameters is only used if locking enforcement is enabled. A value of `-1` blocks the calling thread until the lock can be obtained. The default value is `0`.

Proxy Service Parameters

A Proxy service supports the parameters listed in Table A-73. These settings may also be specified for each service instance as part of the proxy-scheme element in the `coherence-cache-config.xml` descriptor.

Table A-73 Proxy Service Parameters

Parameter Name	Parameter Value Description
`acceptor-config`	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`	Contains the configuration of the clustered service proxies managed by this service.

Table A-73 (Cont.) Proxy Service Parameters

Parameter Name	Parameter Value Description
`thread-count`	**Note:** the `thread-count` element is deprecated and is replaced by setting the `thread-count-min` and `thread-count-max` elements to the same value. Specifies the number of daemon threads for use by the proxy service. Legal values are positive integers or `0`. The default value is `0`, which indicates that dynamic thread pooling is enabled and that the number of threads automatically adjusts based on proxy service load. Specifying a positive value explicitly sets the number of threads in the pool. **Note:** Proxy service threads perform operations on behalf of the calling application. Therefore, when explicitly setting the number of threads, set the value to as many threads as there are concurrent operations. To disable the thread pool, set the `thread-count-max` and `thread-count-min` parameters to `0`. All relevant tasks are performed on the proxy service thread when the thread pool is disabled. The preconfigured system property override is `coherence.proxy.threads`.
`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 preconfigured system property override is `coherence.proxy.task.hung`.
`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. Legal values are positive integers or zero (indicating no default timeout). The preconfigured system property override is `coherence.proxy.task.timeout`.
`request-timeout`	Specifies the maximum amount of time a proxy waits for requests that are sent to other proxies of the same name. This parameter should not be used because requests are never sent between proxies.
`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.

Table A-73 (Cont.) Proxy 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 parameter 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. The preconfigured system property override is coherence.proxy.guard.timeout.
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	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.
proxy-quorum-policy-scheme	Specifies quorum policy settings for the Proxy service. You cannot change this element during a rolling restart. This also includes changing of any of the elements within proxy-quorum-policy-scheme.
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.

Table A-73 (Cont.) Proxy Service Parameters

Parameter Name	Parameter Value Description
service-priority	Specifies the priority for the service thread. Legal values are from 1 to 10 where 10 is the highest priority. The default value is 10.
event-dispatcher-priority	Specifies the priority for the event dispatcher thread for each service. Legal values are from 1 to 10 where 10 is the highest priority. The default value is 10.
worker-priority	Specifies the priority for the worker threads. Legal values are from 1 to 10 where 10 is the highest priority. The default value is 5.
reliable-transport	Specifies the transport protocol used by this service for reliable point-to-point communication. Specifying a value results in the use of a service-specific transport instance rather then the shared transport instance that is defined by the <unicast-listener> element. A service-specific transport instance can result in higher performance but at the cost of increased resource consumption and should be used sparingly for select, high priority services. In general, a shared transport instance uses less resource consumption than service-specific transport instances. Valid values are: • datagram – UDP protocol • tmb (default) – TCP/IP message bus protocol • tmbs – TCP/IP message bus protocol with SSL support. TMBS requires the use of an SSL socket provider. See socket-provider. • sdmb – Socket Direct Protocol (SDP) message bus. • sdmbs – SDP message bus with SSL support. SDMBS requires the use of an SSL socket provider. See socket-provider. The default value is the shared transport instance that is configured in the <reliable-transport> subelement of the <unicast-listener> element. The preconfigured system property override is coherence.proxy.transport.reliable. You cannot change this element from TLS to non-TLS during a rolling restart. You are however able to change between tmb or datagram during a rolling restart.
thread-count-max	Specifies the maximum number of daemon threads that are allowed in the dynamic thread pool. This parameter is only valid if a thread-count value is set to 0. Legal values are positive integers or 0. Setting a value for both this parameter and the thread-count-min parameter indicates that no daemon threads are created and that all client requests are handled by the proxy service thread. The default value is Integer.MAX_VALUE. The preconfigured system property override is coherence.proxy.threads.max.
thread-count-min	Specifies the minimum number of daemon threads that are allowed (and initially created) in the dynamic thread pool. This parameter is only valid if a thread-count value is set to 0. Legal values are positive integers or 0. Setting a value for both this parameter and the thread-count-max parameter indicates that no daemon threads are created and that all client requests are handled by the proxy service thread. The default value is 1. The preconfigured system property override is coherence.proxy.threads.min.

RemoteCache Service Parameters

A RemoteCache service supports the parameters listed in Table A-74. These settings may also be specified for each service instance as part of the remote-cache-scheme element in the coherence-cache-config.xml descriptor.

Table A-74 RemoteCache Service Parameters

Parameter Name	Parameter Value Description
`initiator-config`	Contains the configuration of the connection initiator used by the service to establish a connection with the cluster.
`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.
`defer-key-association-check`	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.

RemoteInvocation Service Parameters

A RemoteInvocation service supports the parameters listed in Table A-75. These settings may also be specified for each service instance as part of the remote-invocation-scheme element in the `coherence-cache-config.xml` descriptor.

Table A-75 RemoteInvocation Service Parameters

Parameter Name	Parameter Value Description
`initiator-config`	Contains the configuration of the connection initiator used by the service to establish a connection with the cluster.
`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.

NameService Parameters

A NameService service supports the parameters listed in Table A-76.

Table A-76 NameService Parameters

Parameter Name	Parameter Value Description
`acceptor-config`	Contains the configuration of a connection acceptor that is used discover proxy services that are available in the cluster for use by Coherence*Extend clients.
`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.

RemoteNameService Parameters

A RemoteNameService service supports the parameters listed in Table A-77.

Table A-77 RemoteNameService Parameters

Parameter Name	Parameter Value Description
`initiator-config/tcp-initiator`	Contains the configuration of the TCP connection initiator used by the service to establish a connection with the cluster.
`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.

FederatedCache Service Parameters

A FederatedCache service supports the parameters listed in Table A-78. These settings may also be specified for each service instance as part of the <federated-scheme> element in the `coherence-cache-config.xml` descriptor.

Table A-78 FederatedCache 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 distributed cache partitions. Each storage-enabled cluster member that is running the distributed cache service manages a balanced number of partitions. Valid values are positive integers between 1 and 32767 and should be a prime number. The default value is 257 partitions. See Define the Partition Count. You cannot change this element during a rolling restart.This may appear to work but you will receive a message like the following indicating the change has been ignored: `<Warning> (thread=DistributedCache:PartitionedCache, member=8): This node is configured with a 'partition-count' value of 801, but the service senior is using a value of 257; overriding the local configuration.`
`local-storage`	Specifies whether this member of the `FederatedCache` 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 `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 `coherence.distributed.localstorage`.

Table A-78 (Cont.) FederatedCache Service Parameters

Parameter Name	Parameter Value Description
`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 `coherence.distributed.transfer`.
`backup-count`	Specifies the number of members of the federated cache service that hold the backup data for each unit of storage in the cache. The default value is 1. The preconfigured system property override is `coherence.distributed.backupcount`. You cannot change this element during a rolling restart. This may appear to work but you will receive a message like the following indicating the change has been ignored: `<Warning> (thread=DistributedCache:PartitionedCache, member=8): This node is configured with a 'backup-count' value of 2, but the service senior is using a value of 1; overriding the local configuration.`
`thread-count`	**Note:** the `thread-count` element is deprecated and is replaced by setting the `thread-count-min` and `thread-count-max` elements to the same value. Specifies the number of daemon threads used by the federated cache service. Legal values are positive integers, `0`, or `-1`. The value `0` indicates that all relevant tasks are performed on the service thread. The value `-1` indicates that tasks are performed on the caller's thread where possible. 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 computer. 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 `coherence.distributed.threads`.
`key-associator`	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`	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`	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 preconfigured system property override is `coherence.distributed.task.hung`.

Table A-78 (Cont.) FederatedCache Service Parameters

Parameter Name	Parameter Value Description
`task-timeout`	Specifies the timeout value in milliseconds for requests executing on the service worker threads. This attribute applies only if the thread pool is used and only applies to entry processor implementations that implement the `PriorityTask` interface. 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. Legal values are positive integers or zero. If zero is specified, the default service-guardian `<timeout-milliseconds>` value is used. The preconfigured system property override is `coherence.distributed.task.timeout`.
`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 `0`. The preconfigured system property override is `coherence.distributed.request.timeout`.
`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 federated 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-78 (Cont.) FederatedCache 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 parameter 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. The preconfigured system property override is coherence.distributed.guard.timeout.
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	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 federated cache service. See partitioned-quorum-policy-scheme. You cannot change this element during a rolling restart. This also includes changing of any of the elements within partitioned-quorum-policy-scheme.

Table A-78 (Cont.) FederatedCache Service Parameters

Parameter Name	Parameter Value Description
`partition-assignment-strategy`	Specifies the strategy that is used by a federated service to manage partition distribution. • `simple` – (default) The simple assignment strategy attempts to balance partition distribution while ensuring machine-safety. • `mirror:<service-name>` – The mirror assignment strategy attempts to co-locate the service's partitions with the partitions of the specified service. This strategy is used to increase the likelihood that key-associated, cross-service cache access remains local to a member. • custom – a class that implements the `com.tangosol.net.partition.PartitionAssignmentStrategy` interface. The preconfigured system property override is `coherence.distributed.assignmentstrategy`. You cannot change this element during a rolling restart.
`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 `coherence.distributed.compressor`. You cannot change this element during a rolling restart.
`service-priority`	Specifies the priority for the service thread. Legal values are from 1 to 10 where 10 is the highest priority. The default value is `10`.
`event-dispatcher-priority`	Specifies the priority for the event dispatcher thread for each service. Legal values are from 1 to 10 where 10 is the highest priority. The default value is `10`.
`worker-priority`	Specifies the priority for the worker threads. Legal values are from 1 to 10 where 10 is the highest priority. The default value is `5`.

Table A-78 (Cont.) FederatedCache Service Parameters

Parameter Name	Parameter Value Description
`reliable-transport`	Specifies the transport protocol used by this service for reliable point-to-point communication. Specifying a value results in the use of a service-specific transport instance rather then the shared transport instance that is defined by the `<unicast-listener>` element. A service-specific transport instance can result in higher performance but at the cost of increased resource consumption and should be used sparingly for select, high priority services. In general, a shared transport instance uses less resource consumption than service-specific transport instances. Valid values are: • `datagram` – UDP protocol • `tmb` (default) – TCP/IP message bus protocol • `tmbs` – TCP/IP message bus protocol with SSL support. TMBS requires the use of an SSL socket provider. See socket-provider. • `sdmb` – Socket Direct Protocol (SDP) message bus. • `sdmbs` – SDP message bus with SSL support. SDMBS requires the use of an SSL socket provider. See socket-provider. The default value is the shared transport instance that is configured in the `<reliable-transport>` subelement of the <unicast-listener> element. The preconfigured system property override is `coherence.distributed.transport.reliable`. You cannot change this element from TLS to non-TLS during a rolling restart. You are however able to change between `tmb` or `datagram` during a rolling restart.
`async-backup`	Specifies whether the federated cache service should backup changes asynchronously while concurrently responding to the client. Legal values are `true` or `false`. The default value is `false`. The preconfigured system property override is `coherence.distributed.asyncbackup`.
`persistence`	Specifies the persistence-related configuration for a federated cache service.
`thread-count-max`	Specifies the maximum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers greater or equal to the value of the `thread-count-min` element. The default value is `Integer.MAX_VALUE`. The preconfigured system property override is `coherence.distributed.threads.max`.
`thread-count-min`	Specifies the minimum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers less than or equal to the value of the `thread-count-max` element. The default value is `1`. The preconfigured system property override is `coherence.distributed.threads.min`.

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-79 describes the subelements of the `service-guardian` element.

Table A-79 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 causes the local node to exit gracefully, halting the process only when the duration of the `coherence.shutdown.timeout` system property exceeds. • `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. The default value is `305000`. The preconfigured system property override is `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 Operational Configuration Attribute Reference.

services

Used in: cluster-config.

Description

The services element contains the declarative data for each service.

Elements

Table A-67 describes the subelements of the `services` element.

Table A-80 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-81 describes the subelements of the `shutdown-listener` element.

Table A-81 shutdown-listener Subelements

Element	Required/ Optional	Description
`<enabled>`	Required	Specifies the type of action to take upon an external JVM shutdown. Legal values: • `none` – Perform no explicit shutdown actions. • `force` – Perform a "hard-stop" on the node by calling `Cluster.stop()`. • `graceful` – (Default) Perform a "normal" shutdown by calling `Cluster.shutdown()`. • `true` – Same as force. • `false` – Same as none. The `coherence.shutdown.timeout` system property configures the duration to wait for shutdown to complete before timing out. The value is a time duration string such as "3s" for 3 seconds, "5m" for 5 minutes, or "1h" for 1 hour. The default value is 2 minutes. When the time duration to complete a graceful shutdown exceeds the `coherence.shutdown.timeout` time, the JVM process is considered hung, and the JVM process terminates abruptly using halt. The preconfigured system property override is `coherence.shutdownhook`.

snapshot-archivers

Used in: cluster-config.

Description

The `snapshot-archivers` element contains the declarative data for each persistence snapshot archiver. Snapshot archivers are used to save persistence snapshots.

Elements

Table A-82 describes the subelements of the `shutdown-listener` element.

Table A-82 snapshot-archivers Subelements

Element	Required/ Optional	Description

`<directory-archiver>`	Optional	Contains the configuration information for the default directory-based snapshot archiver that uses a shared directory to store archived snapshots. The archive directory location and name must be the same across all members and accessible to all members. The `directory-archiver` element supports the `id` attribute which allows multiple directory archiver definitions to be created and referenced by a cache. Specify a shared directory within the `<archive-directory>` element. For example:
		`<directory-archiver id="archiver1">`
		`<archive-directory>/temp/mydirectory</archive-directory>`
		`</directory-archiver>`

`<custom-archiver>`	Optional	Contains the configuration information for a class that implements the `SnapshotArchiver` interface. The `custom-archiver` element supports the `id` attribute which allows multiple custom archiver definitions to be created and referenced by a cache. See instance.

socket-address

Used in: well-known-addresses, address-provider, remote-addresses, and name-service-addresses.

Description

The `socket-address` element specifies the address (IP, or DNS name, and port) to which a socket is bound.

Elements

Table A-83 describes the subelements of the `socket-address` element.

Table A-83 socket-address Subelements

Element	Required/ Optional	Description
`<address>`	Required	Specifies the IP address that a Socket listens or publish on. Enter either an IP address or DNS name. The bind address can also be an external NAT address that routes to a local address; however, both addresses must use the same port.
`<port>`	Required	Specifies the port that the Socket listens or publishes on. Legal values are from 1 to 65535. Use the default cluster port, 7574, unless a different cluster port has been defined.

socket-provider

Used in: socket-providers, unicast-listener, and ssl.

Description

The `<socket-provider>` element contains the configuration information for a socket and channel factory. The following socket providers are supported and 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="coherence.security.keystore">
                  file:keystore.p12</url>
               <password system-property="coherence.security.password"/>
            </key-store>
            <password system-property="coherence.security.password"/>
         </identity-manager>
         <trust-manager>
            <algorithm>PeerX509</algorithm>
            <key-store>
               <url system-property="coherence.security.keystore">
                  file:keystore.p12</url>
               <password system-property="coherence.security.password"/>
            </key-store>
         </trust-manager>
         <socket-provider>tcp</socket-provider>
      </ssl>
   </socket-provider>

   <socket-provider id="sdp">
      <sdp/>
   </socket-provider>
</socket-providers>
```

Alternate SSL definitions can be created to support more elaborate SSL configurations.

Elements

Table A-84 describes the subelements of the `socket-provider` element.

Table A-84 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.

Table A-84 (Cont.) socket-provider Subelements

Element	Required/ Optional	Description
<sdp>	Optional	Specifies a socket provider that produce SDP-based sockets and channel implementations provided that the JVM and underlying network stack supports SDP.

socket-providers

Used in cluster-config.

Description

The socket-providers element contains the declarative data for each socket provider implementation. Coherence supports the following socket providers: system, tcp, ssl, and sdp.

Elements

Table A-85 describes the subelements of the socket-providers element.

Table A-85 socket-providers Subelements

Element	Required/ Optional	Description
<socket-provider>	Optional	Specifies the configuration information for a socket and channel factory.

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-86 describes the subelements of the ssl element.

Table A-86 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.

Table A-86 (Cont.) ssl Subelements

Element	Required/ Optional	Description
<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 the 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.

Table A-86 (Cont.) ssl Subelements

Element	Required/ Optional	Description
<hostname-verifier>	Optional	Coherence provides a default implementation of HostnameVerifier. Use the <action> subelement to specify the action that the default hostname verifier should take during the SSL handshake if there is a mismatch between the host name in the URL and the host name in the digital certificate that the server sends back as part of the SSL connection. <action> takes one of the following two possible values: • **allow**: Allow all connections. There is no hostname verification. For backwards compatibility, this is the default option. • **default**: Use the default hostname verifier to verify the host. The host name verification passes if the host name in the certificate matches the host name in the URL to which the client connects. It also allows wild-card Subject Alternate Names (SAN) and Common Names (CN). If you do not want a match against CN if SAN DNS names are present, set the system property coherence.security.ssl.verifyCNAfterSAN to false. If the URL specifies localhost, you can set the system property coherence.security.ssl.allowLocalhost to true to allow 127.0.0.1, or the default IP address of the local machine. You can also specify 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 the 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. **Note:** In the secured production mode, hostname verification is enabled by default. When secured production mode is not enabled, hostname verification is also not enabled by default. For more information about host name verification, see Using Host Name Verification in *Securing Oracle Coherence*.

Table A-86 (Cont.) ssl Subelements

Element	Required/ Optional	Description
cipher-suites	Optional	Specifies a list of ciphers. Use the name element within the cipher-suites element to enter a cipher. Multiple name elements can be specified. Use the usage attribute to specify whether the list of ciphers are allowed or disallowed. If the usage attribute value is black-list, then the specified ciphers are removed from the default enabled cipher list. If the usage attribute value is white-list, then the specified ciphers are the enabled ciphers. The default value if the usage attribute is not specified is white-list.
protocol-versions	Optional	Specifies a list of protocol versions. Use the name element within the protocol-versions element to enter a protocol version. Multiple name elements can be specified. Use the usage attribute to specify whether the list of protocol versions are allowed or disallowed. If the usage attribute value is black-list, then the specified protocol versions are removed from the default enabled protocol list. If the usage attribute value is white-list, then the specified protocol versions are the enabled protocols. The default value if the usage attribute is not specified is white-list.
<socket-provider>	Optional	Specifies the configuration information for a delegate provider for SSL. Valid values are tcp and sdp. The default value is tcp.
<refresh-period>	Optional	The refresh-period element specifies the period to use for an attempt to refresh keys and certs. This option is used in cases where keys or certs have a short lifetime and need to be refreshed at runtime. If this element is omitted or is set to a value less than or equal to zero, then keys and certs will not be refreshed. The value of this element must be in the following format: (\d)+((.)(\d)+)?(MS|ms|S|s|M|m|H|h|D|d)? Here, 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.
<refresh-policy>	Optional	The refresh-policy element contains the configuration information for a refresh policy that extends the com.tangosol.net.ssl.RefreshPolicy class. A <class-name> subelement is used to provide the name of a class that implements the RefreshPolicy interface. As an alternative, use a <class-factory-name> subelement to specify a factory class for creating the RefreshPolicy 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.

storage-authorizer

Used in: storage-authorizers.

Description

The `storage-authorizer` element contains the declarative data for a storage access authorizer, which must be an instance of the `com.tangosol.net.security.StorageAccessAuthorizer` interface. Storage access authorizers provides server-side access control authorization. The `storage-authorizer` element supports the `id` attribute which allows multiple storage access authorizers to be defined and referenced by a cache. For example:

```
<storage-authorizers>
   <storage-authorizer id="auditing">
      <class-name>package.MyStorageAccessAuthorizer</class-name>
   </storage-authorizer>
   <storage-authorizer id="restricted-access">
      <class-name>package.AnotherStorageAccessAuthorizer</class-name>
   </storage-authorizer>
</storage-authorizers>
```

Elements

Table A-87 describes the subelements of the `storage-authorizer` element.

Table A-87 storage-authorizer Subelements

Element	Required/ Optional	Description
`<class-name>`	Optional	Specifies the fully qualified name of a class that implements the `com.tangosol.net.security.StorageAccessAuthorizer` 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 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	Specifies initialization parameters which are accessible by implementations that include a public constructor with a matching signature. Initialization parameters can be specified when using both the `<class-name>` element and the `<class-factory-name>` element.

storage-authorizers

Used in: cluster-config.

Description

The `storage-authorizers` element contains the declarative data for any number of storage access authorizers.

Elements

Table A-88 describes the subelements of the `storage-authorizers` element.

Table A-88 storage-authorizers Subelements

Element	Required/ Optional	Description
<storage-authorizer>	Optional	Specifies a storage access authorizer. The storage access authorizer must be a class that implements the `StorageAccessAuthorizer` interface.

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 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-89 describes the subelements of the `tcp-ring-listener` element.

Table A-89 tcp-ring-listener Subelements

Element	Required/ Optional	Description
`<enabled>`	Optional	Specifies whether the tcp ring listener should be enabled to defect node failures faster. Valid 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 operating system dependent.

Table A-89 (Cont.) tcp-ring-listener Subelements

Element	Required/ Optional	Description
<priority>	Required	Specifies a priority of the tcp ring listener execution thread. Legal values are from 1 to 10 where 10 is the highest priority.

topology-definitions

Used in: federation-config.

Description

The topology-definitions element specifies the topology configuration information. The topology selected determines how data is synchronized between clusters.

Elements

Table A-90 describes the subelements of the topology-definitions element.

Table A-90 topology-definitions Subelements

Element	Required/ Optional	Description
<active-active>	Optional	Specifies the configuration information for an active-active topology.
<active-passive>	Optional	Specifies the configuration information for an active-passive topology.
<hub-spoke>	Optional	Specifies the configuration information for a hub-spoke topology.
<central-replication>	Optional	Specifies the configuration information for a central-replication topology.
<custom-topology>	Optional	Specifies the configuration for a custom topology.

tracing-config

Used in: coherence.

Description

The tracing-config element contains the configuration information for the tracing facility.

Elements

Table A-91 tracing-config Subelements

Element	Required/ Optional	Description
`<sampling-ratio>`	Required	Controls the likelihood of a tracing span being collected. For example: A value of `1.0` will result in all tracing spans being collected, while a value of `0.1` will result in roughly 1 out of every 10 tracing spans being collected. A value of `0` indicates that tracing spans should only be collected if they are already in the context of another tracing span. A value of `-1` disables tracing completely. This element defaults to `-1` if it is not overridden. The preconfigured system property override is `coherence.tracing.ratio`.

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-92 describes the subelements of the `traffic-jam` element.

Table A-92 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-93 describes the elements of the `trust-manager` element.

Table A-93 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. The `<key-store>` element cannot be specified if the `<cert>` or `<cert-loader>` elements are specified.
`<cert>`	Optional	Specifies the URL to load a certificate from. Multiple `<cert>` elements can be specified to load multiple certificates. The `<cert>` element cannot be specified if the `<key-store>` element is specified.
`<cert-loader>`	Optional	Configures a custom implementation of `com.tangosol.net.ssl.CertificateLoader` that provides a `Certificate`. Multiple `<cert-loader>` elements can be specified to load multiple certificates. A `<class-name>` subelement is used to provide the name of a class that implements the `com.tangosol.net.ssl.CertificateLoader` interface. As an alternative, use a `<class-factory-name>` subelement to specify a factory class for creating the `CertificateLoader` 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. The `<cert-loader>` element cannot be specified if the `<key-store>` element is specified.

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 to which a cluster node binds 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, then the Well Known Addresses feature can be used to eliminate the need for multicast traffic. See well-known-addresses.

Elements

Table A-94 describes the subelements of the `unicast-listener` element.

Table A-94 unicast-listener Subelements

Element	Required/ Optional	Description
<socket-provider>	Optional	Specifies either: the configuration for a socket provider, or it references a socket provider configuration that is defined within the <socket-providers> element. The following socket providers are available: `system` (default), `ssl`, `tcp`, and `sdp`. Refer to the socket provider configuration using their defined `id` attribute name. For example: `<socket-provider>ssl</socket-provider>` The preconfigured system property override is `coherence.socketprovider`.
`<reliable-transport>`	Optional	Specifies the transport protocol that is used for reliable point-to-point communication. By default, all services use the configured protocol and use a shared transport instance. A service can also explicitly specify the transport protocol using a `reliable-transport` service parameter, which results in a service-specific transport instance. A service-specific transport instance can result in higher performance but at the cost of increased resource consumption and should be used sparingly for select, high priority services. In general, a shared transport instance uses less resource consumption than service-specific transport instances. Valid values are: • `datagram` – UDP protocol • `tmb` (default) – TCP/IP message bus protocol • `tmbs` – TCP/IP message bus protocol with SSL support. TMBS requires the use of an SSL socket provider. See socket-provider. • `sdmb` – Socket Direct Protocol (SDP) message bus. • `sdmbs` – SDP message bus with SSL support. SDMBS requires the use of an SSL socket provider. See socket-provider. The preconfigured system property override is `coherence.transport.reliable`. You cannot change this element from TLS to non-TLS during a rolling restart. You are however able to change between `tmb` or `datagram` during a rolling restart.
<well-known-addresses>	Optional	Contains a list of well-known addresses that are used by the cluster discovery protocol instead of using multicast broadcast to discover cluster members.
`<machine-id>`	Optional	Specifies an identifier that should uniquely identify each server machine. The default value is generated from the `machine-name` element or, lacking that, from the address of the default network interface. Instead of setting `machine-id`, set `machine-name` (as well as `rack-name` and `site-name`).
`<discovery-address>`	Optional	Specifies an IP address or DNS name. The address may also be entered using CIDR notation as a subnet and mask (for example, `192.168.1.0/24`), which allows runtime resolution against the available local IP addresses. The default value is the wildcard address.

Table A-94 (Cont.) unicast-listener Subelements

Element	Required/ Optional	Description
<address>	Required	Specifies the IP address on which a unicast socket listens or publishes. The address may also be entered using CIDR notation as a subnet and mask (for example, 192.168.1.0/24), which allows runtime resolution against the available local IP addresses. The bind address can also be an external NAT address that routes to a local address; however, both addresses must use the same port. The default value is unspecified and indicates that an address is automatically selected. The preconfigured system property override is coherence.localhost.
<port>	Required	Specifies the port on which the unicast socket listens or publishes. A second port is automatically opened and defaults to the next port. Legal values are from 0 to 65535. The default value is 0 and indicates that the listener ports are automatically assigned from a computer's available ephemeral ports so as to avoid port conflicts with other applications. The preconfigured system property override is coherence.localport.
<port-auto-adjust>	Required	Specifies whether the port automatically increments if the specified port cannot be bound to because it is already in use. Alternatively, port conflicts can be avoided by setting the <port> element to 0. Valid values are true, false, or the upper limit on the port range. The lower limit is the value specified for the <port> element. The default value is true. The preconfigured system property override is 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 where 10 is the highest priority.

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-95 describes the subelements of the packet-speaker element.

Table A-95 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 Performing a Multicast Connectivity Test in *Administering Oracle Coherence*.

Description

By default, Coherence uses a multicast protocol to discover other members when forming a cluster. If multicast networking is undesirable, or unavailable in your environment, the Well Known Addresses (WKA) feature can be used to eliminate the need for multicast traffic. When in use, the cluster is configured with a relatively small list of members 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 machines to be simultaneously active at any point in time; however at least one WKA machine must be running for other members to join the cluster.

Example

The following example configures two well-known addresses from which joining members can discover or form the cluster.

```
<cluster-config>
   <unicast-listener>
      <well-known-addresses>
         <address id="1">192.168.0.100</address>
         <address id="2">192.168.0.101</address>
      </well-known-addresses>
   </unicast-listener>
</cluster-config>
```

Elements

Table A-96 describes the subelements of the `well-known-addresses` element.

Table A-96 well-known-addresses Subelements

Element	Required/ Optional	Description
<socket-address>	Optional	The use of the `<socket-address>` element within the `<well-known-addresses>` element is deprecated. Use the `<address>` element instead. Specifies a list of WKA that are used by the cluster discovery protocol instead of using multicast broadcast. Additionally, all cluster communication is performed using unicast. If empty or unspecified, multicast communications is used. The preconfigured system property overrides are `coherence.wka` and `coherence.wka.port`.
`<address>`	Optional	Specifies a list of WKA machine addresses (IP addresses or DNS names) that are used by the cluster discovery protocol instead of using multicast broadcast. The bind address can also be an external NAT address that routes to a local address; however, both addresses must use the same port. Multiple `<address>` elements must be differentiated by including an `id` attribute that is set to a unique value. If a list of WKA machine addresses is specified, then a member must be started on one of the WKA machines. In addition, all cluster communication is performed using unicast. If the element is empty or unspecified, multicast communications is used. The preconfigured system property override is `coherence.wka`.
<address-provider>	Optional	Contains the configuration for a `com.tangosol.net.AddressProvider` implementation that supplies the WKA machine list. The calling component attempts to obtain the full list upon member startup, the provider must return a terminating `null` address to indicate that all available addresses have been returned.

Operational Configuration Attribute Reference

The operational configuration attribute reference describes the attributes available in the operational deployment descriptor.

Table A-97 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: address-provider, 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, persistence-environment, and mbeans
`id`	Optional	The `id` attribute differentiates elements that can have multiple occurrences (for example, <service>). See Understanding the XML Override Feature.
`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 System Property Overrides.

B

Cache Configuration Elements

The cache configuration element reference provides a detailed description of the cache configuration deployment descriptor elements.

> **Note:**
>
> Coherence XML configuration files are validated at runtime using the corresponding XML schemas. It is important that any custom configuration files conform to the schema, because something as simple as having elements in the wrong order will cause validation to fail. As validation errors produced by XML schema validators can be difficult to understand, we recommend that custom configuration files are edited in a development environment capable of validating the XML against the schema as it is being written. This can save a lot of time compared to debugging validation errors at runtime.

This appendix includes the following sections:

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/1.3/coherence-cache-config.xsd

The `<cache-config>` element is the root element of the cache configuration descriptor and includes the XSD and namespace declarations. 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">
```

Note:

- 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.

Cache Configuration Element Reference

The cache configuration element reference includes all non-terminal cache configuration elements.Each section includes instructions on how to use the element and also includes descriptions for all valid subelements.

- http-acceptor
- identity-manager
- incoming-message-handler
- initiator-config
- init-param
- init-params
- instance
- interceptor
- interceptors
- invocation-scheme
- invocation-service-proxy
- key-associator
- key-partitioning
- key-store
- listener
- local-address
- local-scheme
- memcached-acceptor
- name-service-addresses
- near-scheme
- nio-file-manager
- operation-bundling
- optimistic-scheme
- outgoing-message-handler
- overflow-scheme
- page-size
- paged-topic-scheme
- paged-external-scheme
- partitioned-quorum-policy-scheme
- partition-listener
- persistence
- provider
- proxy-config
- proxy-quorum-policy-scheme
- proxy-scheme
- ramjournal-scheme
- read-write-backing-map-scheme
- remote-addresses

- remote-cache-scheme
- remote-grpc-cache-scheme
- remote-invocation-scheme
- replicated-scheme
- resource-config
- serializer
- socket-address
- socket-provider
- ssl
- tcp-acceptor
- tcp-initiator
- topologies
- topology
- transactional-scheme
- trust-manager
- view-scheme

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-1 describes the subelements of the `acceptor-config` element.

Table B-1 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>` or `<memcached-acceptor>` elements.
<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>` or `<memcached-acceptor>` elements.
<memcached-acceptor>	Optional	Specifies the configuration information for a connection acceptor that accepts connections from remote memcached clients over TCP/IP. This element cannot be used together with the `<tcp-acceptor>` or `<http-acceptor>` elements.

Table B-1 (Cont.) acceptor-config Subelements

Element	Required/ Optional	Description
<grpc-acceptor>	Optional	Specifies the configuration information for a connection acceptor that accepts connections from remote gRPC clients over TCP/IP. This element cannot be used together with the <tcp-acceptor>, <memcached-acceptor>, or <http-acceptor> elements.
<incoming-message-handler>	Optional	Specifies the configuration information that is used to regulate client-to-cluster connection resource usage.
<outgoing-message-handler>	Optional	Specifies the configuration information used by the connection acceptor to detect dropped client-to-cluster connections.
<use-filters>	Optional	Contains the list of filter names to be used by this connection acceptor. For example, specifying <use-filter> as follows activates gzip compression for all network messages, which can help substantially with WAN and low-bandwidth networks. <use-filters> <filter-name>gzip</filter-name> </use-filters>
<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> You cannot change this element during a rolling restart. For example, you cannot change from java to pof serialization or the serializer class.
<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: name-service-addresses, remote-addresses, tcp-acceptor, memcached-acceptor

Description

The address-provider element specifies either socket address information (IP, or DNS name, and port) or an implementation of the com.tangosol.net.AddressProvider interface. The interface offers a programmatic way to define socket addresses.

The preferred approach is to reference an address provider definition that is included in an operational override file. This approach decouples deployment configuration from application

configuration. However, socket addresses can also be configured in-line and is typical during development. See address-provider.

The following example references an address provider definition that contains the socket address to which a TCP/IP acceptor is bound.

```
<tcp-acceptor>
   <address-provider>ap2</address-provider>
</tcp-acceptor>
```

The following example references an address provider definition that contains the socket address of a TCP/IP acceptor on the cluster.

```
<tcp-initiator>
   <remote-addresses>
      <address-provider>ap3</address-provider>
   </remote-addresses>
</tcp-initiator>
```

The following example references an address provider definition that contains the socket address of a name service TCP/IP acceptor on the cluster.

```
<tcp-initiator>
   <name-service-addresses>
      <address-provider>ap4</address-provider>
   </name-service-addresses>
</tcp-initiator>
```

The following example references an address provider definition that contains the socket address to which a TCP/IP memcached acceptor is bound.

```
<memcached-acceptor>
   <address-provider>ap5</address-provider>
</memcached-acceptor>
```

Elements

Table B-2 describes the subelements of the `address-provider` element.

Table B-2 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 that 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.

Table B-2 (Cont.) address-provider Subelements

Element	Required/ Optional	Description
<socket-address>	Optional	Specifies the address (IP, or DNS name, and port) to which a socket is bound. This element cannot be used if an address provider implementation is defined using the `<class-name>` or `<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
- nio-file-manager—uses NIO to implement memory-mapped file based cache

Implementation

This store manager is implemented by the `com.tangosol.io.AsyncBinaryStoreManager` class.

Elements

Table B-3 describes the subelements of the `async-store-manager` element.

Table B-3 async-store-manager Subelements

Element	Required/ Optional	Description
`<class-name>`	Optional	Specifies a custom implementation of an asynchronous 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.
<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.
<nio-file-manager>	Optional	Configures the external cache to use a memory-mapped file for cache storage.

Table B-3 (Cont.) async-store-manager Subelements

Element	Required/ Optional	Description
`<async-limit>`	Optional	Specifies the maximum number of bytes that are queued to be written asynchronously. Setting the value to zero indicates that the implementation default for the maximum number of bytes is used. The value of this element must be in the following format: `(\d)+((.)(\d)+)?[K|k|M|m]?[B|b]?` where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied: • K (kilo, 2^{10}) • M (mega, 2^{20}) 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-4 describes the subelements of the `authorized-hosts` element.

Table B-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 are allowed to join the proxy as an extend client. The content override attributes ID can be optionally used to fully or partially override the contents of this element with an XML document that is external to the base document. As of release 14.1.2.0.3 and later, this can be a comma-separated list of IP addresses and/or host names. The default cache configuration file allows setting this value with the system property `coherence.extend.authorized.hosts`.
`<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.

Table B-4 (Cont.) authorized-hosts Subelements

Element	Required/ Optional	Description
<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. **Note:** If this value is set, both <host-address> and <host-range> are superceded by this setting. If a host-filter is provided, then setting system property coherence.extend.authorized.hosts has no effect.

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-5 describes the subelements of the back-scheme element.

Table B-5 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.
<federated-scheme>	Optional	Defines a distributed scheme that synchronizes cache entries across clusters that are participants of a federation.
<replicated-scheme>	Optional	Defines a cache scheme where each cache entry is stored on all cluster nodes.
<optimistic-scheme>	Optional	Defines a replicated cache scheme which uses optimistic rather then pessimistic locking.
<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.

Table B-5 (Cont.) back-scheme Subelements

Element	Required/ Optional	Description
<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 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.

> **Note:**
>
> The `partitioned` subelement is only used if the parent element is the `distributed-scheme` element.

Elements

Table B-6 describes the subelements of the `backing-map-scheme` element.

Table B-6 backing-map-scheme Subelements

Element	Required/ Optional	Description
`<federate-apply-synthetic>`	Optional	Specifies whether the changes received from remote federation participants should be applied locally as synthetic updates. Valid values are `true` or `false`. The default value is `false`. **Note:** As a federation origin (source cluster), synthetic updates are not considered for federation to destination participants. Enabling this property may disable/cause disruption for topologies other than active-active or active-standby; multihop federation topologies may not forward changes if this flag is set.

Table B-6 (Cont.) 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 the value is true, the scheme that is specified as the backing map is used to configure backing maps for each individual partition of the PartitionAwareBackingMap. If the value is false, the scheme 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. **Note:** Backing maps that use RAM and Flash journaling are always partitioned.
<transient>	Optional	Specifies whether or not the enclosed backing map should be persisted using a persistence environment. This element is respected only for the backing-map-scheme element that is a child of a distributed-scheme element. Valid values are true or false. If set to false, a persistence environment is used to persist the contents of the backing map. If set to true, the backing map is assumed to be transient and its contents will not be recoverable upon cluster restart. The default value is false.
<sliding-expiry>	Optional	Specifies whether or not the expiry delay of entries should be extended by read operations. By default the expiry delay is only extended upon updates. The default value is false. For details on setting the expiry delay, see the <expiry-delay> element.
<storage-authorizer>	Optional	Specifies a reference to a storage access authorizer that is defined in an operational configuration file. The storage access authorizer is used by a partitioned cache to authorize access to the underlying cache data. If configured, all read and write access to the data in the cache storage (backing map) will be validated and/or audited by the configured authorizer. The following example references a storage access authorizer definition with the id attribute authorizer1. <storage-authorizer>auditing</storage-authorizer> See storage-authorizer.
<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.
<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.

Table B-6 (Cont.) backing-map-scheme Subelements

Element	Required/ Optional	Description
<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-7 describes the subelements of the `backup-storage` element.

Table B-7 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`— (default) The corresponding implementations class is `java.util.HashMap`. • `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 preconfigured system property override is `coherence.distributed.backup`.
`<initial-size>`	Optional	Specifies the initial buffer size in bytes. The class name is only applicable if the `<type>` element is set to `off-heap` or `file-mapped`. Specifies the initial buffer size in bytes.The value of this element must be in the following format: `(\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 mega is assumed. Legal values are positive integers between 1 and `Integer.MAX_VALUE` (2147483647). The default value is `1MB`.

Table B-7 (Cont.) backup-storage Subelements

Element	Required/ Optional	Description
<maximum-size>	Optional	Specifies the initial buffer size in bytes. The class name is only applicable if the <type> element is set to off-heap or file-mapped. The value of this element must be in the following format: (\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 mega is assumed. Legal values are positive integers between 1 and Integer.MAX_VALUE (2147483647). The default value is 1024MB.
<directory>	Optional	Specifies the path name for the directory that the disk persistence manager (com.tangosol.util.nio.MappedBufferManager) uses as the root to store files. The directory is only applicable if the <type> element is set to file-mapped. If a value is not specified or a non-existent directory is specified, a temporary file in the default location is used. The default value is the default temporary directory designated by the Java run time.
<class-name>	Optional	Specifies a class name for the custom storage implementation. The class name is only applicable if the <type> element is set to custom.
<scheme-name>	Optional	Specifies a scheme name for the ConfigurableCacheFactory. The scheme name is only applicable if the <type> element is set to scheme.

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 Oracle Berkeley Database JE.

Description

The BDB store manager is used to define external caches which uses Berkeley Database JE on disk embedded databases for storage. See Persistent Cache on Disk and In-memory Cache with Disk Based Overflow.

Implementation

iThis store manager is implemented by the com.tangosol.io.bdb.BerkeleyDBBinaryStoreManager class, and produces BinaryStore objects implemented by the sscom.tangosol.io.bdb.BerkeleyDBBinaryStore class.

Elements

Table B-8 describes the subelements of the `bdb-store-manager` element.

Table B-8 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.
<init-params>	Optional	Specifies additional Berkeley DB configuration settings. See The JE Properties File in *Getting Started with Berkeley DB Java Edition*. Also used to specify initialization parameters, for use in custom implementations.
`<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.

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-9 describes the subelements of the `bundle-config` element.

Table B-9 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. See Cache Configuration Deployment Descriptor.

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-10 describes the subelements of the `cache-config` element.

Table B-10 cache-config Subelements

Element	Required/ Optional	Description
<defaults>	Optional	Defines factory wide default settings.
<interceptors>	Optional	Specifies any number of event interceptors that process events. Specifiying the `<interceptors>` element as a child of the `<cache-config>` element scopes the interceptors to the cache configuration and allows interceptors to receive events such as `ConfigurableCacheFactory` lifecycle events.
<caching-scheme-mapping>	Optional	Specifies the caching-scheme that is used for caches, based on the cache's name.
<topic-scheme-mapping	Optional	Specifies the `paged-topic-scheme` that is used for topics, based on topic'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 cache name pattern used by an application.

Elements

Table B-11 describes the subelements of the `cache-mapping` element.

Table B-11 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 slash (`/`) and colon (`:`) are reserved characters and cannot be used in cache names. The following cache name patterns are supported: • Exact match. For example, `MyCache`. • Prefix match using a wildcard (*prefix*`*`). For example, `My*` that matches to any cache name starting with `My`. • Any match using a wildcard (`*`). Matches to any cache name. If a cache name can be matched to multiple cache mappings, then exact matches are selected over wildcard matches. If no exact match is specified, then the last matching wildcard pattern (based on the order in which they are defined in the file) is selected.
`<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.
`<key-type>`	Optional	Specifies the fully-qualified name of the Java class for `NamedCache` entry keys.

Table B-11 (Cont.) cache-mapping Subelements

Element	Required/ Optional	Description
<value-type>	Optional	Specifies the fully-qualified name of the Java class for NamedCache entry values.
<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.
<interceptors>	Optional	Specifies any number of event interceptors that process events for a specific cache.
<federated>	Optional	Specifies whether a cache that is mapped to a federated-scheme should be federated. Valid values are true or false. Default value is true.

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-12 describes the subelements of the cache-service-proxy element.

Table B-12 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.

Elements

Table B-13 describes the subelements of the cachestore-scheme element.

Table B-13 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.
<class-scheme>	Optional	Specifies the implementation of the cache store. The specified class must implement either of the following interfaces. • com.tangosol.net.cache.CacheStore—for read-write support • com.tangosol.net.cache.CacheLoader—for read-only support • com.tangosol.net.cache.BinaryEntryStore— similar to CacheStore, but operates on BinaryEntry objects.

Table B-13 (Cont.) cachestore-scheme Subelements

Element	Required/ Optional	Description
<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.
<federated-loading>	Optional	Specifies whether the federation service federates entries loaded from the cache store to remote participants. Valid values are `true` or `false`. The default value is `false`.

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-14 describes the subelement you can define within the `caching-scheme-mapping` element.

Table B-14 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-15 describes the different types of schemes you can define within the `caching-schemes` element.

Table B-15 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.
<federated-scheme>	Optional	Defines a federated scheme to federate cache data between separate Coherence clusters.
<view-scheme>	Optional	Defines a local view of the entries of an existing cache.
<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.
<paged-topic-scheme>	Optional	Defines a paged-topic scheme where storage of values and metadata is partitioned across the cluster nodes.
<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.
<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.

Table B-15 (Cont.) caching-schemes Subelements

Element	Required/ Optional	Description
<replicated-scheme>	Optional	**Note:** Replicated cache is deprecated. A View, Near, or Distributed/Partitioned cache should be used in its place. Defines a cache scheme where each cache entry is stored on all cluster nodes.
<optimistic-scheme>	Optional	**Note:** Optimistic cache is deprecated. A View, Near or Distributed/Partitioned cache should be used in its place. Defines a replicated cache scheme which uses optimistic rather then pessimistic locking.

class-scheme

Used in: caching-schemes, local-scheme, distributed-scheme, replicated-scheme, optimistic-scheme, near-scheme, front-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.

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-16 describes the subelements of the `class-scheme` element.

Table B-16 class-scheme Subelements

Element	Required/ Optional	Description
<scheme-name>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.

Table B-16 (Cont.) class-scheme Subelements

Element	Required/ Optional	Description
<scheme-ref>	Optional	Specifies the name of another scheme to inherit from. See Using Scheme Inheritance.
<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 that 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-17 describes the subelements of the `custom-store-manager` element.

Table B-17 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.

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-18 describes the subelements of the `defaults` element.

Table B-18 defaults Subelements

Element	Required/ Optional	Description
`<scope-name>`	Optional	Specifies the scope name for this configuration. The scope name is added (as a prefix) to all services generated by a cache factory. The name is used to isolate services when using multiple cache factories; thus, avoiding unintended joining of services with similar names from different configurations.
<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>` You cannot change this element during a rolling restart. For example, you cannot change from java to pof serialization or the serializer class.
<socket-provider>	Optional	Specifies either: the configuration for a socket provider, or it references a socket provider configuration that is defined within the <socket-providers> element of the operational deployment descriptor. The following socket providers are available: `system` (default), `tcp`, `ssl`, and `sdp`. Refer to the socket providers 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 Understanding Distributed Caches and Partitioned Cache.

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-19 describes the subelements of the `distributed-scheme` element.

Table B-19 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.

Table B-19 (Cont.) distributed-scheme Subelements

Element	Required/ Optional	Description
<scope-name>	Optional	Specifies the scope name for this configuration. The scope name is added (as a prefix) to all services generated by a cache factory. The name is used to isolate services when using multiple cache factories; thus, avoiding unintended joining of services with similar names from different configurations.
<service-name>	Optional	Specifies a name for the distributed cache service instance that manages the cache that is created from this distributed scheme. The distributed cache service definition is defined within the <services> element in the tangosol-coherence.xml file. See DistributedCache Service Parameters. Different distributed schemes can use different partitioned cache service instances to maintain separate caches. The slash (/) and colon (:) are reserved characters and cannot be used in service names. The default name if no name is specified is DistributedCache.
<service-priority>	Optional	Specifies the priority for the service thread. Legal values are from 1 to 10 where 10 is the highest priority. The default value is the service-priority value specified in the tangosol-coherence.xml descriptor. See the service-priority parameter in DistributedCache Service Parameters.
<event-dispatcher-priority>	Optional	Specifies the priority for the event dispatcher thread for each service. Legal values are from 1 to 10 where 10 is the highest priority. The default value is the event-dispatcher-priority value specified in the tangosol-coherence.xml descriptor. See the event-dispatcher-priority parameter in DistributedCache Service Parameters.
<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. You cannot change this element during a rolling restart. For example, you cannot change from java to pof serialization or the serializer class.
<reliable-transport>	Optional	Specifies the transport protocol used by this service for reliable point-to-point communication. Specifying a value results in the use of a service-specific transport instance rather then the shared transport instance that is defined by the <unicast-listener> element. A service-specific transport instance can result in higher performance but at the cost of increased resource consumption and should be used sparingly for select, high priority services. In general, a shared transport instance uses less resource consumption than service-specific transport instances. Valid values are: • datagram – UDP protocol • tmb (default) – TCP/IP message bus protocol • tmbs – TCP/IP message bus protocol with SSL support. TMBS requires the use of an SSL socket provider. See socket-provider. • sdmb – Socket Direct Protocol (SDP) message bus. • sdmbs – SDP message bus with SSL support. SDMBS requires the use of an SSL socket provider. See socket-provider. The default value is the <reliable-transport> value specified in the tangosol-coherence.xml descriptor. See the reliable-transport parameter in DistributedCache Service Parameters.

Table B-19 (Cont.) distributed-scheme Subelements

Element	Required/ Optional	Description
<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. 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	**Note:** The thread-count element is deprecated and is replaced by setting the thread-count-min and thread-count-max elements to the same value. Specifies the number of daemon threads used by the partitioned cache service. Legal values are positive integers, 0, or -1. The value 0 indicates that all relevant tasks are performed on the service thread. The value -1 indicates that tasks are performed on the caller's thread where possible.
<thread-count-max>	Optional	Specifies the maximum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers greater or equal to the value of the thread-count-min element. The default value is the thread-count-max value specified in the tangosol-coherence.xml descriptor. See the thread-count-max parameter in DistributedCache Service Parameters.
<thread-count-min>	Optional	Specifies the minimum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers less than or equal to the value of the thread-count-max element. The default value is the thread-count-min value specified in the tangosol-coherence.xml descriptor. See the thread-count-max parameter in DistributedCache Service Parameters.
<worker-priority>	Optional	Specifies the priority for the worker threads. Legal values are from 1 to 10 where 10 is the highest priority. The default value is the worker-priority value specified in the tangosol-coherence.xml descriptor. See the worker-priority parameter in DistributedCache Service Parameters.
<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.

Table B-19 (Cont.) distributed-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. 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.
<partition-count>	Optional	Specifies the number of distributed cache partitions. Each storage-enabled cluster member that is running the distributed cache service manages a balanced number of partitions. Valid values are positive integers between 1 and 32767 and should be a prime number. A list of primes can be found at http://primes.utm.edu/lists/. The default value is the value specified in the tangosol-coherence.xml descriptor. See the partition-count parameter in DistributedCache Service Parameters.
<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.
<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. The default value is the backup-count value specified in the tangosol-coherence.xml descriptor. See DistributedCache Service Parameters.
<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-19 (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. The default value is the `partition-assignment-strategy` value specified in the `tangosol-coherence.xml` descriptor. See the `partition-assignment-strategy` parameter in DistributedCache Service Parameters. • `simple` – The simple assignment strategy attempts to balance partition distribution while ensuring machine-safety • `mirror:<service-name>` – The mirror assignment strategy attempts to co-locate the service's partitions with the partitions of the specified service. This strategy is used to increase the likelihood that key-associated, cross-service cache access remains local to a member. • custom – a class that implements the `com.tangosol.net.partition.PartitionAssignmentStrategy` interface. Enter a custom strategy using the <instance> element.
<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-min` 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.
`<task-timeout>`	Optional	Specifies the timeout value in milliseconds for requests executing on the service worker threads. This attribute applies only if the thread pool is used (the `thread-count-min` value is positive) and only applies to entry processor implementations that implement the `PriorityTask` interface. 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.
`<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.

Table B-19 (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 instance. 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-19 (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.
<persistence>	Optional	Specifies the persistence-related configuration for a partitioned cache service.
<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`.
<interceptors>	Optional	Specifies any number of event interceptors that process events for all caches of a specific distributed service.
<async-backup>	Optional	Specifies whether the partitioned cache service backs up data asynchronously while concurrently responding to the client. Asynchronous backup is often used to increase client performance. However, applications that require strict data integrity must be designed and tested to ensure that data is not at risk. 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) .

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 NIO to implement memory-mapped file based 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. The Berkly database JE implementation does however support persistence for non-clustered caches, see the `<store-name>` subelement of bdb-store-manager. Clustered persistence should be configured by using a read-write-backing-map-scheme on a distributed-scheme.

Elements

Table B-20 describes the subelements of the `external-scheme` element.

Table B-20 external-scheme Subelements

Element	Required/ Optional	Description
`<scheme-name>`	Optional	Specifies the scheme's name. The name must be unique within a configuration file.

Table B-20 (Cont.) external-scheme Subelements

Element	Required/ Optional	Description
<scheme-ref>	Optional	Specifies the name of another scheme to inherit from. See Using Scheme Inheritance.
<class-name>	Optional	Specifies a custom implementation of the external cache. Any custom implementation must extend either of the following classes and declare the exact same set of public constructors as the superclass.: • 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 • com.tangosol.net.cache.CompactSerializationCache—for compact on-heap footprint
<init-params>	Optional	Specifies initialization parameters, for use in custom external cache implementations.
<async-store-manager>	Optional	Configures the external cache to use an asynchronous storage manager wrapper for any other storage manager. See Pluggable Storage Manager.
<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.
<nio-file-manager>	Optional	Configures the external cache to use a memory-mapped file for cache storage.
<high-units>	Optional	Specifies the size limit of the cache. The value represents 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 the <unit-calculator> subelement). 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. Valid values are positive integers and 0. The value of this element must be in the following format: (\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: • B or b (byte, 1) • K or k (kilo, 2^{10}) • M or m (mega, 2^{20}) • G or g (giga, 2^{30}) • T or t (tera, 2^{40}) The default value is 0 and implies no limit. If the value exceeds Integer.MAX_VALUE, then a unit factor is automatically used to adjust the value accordingly. If the value does not contain a factor, a factor of one is assumed.

Table B-20 (Cont.) external-scheme Subelements

Element	Required/ Optional	Description
<unit-calculator>	Optional	Specifies the type of unit calculator to use. A unit calculator is used to determine the cost (in "units") of a given object. This element is used only if the <high-units> element is set to a positive number. 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. • <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.
<unit-factor>	Optional	Specifies the factor by which the <low-units> and <high-units> settings are adjusted. For example, if you use a BINARY unit calculator, a factor of 1048576 can be used to specify megabytes instead of bytes in the <low-units> and <high-units> settings. The unit factor also adjusts the Cache MBean Units attribute. If a <high-units> is specified that would exceed the Integer.MAX_VALUE units, then a unit factor is automatically calculated on start up and does not need to be explicitly set. 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 proactively evicted. The value of this element must be in the following format: (\d)+((.)(\d)+)?[MS|ms|S|s|M|m|H|h|D|d]? where the first non-digits (from left to right) indicate the unit of time duration: • MS or ms (milliseconds) • S or s (seconds) • M or m (minutes) • H or h (hours) • D or d (days) If the value does not contain a unit, a unit of seconds is assumed. A value of zero implies no expiry. 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.

federated-scheme

Used in: caching-schemes

Description

The `federated-scheme` element contains the federated caching scheme configuration information. A federated cache is a partitioned cache that can be synchronized across clusters that are participants of a federation.

Elements

Table B-21 describes the subelements of the `federated-scheme` element.

Table B-21 federated-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.
`<scope-name>`	Optional	Specifies the scope name for this configuration. The scope name is added (as a prefix) to all services generated by a cache factory. The name is used to isolate services when using multiple cache factories; thus, avoiding unintended joining of services with similar names from different configurations.
`<service-name>`	Optional	Specifies a name for the federated cache service instance that manages the cache that is created from this federated scheme. The federated cache service definition is defined within the `<services>` element in the `tangosol-coherence.xml` file. See FederatedCache Service Parameters. Different federated schemes can use different federated cache service instances to maintain separate caches. The default name if no name is specified is `FederatedCache`.
`<service-priority>`	Optional	Specifies the priority for the service thread. Legal values are from 1 to 10 where 10 is the highest priority. The default value is the `service-priority` value specified in the `tangosol-coherence.xml` descriptor. See the `service-priority` parameter in FederatedCache Service Parameters.
`<event-dispatcher-priority>`	Optional	Specifies the priority for the event dispatcher thread for each service. Legal values are from 1 to 10 where 10 is the highest priority. The default value is the `event-dispatcher-priority` value specified in the `tangosol-coherence.xml` descriptor. See the `event-dispatcher-priority` parameter in FederatedCache Service Parameters.
<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. You cannot change this element during a rolling restart. For example, you cannot change from java to pof serialization or the serializer class.

Table B-21 (Cont.) federated-scheme Subelements

Element	Required/ Optional	Description
<reliable-transport>	Optional	Specifies the transport protocol used by this service for reliable point-to-point communication. Specifying a value results in the use of a service-specific transport instance rather then the shared transport instance that is defined by the <unicast-listener> element. A service-specific transport instance can result in higher performance but at the cost of increased resource consumption and should be used sparingly for select, high priority services. In general, a shared transport instance uses less resource consumption than service-specific transport instances. Valid values are: • datagram – UDP protocol • tmb (default) – TCP/IP message bus protocol • tmbs – TCP/IP message bus protocol with SSL support. TMBS requires the use of an SSL socket provider. See socket-provider. • sdmb – Socket Direct Protocol (SDP) message bus. • sdmbs – SDP message bus with SSL support. SDMBS requires the use of an SSL socket provider. See socket-provider. The default value is the <reliable-transport> value specified in the tangosol-coherence.xml descriptor. See the reliable-transport parameter in FederatedCache Service Parameters.
<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 FederatedCache Service Parameters. 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	**Note:** The thread-count element is deprecated and is replaced by setting the thread-count-min and thread-count-max elements to the same value. Specifies the number of daemon threads used by the partitioned cache service. Legal values are positive integers, 0, or -1. The value 0 indicates that all relevant tasks are performed on the service thread. The value -1 indicates that tasks are performed on the caller's thread where possible.
<thread-count-max>	Optional	Specifies the maximum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers greater or equal to the value of the thread-count-min element. The default value is the thread-count-max value specified in the tangosol-coherence.xml descriptor. See the thread-count-max parameter in DistributedCache Service Parameters.

Table B-21 (Cont.) federated-scheme Subelements

Element	Required/ Optional	Description
<thread-count-min>	Optional	Specifies the minimum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers less than or equal to the value of the thread-count-max element. The default value is the thread-count-min value specified in the tangosol-coherence.xml descriptor. See the thread-count-max parameter in DistributedCache Service Parameters.
<worker-priority>	Optional	Specifies the priority for the worker threads. Legal values are from 1 to 10 where 10 is the highest priority. The default value is the worker-priority value specified in the tangosol-coherence.xml descriptor. See the worker-priority parameter in FederatedCache Service Parameters.
<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 FederatedCache Service Parameters.
<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 FederatedCache Service Parameters.
<partition-count>	Optional	Specifies the number of federated cache partitions. Each storage-enabled cluster member that is running the federated cache service manages a balanced number of partitions. Valid values are positive integers between 1 and 32767 and should be a prime number. A list of primes can be found at http://primes.utm.edu/lists/. The default value is the value specified in the tangosol-coherence.xml descriptor. See the partition-count parameter in FederatedCache Service Parameters.
<transfer-threshold>	Optional	Specifies the threshold for the primary buckets distribution in kilobytes. When a new node joins the federated 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 FederatedCache Service Parameters.
<backup-count>	Optional	Specifies the number of members of the federated 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. The default value is the backup-count value specified in the tangosol-coherence.xml descriptor. See FederatedCache Service Parameters.

Table B-21 (Cont.) federated-scheme Subelements

Element	Required/ Optional	Description
<backup-count-after-writebehind>	Optional	Specifies the number of members of the federated 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 federated 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.
<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 federated service to manage partition distribution. The default value is the partition-assignment-strategy value specified in the tangosol-coherence.xml descriptor. See the partition-assignment-strategy parameter in FederatedCache Service Parameters. • simple – The simple assignment strategy attempts to balance partition distribution while ensuring machine-safety. • mirror:<*service-name*> – The mirror assignment strategy attempts to co-locate the service's partitions with the partitions of the specified service. This strategy is used to increase the likelihood that key-associated, cross-service cache access remains local to a member. • custom – a class that implements the com.tangosol.net.partition.PartitionAssignmentStrategy interface. Enter a custom strategy using the <instance> element.
<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-min 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 FederatedCache Service Parameters.

Table B-21 (Cont.) federated-scheme Subelements

Element	Required/ Optional	Description
`<task-timeout>`	Optional	Specifies the timeout value in milliseconds for requests executing on the service worker threads. This attribute applies only if the thread pool is used (the `thread-count-min` value is positive) and only applies to entry processor implementations that implement the `PriorityTask` interface. 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 FederatedCache Service Parameters.
`<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 FederatedCache Service Parameters.
`<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-21 (Cont.) federated-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. See the subelements for instance. 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.
<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.
<persistence>	Optional	Specifies the persistence-related configuration for a federated cache service.
<partitioned-quorum-policy-scheme>	Optional	Specifies quorum policy settings for the federated cache service.
<listener>	Optional	Specifies an implementation of a `MapListener` which is notified of events occurring on the cache.

Table B-21 (Cont.) federated-scheme Subelements

Element	Required/ Optional	Description
<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.
<interceptors>	Optional	Specifies any number of event interceptors that process events for all caches of a specific service.
<async-backup>	Optional	Specifies whether the partitioned cache service backs up data asynchronously while concurrently responding to the client. Asynchronous backup is often used to increase client performance. However, applications that require strict data integrity must be designed and tested to ensure that data is not at risk. Legal values are true or false. The default value is false.
<journalcache-highunits>	Optional	Specifies either a memory limit or maximum number of cache entries that the federated cache service's internal cache holds for replication to remote participants. It provides a mechanism to constrain resources utilized by federation service internal caches. Once the high-units is reached, the federation service moves all the remote participants to ERROR state and drops all the pending entries from the internal cache. Valid values are memory values (e.g. "1G") or positive integers and 0. A memory value is treated as a memory limit on federation's backlog. If no units are specified, then the value is treated as a limit on the number of entries in the backlog. A value of 0 implies no limit. The default value is 0.
<journalcache-backing-map-scheme>	Optional	Specifies a backing-map-scheme-type that contains the backing map configuration for federation's internal caches. By default, federation uses Elastic Data for its internal caches.
<socket-provider>	Optional	Specifies the configuration for a socket and channel factory.
<address-provider>	Optional	Specifies either the local address (IP, or DNS name, and port) on which the TCP/IP server socket is bound or an implementation of the com.tangosol.net.AddressProvider interface that programmatically provides a socket address. The address-provider element also supports socket address references.
<load-balancer>	Optional	Specifies a pluggable strategy that is used by a federated service to distribute connections across the set of clustered federated service members. Legal values are: • federation – (default) This strategy attempts to distribute connections equally across federated service members based upon existing connection count and incoming message backlog. • client – This strategy relies upon the address provider implementation to dictate the distribution across federation service members. If no address provider implementation is provided, then each federation service member is tried 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.federation.FederatedServiceLoadBalancer interface.
<topologies>	Optional	Specifies the configuration information for one or more topology definitions.

flashjournal-scheme

Used in: back-scheme, backing-map-scheme, caching-schemes, 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.

This scheme uses the `com.tangosol.net.cache.CompactSerializationCache` 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-22 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.
`<class-name>`	Optional	Specifies a custom implementation of the simple serialization map cache. Any custom implementation must extend the `com.tangosol.net.cache.CompactSerializationCache` 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.
`<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 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 `ConfigurableCacheMap.EvictionPolicy` interface or extend the `AbstractEvictionPolicy` class.

Table B-22 (Cont.) flashjournal-scheme Subelements

Element	Required/ Optional	Description
<high-units>	Optional	Specifies the size limit of the cache. The value represents 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 the <unit-calculator> subelement). When this limit is exceeded, the cache begins the pruning process and evicts entries according to the eviction policy. Valid values are positive integers and 0. The value of this element must be in the following format: `(\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: • B or b (byte, 1) • K or k (kilo, 2^{10}) • M or m (mega, 2^{20}) • G or g (giga, 2^{30}) • T or t (tera, 2^{40}) The default value is 0 and implies no limit. If the value exceeds Integer.MAX_VALUE, then a unit factor is automatically used to adjust the value accordingly. If the value does not contain a factor, a factor of one is assumed.
<low-units>	Optional	Specifies 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 the <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 80% of the <high-units> setting (that is, for a <high-units> setting of 1000 the default <low-units> is 800).
<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 – 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. • <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-22 (Cont.) flashjournal-scheme Subelements

Element	Required/ Optional	Description
<unit-factor>	Optional	Specifies the factor by which the <low-units> and <high-units> settings are adjusted. For example, if you use a BINARY unit calculator, a factor of 1048576 can be used to specify megabytes instead of bytes in the <low-units> and <high-units> settings. The unit factor also adjusts the Cache MBean Units attribute. If a <high-units> is specified that would exceed the Integer.MAX_VALUE units, then a unit factor is automatically calculated on start up and does not need to be explicitly set. 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 proactively evicted. The value of this element must be in the following format: (\d)+((.)(\d)+)?[MS|ms|S|s|M|m|H|h|D|d]? where the first non-digits (from left to right) indicate the unit of time duration: • MS or ms (milliseconds) • S or s (seconds) • M or m (minutes) • H or h (hours) • D or d (days) If the value does not contain a unit, a unit of seconds is assumed. A value of zero implies no expiry. 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.

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.
<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.

grpc-acceptor

Used in: acceptor-config

Description

The `grpc-acceptor` element specifies the configuration information for a connection acceptor that accepts connections from remote gRPC clients. Use of the `grpc-acceptor` requires the Coherence gRPC Proxy module to be on the class path.

Elements

Table B-24 describes the subelements of the `grpc-acceptor` element.

Table B-24 grpc-acceptor subelements

Elements	Required/ Optional	Description
<local-address>	Required	Specifies the local address (IP or DNS name, and port) on which the gRPC acceptor is bound. The preconfigured system property overrides are `coherence.grpc.server.address` and `coherence.grpc.server.port`.
<socket-provider>	Required	Specifies either: the configuration for a socket provider, or it references a socket provider configuration that is defined within the `<socket-providers>` element of the operational deployment descriptor. The preconfigured system property overrides is `coherence.grpc.server.socketprovider`.

Table B-24 (Cont.) grpc-acceptor subelements

Elements	Required/ Optional	Description
`<in-process-name>`	Optional	Sets the name to use for the in-process gRPC server. The preconfigured system property override is `coherence.grpc.inprocess.name`.
`<grpc-controller>`	Optional	The grpc-controller element defines a `GrpcAcceptorController` implementation class or `class-factory` name and `init-params`.
`<channelz-max-page-size>`	Optional	Configures the maximum page size used by the gRPC debug Channelz service. See https://grpc.io/blog/a-short-introduction-to-channelz/. The default is 100.

http-acceptor

Used in acceptor-config

Description

The http-acceptor element specifies an acceptor for connections from remote REST clients over HTTP.

Elements

Table B-25 describes the subelements of the `http-acceptor` element.

Table B-25 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 the following implementations: • com.tangosol.coherence.rest.server.DefaultHttpServer (backed by Oracle's lightweight HTTP server) • com.tangosol.coherence.rest.server.NettyHttpServer (backed by Netty) • com.tangosol.coherence.rest.server.SimpleHttpServer (backed by Simple HTTP server) • com.tangosol.coherence.rest.server.JettyHttpServer (backed by Jetty HTTP server) • 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.
<socket-provider>	Optional	Specifies the configuration for a socket and channel factory.
<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 is used by the HTTP acceptor to load resource and provider classes.
<auth-method>	Optional	Specifies the authentication mechanism for the HTTP server. A client must have authenticated using the configured mechanism as a prerequisite to gaining access to any resources exposed by the server. Legal values are: • basic – This method requires the client to be authenticated using HTTP basic authentication. • cert – This method requires the client to be authenticated using client-side SSL certificate-based authentication. The certificate must be passed to the server to authenticate. An SSL-based socket provider must be configured using the <socket-provider> element. • cert+basic – This method requires the client to be authenticated using both client-side SSL certificate and HTTP basic authentication. • none (default) – This method does not require the client to be authenticated.

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-26 describes the elements you can define within the `identity-manager` element.

Table B-26 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>`	Optional	Specifies the private key password.
<password-provider>	Optional	Specifies a password provider implementation for retrieving the private key password. This element cannot be used with the `<password>` element.
`<password-url>`	Optional	Specifies a URL to use to load the password of a private key or key store. This element cannot be used with the `<password>` element.

incoming-message-handler

Used in: acceptor-config, initiator-config.

Description

The `<incoming-message-handler>` element contains the configuration information that is used to regulate client-to-cluster connection resource usage. Connection initiators and acceptors use this information to proactively detect and release connections that use excessive resources.

Elements

Table B-26 describes the elements you can define within the `incoming-message-handler` element.

Table B-27 incoming-message-handler Subelements

Element	Required/ Optional	Description
`max-message-size`	Optional	Specifies the size limit of messages being sent over Coherence*Extend connections. The value of this element must be in the following format: `(\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 kilo is assumed. Legal values are positive integers between 0 and `Integer.MAX_VALUE` (2147483647). The default value is `0` and indicates that there is no limit on the message size.

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-28 describes the subelements of the `initiator-config` element.

Table B-28 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.
<incoming-message-handler>	Optional	Specifies the configuration information that is used to regulate client-to-cluster connection resource usage.
<outgoing-message-handler>	Optional	Specifies the configuration information used by the connection initiator to detect dropped client-to-cluster connections.

`<use-filters>`	Optional	Contains the list of filter names to be used by this connection acceptor. For example, specifying `<use-filter>` as follows activates gzip compression for all network messages, which can help substantially with WAN and low-bandwidth networks.
		`<use-filters>`
		`  <filter-name>gzip</filter-name>`
		`</use-filters>`

Table B-28 (Cont.) initiator-config Subelements

Element	Required/ Optional	Description
<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. You cannot change this element during a rolling restart. For example, you cannot change from java to pof serialization or the serializer class.
`<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 the `<request-timeout>` value. See Table B-48.

init-param

Used in: init-params.

Description

Defines a parameter and value that is used in the constructor of a class when it is instantiated. See Cache of a Database.

Initialization parameters can be specified by type or name. When using the `<param-type>` element, an object of the specified type is instantiated and initialized with the value specified in `<param-value>`. A constructor for `<param-type>` with the value of `<param-value>` is called to instantiate the object. When using the `<param-name>` element, a constructor for `<param-name>` with the value of `<param-value>` is called to instantiate the object.

Elements

Table B-29 describes the subelements of the `init-param` element.

Table B-29 init-param Subelements

Element	Required/ Optional	Description

`<param-name>`	Optional	Specifies the name of the initialization parameter. For example:
		`<init-param>`
		`<param-name>sTableName</param-name>`
		`<param-value>EmployeeTable</param-value>`
		`</init-param>`
		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-param>`
		`<param-type>java.lang.String</param-type>`
		`<param-value>EmployeeTable</param-value>`
		`</init-param>`
		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.

Elements

Table B-30 describes the subelements of the `init-params` element.

Table B-30 init-params Subelements

Element	Required/ Optional	Description
<init-param>	Optional	Defines an individual initialization parameter.

instance

Used in: interceptor, serializer, 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. You can initialize parameters by writing XML which nests `<instance>` and `<class-scheme>` (or any other custom namespace) inside of `<param-value>` elements.

For example, given the following Java code:

```
public class MyClass
  {
  public MyClass(String s, OtherClass o, int i) { ... }
  }

public class OtherClass
  {
  public OtherClass(String s) { ... }
  }
```

You can initialize the `MyClass` and `OtherClass` classes by writing the following XML. In the XML, the `MyClass` class is initialized with the string `Hello World` and the integer `42`. The instance of the `OtherClass` class which appears in the `MyClass` class, is initialized with the string `Goodbye World`.

```
<instance>
  <class-name>MyClass</class-name>
    <init-params>
      <init-param>
        <param-value>Hello World</param-value>
      </init-param>
      <init-param>
        <param-value>
          <instance>
            <class-name>OtherClass</class-name>
              <init-params>
                <init-param>
                  <param-value>Goodbye World</param-value>
                </init-param>
              </init-params>
          </instance>
        </param-value>
      </init-param>
      <init-param>
        <param-value>42</param-value>
      </init-param>
```

```
    </init-params>
  </instance>
```

Elements

Table B-31 describes the subelements of the `instance` element.

Table B-31 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.

interceptor

Used in: interceptors

Description

The `interceptor` element defines the configuration associated with an event interceptor that is responsible for processing live events. Event interceptors implement the `com.tangosol.net.events.EventInterceptor` interface. The interface includes support to restrict an event interceptor to a specific cache or service. The interface also provides support for the `@Interceptor` annotation, which allows an implementation to register for a subset of events based on event types and to configure an event interceptor identifier and the ordering of event interceptors. Specifying an interceptor's identifier and ordering within the `interceptor` element overrides the settings in the implementation.

The Interceptor Element for Federation

Interceptors can be used within federation schemes to capture federation change records. When defined within federation schemes, the interceptor is applied only to the configured federated cache service. Interceptors that are defined as part of a participant definition in an operational file are applied to all federated cache services that use the participant. See participant.

Elements

Table B-32 describes the subelements of the `interceptor` element.

Table B-32 interceptor Subelements

Element	Required/ Optional	Description
`<name>`	Optional	Specifies a unique identifier for the interceptor.

Table B-32 (Cont.) interceptor Subelements

Element	Required/ Optional	Description
<order>	Optional	Specifies whether the interceptor is the first interceptor in a chain of interceptors. The legal values are LOW and HIGH. A value of HIGH indicates that the interceptor is first in the chain of interceptors. A value of LOW indicates no order preference. The default value is LOW.
<instance>	Required	Specifies the interceptor class to instantiate. The interceptor class must implement the EventInterceptor interface.

interceptors

Used in: cache-mapping, distributed-scheme, and federated-scheme

Description

The interceptors element contains any number of event interceptor definitions.

Elements

Table B-33 describes the subelements of the interceptors element.

Table B-33 interceptors Subelements

Element	Required/ Optional	Description
<interceptor>	Optional	Specifies an event interceptor implementation.

invocation-scheme

Used in: caching-schemes.

Description

Defines an Invocation Service (com.tangosol.net.InvocationService). The invocation service may be used to perform custom operations in parallel on any number of cluster nodes.

Elements

Table B-34 describes the subelements of the invocation-scheme element.

Table B-34 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.

Table B-34 (Cont.) invocation-scheme Subelements

Element	Required/ Optional	Description
<scope-name>	Optional	Specifies the scope name for this configuration. The scope name is added (as a prefix) to all services generated by a cache factory. The name is used to isolate services when using multiple cache factories; thus, avoiding unintended joining of services with similar names from different configurations.
<service-name>	Optional	Specifies the name of the service which manages invocations from this scheme. The slash (/) and colon (:) are reserved characters and cannot be used in service names.
<service-priority>	Optional	Specifies the priority for the service thread. Legal values are from 1 to 10 where 10 is the highest priority. The default value is the service-priority value specified in the tangosol-coherence.xml descriptor. See the service-priority parameter in Invocation Service Parameters.
<event-dispatcher-priority>	Optional	Specifies the priority for the event dispatcher thread for each service. Legal values are from 1 to 10 where 10 is the highest priority. The default value is the event-dispatcher-priority value specified in the tangosol-coherence.xml descriptor. See the event-dispatcher-priority parameter in Invocation Service Parameters.
<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. You cannot change this element during a rolling restart. For example, you cannot change from java to pof serialization or the serializer class.
<reliable-transport>	Optional	Specifies the transport protocol used by this service for reliable point-to-point communication. Specifying a value results in the use of a service-specific transport instance rather then the shared transport instance that is defined by the <unicast-listener> element. A service-specific transport instance can result in higher performance but at the cost of increased resource consumption and should be used sparingly for select, high priority services. In general, a shared transport instance uses less resource consumption than service-specific transport instances. Valid values are: • datagram – UDP protocol • tmb (default) – TCP/IP message bus protocol • tmbs – TCP/IP message bus protocol with SSL support. TMBS requires the use of an SSL socket provider. See socket-provider. • sdmb – Socket Direct Protocol (SDP) message bus. • sdmbs – SDP message bus with SSL support. SDMBS requires the use of an SSL socket provider. See socket-provider. The default value is the <reliable-transport> value specified in the tangosol-coherence.xml descriptor. See the reliable-transport parameter in Invocation Service Parameters.
<thread-count>	Optional	**Note:** The thread-count element is deprecated and is replaced by setting the thread-count-min and thread-count-max elements to the same value. Specifies the number of daemon threads used by the partitioned cache service. Legal values are positive integers, 0, or -1. The value 0 indicates that all relevant tasks are performed on the service thread. The value -1 indicates that tasks are performed on the caller's thread where possible.

Table B-34 (Cont.) invocation-scheme Subelements

Element	Required/ Optional	Description
`<thread-count-max>`	Optional	Specifies the maximum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers greater or equal to the value of the `thread-count-min` element. The default value is the `thread-count-max` value specified in the `tangosol-coherence.xml` descriptor. See the `thread-count-max` parameter in Invocation Service Parameters.
`<thread-count-min>`	Optional	Specifies the minimum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers less than or equal to the value of the `thread-count-max` element. The default value is the `thread-count-min` value specified in the `tangosol-coherence.xml` descriptor. See the `thread-count-max` parameter in Invocation Service Parameters.
`<worker-priority>`	Optional	Specifies the priority for the worker threads. Legal values are from 1 to 10 where 10 is the highest priority. The default value is the `worker-priority` value specified in the `tangosol-coherence.xml` descriptor. See the `worker-priority` parameter in Invocation Service Parameters.
`<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-min` 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 Invocation Service Parameters.
`<task-timeout>`	Optional	Specifies the default timeout value for tasks that can time 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-min` 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 Invocation Service Parameters.
`<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 `request-timeout` value specified in the `tangosol-coherence.xml` descriptor. See the `request-timeout` parameter in Invocation Service Parameters.

Table B-34 (Cont.) invocation-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 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-35 describes the subelements of the `invocation-service-proxy` element.

Table B-35 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-36 describes the subelements of the `key-associator` element.

Table B-36 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` parameter specified in the `tangosol.coherence.xml` descriptor. See DistributedCache Service Parameters.
`<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-37 describes the subelements of the `key-partitioning` element.

Table B-37 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` parameter specified in the `tangosol-coherence.xml` descriptor. See DistributedCache Service Parameters.
`<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-38 describes the elements you can define within the `key-store` element.

Table B-38 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.

Table B-38 (Cont.) key-store Subelements

Element	Required/ Optional	Description
<password-provider>	Optional	Specifies a password provider implementation for retrieving the key store password. This element cannot be used with the <password> element.
<type>	Optional	Specifies the type of a java.security.KeyStore instance. The default value is PKCS12. **Note**: The default key store type changed as of JDK 9. If you want to continue using JKS for backward compatibility, specify JKS in the <type> element.

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-39 describes the subelements of the listener element.

Table B-39 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: http-acceptor, tcp-acceptor, tcp-initiator

Description

The local-address element specifies the local address (IP, or DNS name, and port) to which a socket is bound.

A local address for the <tcp-acceptor> element specifies a TCP/IP server socket that is used by the proxy service to accept connections from Coherence*Extend clients. A local address for the <http-acceptor> element specifies a HTTP server socket that is used to accept connections from REST clients. The following example binds a server socket to 192.168.0.2:7077.

```
<local-address>
  <address>192.168.0.2</address>
  <port>7077</port>
</local-address>
```

A local address for the `<tcp-initiator>` element specifies a TCP/IP client socket that 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 `7077`:

```
<local-address>
  <address>192.168.0.1</address>
  <port>7077</port>
</local-address>
```

> **Note:**
>
> A socket address for the TCP/IP acceptor can also be defined using an `address-provider` element. See address-provider.

Elements

Table B-40 describes the subelements of the `local-address` element.

Table B-40 local-address Subelements

Element	Required/ Optional	Description
`<address>`	Optional	Specifies the address (IP or DNS name) on which a socket listens and publishes. If the address is a bind address, then the address may also be entered using CIDR notation as a subnet and mask (for example, `192.168.1.0/24`), which allows runtime resolution against the available local IP addresses. The bind address can also be an external NAT address that routes to a local address; however, both addresses must use the same port.
`<port>`	Optional	Specifies the port on which a TCP/IP socket listens and publishes. The legal values are from 0 to 65535. The default value is `0` and indicates that the listener ports are automatically assigned from a computer's available ephemeral ports so as to avoid port conflicts with other applications.
`<port-auto-adjust>`	Optional	Specifies whether the port automatically increments if the specified port cannot be bound to because it is already in use. Alternatively, port conflicts can be avoided by setting the `<port>` element to `0`. Valid values are `true`, `false`, or the upper limit on the port range. The lower limit is the value specified for the `<port>` element. The default value is `true`.

local-scheme

Used in: caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme, near-scheme, front-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.

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. 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. See read-write-backing-map-scheme.

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-41 describes the subelements of the `local-scheme` element.

Table B-41 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.
`<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.
`<scope-name>`	Optional	Specifies the scope name for this configuration. The scope name is added (as a prefix) to all services generated by a cache factory. The name is used to isolate services when using multiple cache factories; thus, avoiding unintended joining of services with similar names from different configurations.
`<service-name>`	Optional	Specifies the name of the service which manages caches created from this scheme. The slash (`/`) and colon (`:`) are reserved characters and cannot be used in service names. Services are configured from within the `<services>` element in the `tangosol-coherence.xml` descriptor. See Operational Configuration Elements.

Table B-41 (Cont.) local-scheme Subelements

Element	Required/ Optional	Description
<eviction-policy>	Optional	Specifies the type of eviction policy to use. The 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 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 ConfigurableCacheMap.EvictionPolicy interface or extend the AbstractEvictionPolicy class.
<high-units>	Optional	Specifies the size limit of the cache. The value represents 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 the <unit-calculator> subelement). When this limit is exceeded, the cache begins the pruning process and evicts entries according to the eviction policy. Valid values are positive integers and 0. The value of this element must be in the following format: (\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: • B or b (byte, 1) • K or k (kilo, 2^{10}) • M or m (mega, 2^{20}) • G or g (giga, 2^{30}) • T or t (tera, 2^{40}) The default value is 0 and implies no limit. If the value exceeds Integer.MAX_VALUE, then a unit factor is automatically used to adjust the value accordingly. If the value does not contain a factor, a factor of one is assumed.
<low-units>	Optional	Specifies 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 the <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 80% of the <high-units> setting (that is, for a <high-units> setting of 1000 the default <low-units> is 800).

Table B-41 (Cont.) local-scheme Subelements

Element	Required/ Optional	Description
<unit-calculator>	Optional	Specifies the type of unit calculator to use. A unit calculator is used to determine the cost (in "units") of a given object. 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 . • <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.
<unit-factor>	Optional	Specifies the factor by which the <low-units> and <high-units> settings are adjusted. For example, if you use a BINARY unit calculator, a factor of 1048576 can be used to specify megabytes instead of bytes in the <low-units> and <high-units> settings. The unit factor also adjusts the Cache MBean Units attribute. If a <high-units> is specified that would exceed the Integer.MAX_VALUE units, then a unit factor is automatically calculated on start up and does not need to be explicitly set. 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 proactively evicted. When using a cache store, 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.

Table B-41 (Cont.) local-scheme Subelements

Element	Required/ Optional	Description
<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.

memcached-acceptor

Used in: acceptor-config

Description

The memcached-acceptor element contains the configuration information for an acceptor that accepts connections from remote memcached clients over TCP/IP. The acceptor allows memcached clients to use a Coherence cache over the memcached binary protocol.

Elements

Table B-42 describes the subelements of the memcached-acceptor element.

Table B-42 memcached-acceptor Subelements

Element	Required/ Optional	Description
<cache-name>	Required	Specifies the cache that is used by memcached clients that connect to the memcached acceptor. The cache name is resolved to an actual cache scheme using cache mappings. See cache-mapping. The cache name must resolve to a partitioned cache scheme.
<interop-enabled>	Optional	Specifies wether the memcached acceptor can by-pass the configured cache service serializer while storing the values in the cache. This is only required when sharing data between Coherence*Extend and memcached clients. The assumption is that memcached clients are using a Coherence serializer, like the POF serializer, to convert the objects into byte[] and the cache service is also using the same serializer. Legal values are true or false. The default value is false.
<memcached-auth-method>	Optional	Specifies the authentication mechanism for the memcached acceptor. A client must authenticate using the configured mechanism to gain access to any resources exposed by the server. Legal values are plain (SASL PLAIN) and none. The default value is none.
<socket-provider>	Optional	Specifies the configuration for a socket and channel factory.
<address-provider>	Required	Specifies either the local address (IP, or DNS name, and port) on which the TCP/IP server socket is bound or an implementation of the com.tangosol.net.AddressProvider interface that programmatically provides a socket address. The address-provider element also supports socket address references.

name-service-addresses

Used in: tcp-initiator

Description

The `name-service-addresses` element contains the address (IP, or DNS name, and port) of one or more name service TCP/IP acceptors. A TCP/IP initiator uses this information to establish a connection with a remote cluster. The TCP/IP initiator attempts to connect to the addresses in a random order until either the list is exhausted or a connection is established. See Defining a Remote Cache in *Developing Remote Clients for Oracle Coherence*.

Elements

Table B-43 describes the subelements of the `name-service-addresses` element.

Table B-43 name-service-addresses Subelements

Element	Required/ Optional	Description
<socket-address>	Optional	Specifies the address (IP, or DNS name, and port) on which a name service TCP/IP acceptor is listening. Multiple `<socket-address>` elements can be defined. The `<socket-address>` element cannot be used together with an `<address-provider>` element or an `<address>` element.
<address-provider>	Optional	Specifies the address (IP, or DNS name, and port) on which a name service TCP/IP acceptor is listening or the configuration for a `com.tangosol.net.AddressProvider` implementation that supplies the address. The `address-provider` element also supports socket address references. The `<address-provider>` element cannot be used together with a `<socket-address>` element or an `<address>` element.
`<address>`	Optional	Specifies an IP address or DNS name. The bind address can also be an external NAT address that routes to a local address; however, both addresses must use the same port. Multiple `<address>` elements can be defined. The `<address>` element cannot be used together with an `<address-provider>` element or a `<socket-address>` element.

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 cache 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 Understanding Near Caches and Near Cache.

Note that:

- The front map is reset if the back cache is destroyed.
- The front map is reset if all storage nodes are gone.

- The front map is reset if an extend client is disconnected with the proxy and the invalidation strategy is set to `none`. Use `none` as the invalidation strategy if you want the front map to continue to functioning after proxy disconnect.

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-44 describes the subelements of the `near-scheme` element.

Table B-44 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.
`<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.
`<service-name>`	Optional	Specifies the name of the service which manages caches created from this scheme. The slash (`{{{{}/{}}}`) and colon (`{{{{}:{}}}`) are reserved characters and cannot be used in service names. Services are configured from within the `{{<services>}}` element in the `{{tangosol-coherence.xml}}` descriptor.
<init-params>	Optional	Specifies initialization parameters for custom near cache implementations.
<front-scheme>	Required	Specifies the cache to use as the front-tier cache.
<back-scheme>	Required	Specifies the cache to use as the back-tier cache.

Table B-44 (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. Legal values are: • auto (default) – This strategy is identical to the present strategy. • 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. Note that the front map is reset if an extend client is disconnected with the proxy. • 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. • logical – instructs a near cache to listen to all backing map events that are not synthetic deletes. A synthetic delete event could be emitted as a result of eviction or expiration. With this invalidation strategy, it is possible for the front map to contain cache entries that have been synthetically removed from the backing map. Any subsequent re-insertion of the entries to the backing map causes the corresponding entries in the front map to be invalidated.
<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-45 describes the subelements of the nio-file-manager element.

Table B-45 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.
<initial-size>	Optional	Specifies the initial buffer size in megabytes.The value of this element must be in the following format: (\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 mega is assumed. Legal values are positive integers between 1 and Integer.MAX_VALUE (2147483647). 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)+[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 mega is assumed. Legal values are positive integers between 1 and Integer.MAX_VALUE (2147483647). 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.

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-46 describes the subelement for the `operation-bundling` element.

Table B-46 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

> **Note:**
>
> The Optimistic cache type is deprecated. View, Near or Distributed/Partitioned Cache types should be used in its place. The read and write performance and memory usage characteristics, described in Table 12-1, can be used to determine which cache type to use.

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-47 describes the subelements of the `optimistic-scheme` element.

Table B-47 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.
<scope-name>	Optional	Specifies the scope name for this configuration. The scope name is added (as a prefix) to all services generated by a cache factory. The name is used to isolate services when using multiple cache factories; thus, avoiding unintended joining of services with similar names from different configurations.
<service-name>	Optional	Specifies the name of the service which manages caches created from this scheme. The slash (/) and colon (:) are reserved characters and cannot be used in service names. Services are configured from within the <services> parameter in tangosol-coherence.xml. See Operational Configuration Elements.
<service-priority>	Optional	Specifies the priority for the service thread. Legal values are from 1 to 10 where 10 is the highest priority. The default value is 10.
<event-dispatcher-priority>	Optional	Specifies the priority for the event dispatcher thread for each service. Legal values are from 1 to 10 where 10 is the highest priority. The default value is 10.
<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. You cannot change this element during a rolling restart. For example, you cannot change from java to pof serialization or the serializer class.
<reliable-transport>	Optional	Specifies the transport protocol used by this service for reliable point-to-point communication. Specifying a value results in the use of a service-specific transport instance rather then the shared transport instance that is defined by the <unicast-listener> element. A service-specific transport instance can result in higher performance but at the cost of increased resource consumption and should be used sparingly for select, high priority services. In general, a shared transport instance uses less resource consumption than service-specific transport instances. Valid values are: • datagram – UDP protocol • tmb (default) – TCP/IP message bus protocol • tmbs – TCP/IP message bus protocol with SSL support. TMBS requires the use of an SSL socket provider. See socket-provider. • sdmb – Socket Direct Protocol (SDP) message bus. • sdmbs – SDP message bus with SSL support. SDMBS requires the use of an SSL socket provider. See socket-provider. The default value is the <reliable-transport> value specified in the tangosol-coherence.xml descriptor. See the reliable-transport parameter in OptimisticCache Service Parameters.

Table B-47 (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.
`<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-47 (Cont.) optimistic-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 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-48 describes the subelements of the `outgoing-message-handler` element.

Table B-48 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 the heartbeat ping request before closing the underlying 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. The value must be less than or equal to the <heartbeat-interval> value. If a larger value is specified, the <heartbeat-interval> value will be used. The default value is the <request-timeout> value.
<max-message-size>	Optional	Specifies the size limit of messages being sent over Coherence*Extend connections. The value of this element must be in the following format: `(\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 kilo is assumed. Legal values are positive integers between 0 and `Integer.MAX_VALUE` (2147483647). The default value is `0` and indicates that there is no limit on the message size.

Table B-48 (Cont.) outgoing-message-handler Subelements

Element	Required/ Optional	Description
`<request-timeout>`	Optional	Specifies the maximum amount of time a client waits for a response message before abandoning the request. The connection is not closed until the `<heartbeat-timeout>` value is reached. **Note:** This element is not used by a proxy service acceptor, because proxies never send requests to extend clients. 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 30s.

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-49 describes the subelements of the `overflow-scheme` element.

Table B-49 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.
`<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.
<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.
`<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 than `com.tangosol.net.cache.SimpleOverflowMap`. Legal values are `true` or `false`. The default value is `false`.

Table B-49 (Cont.) overflow-scheme Subelements

Element	Required/ Optional	Description
<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 proactively evicted. The value of this element must be in the following format: `(\d)+((.)(\d)+)?[MS|ms|S|s|M|m|H|h|D|d]?` where the first non-digits (from left to right) indicate the unit of time duration: • MS or ms (milliseconds) • S or s (seconds) • M or m (minutes) • H or h (hours) • D or d (days) If the value does not contain a unit, a unit of seconds is assumed. A value of zero implies no expiry. 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.

page-size

Used in: paged-topic-scheme

Description

The page-size element specifies the target page size. The default value is 1MB.

paged-topic-scheme

Used in: caching-schemes

Description

The `paged-topic-scheme` defines topics where the storage for values and metadata is partitioned across cluster nodes.

Elements

Table C-1 describes the subelements of the `paged-topic-scheme` element.

Table B-50 paged-topic-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
`<scope-name>`	Optional	Specifies the scope name for this configuration. The scope name is added (as a prefix) to all services generated by a cache factory. The name is used to isolate services when using multiple cache factories; thus, avoiding unintended joining of services with similar names from different configurations.
`<service-name>`	Optional	Specifies a name for the distributed cache service instance that manages the topic that is created from this scheme. The distributed cache service definition is defined within the `<services>` element in the `tangosol-coherence.xml` file. See DistributedCache Service Parameters . Different distributed schemes can use different partitioned cache service instances to maintain separate topics. The slash (`/`) and colon (`:`) are reserved characters and cannot be used in service names.
`<service-priority>`	Optional	Specifies the priority for the service thread. Legal values are from 1 to 10 where 10 is the highest priority. The default value is the `service-priority` value specified in the `tangosol-coherence.xml` descriptor. See the `service-priority` parameter in DistributedCache Service Parameters.
`<event-dispatcher-priority>`	Optional	Specifies the priority for the event dispatcher thread for each service. Legal values are from 1 to 10 where 10 is the highest priority. The default value is the `event-dispatcher-priority` value specified in the `tangosol-coherence.xml` descriptor. See the `event-dispatcher-priority` parameter in DistributedCache Service Parameters.
`<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. You cannot change this element during a rolling restart. For example, you cannot change from java to pof serialization or the serializer class.

Table B-50 (Cont.) paged-topic-scheme Subelements

Element	Required/ Optional	Description
<reliable-transport>	Optional	Specifies the transport protocol used by this service for reliable point-to-point communication. Specifying a value results in the use of a service-specific transport instance rather then the shared transport instance that is defined by the <unicast-listener> element. A service-specific transport instance can result in higher performance but at the cost of increased resource consumption and should be used sparingly for select, high priority services. In general, a shared transport instance uses less resource consumption than service-specific transport instances. Valid values are: • datagram – UDP . • tmb (default) – TCP/IP message bus protocol. • tmbs – TCP/IP message bus protocol with SSL support. TMBS requires the use of an SSL socket provider. See socket-provider. • sdmb – Socket Direct Protocol (SDP) message bus. • sdmbs – SDP message bus with SSL support. SDMBS requires the use of an SSL socket provider. See socket-provider. The default value is the <reliable-transport> value specified in the tangosol-coherence.xml descriptor. See the reliable-transport parameter in DistributedCache Service Parameters. You cannot change this element from TLS to non-TLS during a rolling restart. You are however able to change between tmb or datagram during a rolling restart.
<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. 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-max>	Optional	Specifies the maximum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers greater or equal to the value of the thread-count-minelement. The default value is the thread-count-max value specified in the tangosol-coherence.xml descriptor. See the thread-count-max parameter in DistributedCache Service Parameters.
<thread-count-min>	Optional	Specifies the minimum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers less than or equal to the value of the thread-count-max element. The default value is the thread-count-min value specified in the tangosol-coherence.xml descriptor. See the thread-count-max parameter in DistributedCache Service Parameters

Table B-50 (Cont.) paged-topic-scheme Subelements

Element	Required/ Optional	Description
<worker-priority>	Optional	Specifies the priority for the worker threads. Legal values are from 1 to 10 where 10 is the highest priority. The default value is the worker-priority value specified in the tangosol-coherence.xml descriptor. See the worker-priority parameter in DistributedCache Service Parameters.
<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.
<partition-count>	Optional	Specifies the number of distributed cache partitions. Each storage-enabled cluster member that is running the distributed cache service manages a balanced number of partitions. Valid values are positive integers between 1 and 32767 and should be a prime number. A list of primes can be found at http://primes.utm.edu/lists/ . The default value is the value specified in the tangosol-coherence.xml descriptor. See the partition-count parameter in DistributedCache Service Parameters. You cannot change this element during a rolling restart.This may appear to work but you will receive a message like the following indicating the change has been ignored: <Warning> (thread=DistributedCache:PartitionedCache, member=8): This node is configured with a 'partition-count' value of 801, but the service senior is using a value of 257; overriding the local configuration.
<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.
<backup-count>	Optional	Specifies the number of members of the paged topic service that hold the backup data for each unit of storage in the topic. A value of 0 means that for abnormal termination, some portion of the data in the topic is lost. The default value is the backup-count value specified in the tangosol-coherence.xmldescriptor. See DistributedCache Service Parameters You cannot change this element during a rolling restart.This may appear to work but you will receive a message like the following indicating the change has been ignored: <Warning> (thread=DistributedCache:PartitionedCache, member=8): This node is configured with a 'backup-count' value of 2, but the service senior is using a value of 1; overriding the local configuration.

Table B-50 (Cont.) paged-topic-scheme Subelements

Element	Required/ Optional	Description
<partition-assignment-strategy>	Optional	Specifies the strategy that is used by a partitioned service to manage partition distribution. The default value is the partition-assignment-strategy value specified in the tangosol-coherence.xml descriptor. See the partition-assignment-strategy parameter in DistributedCache Service Parameters • simple – The simple assignment strategy attempts to balance partition distribution while ensuring machine-safety • mirror:<service-name> – The mirror assignment strategy attempts to co-locate the service's partitions with the partitions of the specified service. This strategy is used to increase the likelihood that key-associated, cross-service cache access remains local to a member. • custom – a class that implements the com.tangosol.net.partition.PartitionAssignmentStrategy interface. Enter a custom strategy using the <instance> element. You cannot change this element during a rolling restart.
<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]?
<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 instance. The MemberListener implementation receives cache service lifecycle events. The element is used as an alternative to programmatically adding a MapListener on a service.

Table B-50 (Cont.) paged-topic-scheme Subelements

Element	Required/ Optional	Description
<storage>	Optional	This enum type specifies the storage scheme used to hold topic values and metadata. Valid values are on-heap, flashjournal or ramjournal. Default value is on-heap.
<transient>	Optional	Specifies whether or not the topic values and metadata should be persisted using a persistence environment. Valid values are true or false. If set to false, a persistence environment is used to persist the contents of the topic values and metadata. If set to true, the topics values and metadata is assumed to be transient and its contents will not be recoverable upon cluster restart. The default value is false.
<persistence>	Optional	Specifies the persistence-related configuration for a partitioned cache service. If this element exists and no persistence.environment value is specified, the default is default-on-demand persistence.
<storage-authorizer>	Optional	Specifies a reference to a storage access authorizer that is defined in an operational configuration file. The storage access authorizer is used by a partitioned cache to authorize access to the underlying cache data. If configured, all read and write access to the data in the topic storage will be validated and/or audited by the configured authorizer. The following example references a storage access authorizer definition with the id attribute auditing. <storage-authorizer>auditing</storage-authorizer> See storage-authorizer.
<partitioned-quorum-policy>	Optional	Specifies quorum policy settings for the partitioned cache service.
<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.
<interceptors>	Optional	Specifies any number of event interceptors that process events for all caches of a specific distributed service.
<page-size>	Optional	The page-size element specifies the target page size.

Table B-50 (Cont.) paged-topic-scheme Subelements

Element	Required/ Optional	Description
`<expiry-delay>`	Optional	Specifies the amount of time since a value was published to a topic that it is available to be received by subscriber(s). An expired value is no longer accessible. 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.
`<high-units>`	Optional	Specifies the size limit of the topic. The value represents the maximum number of bytes for the unprocessed values retained by the topic. Dependent on how the publisher is configured and its threads try-with-resource state, the next send could be throttled by flow control, fail on full or proceed to send values to the topic. Valid values are positive integers and 0. The value of this element must be in the following format: `(\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: • B or b (byte, 1) • K or k (kilo, 2^{10}) • M or m (mega, 2^{20}) • G or g (giga, 2^{30}) • T or t (tera, 2^{40}) The default value is `0` and implies no limit. If the value exceeds `Integer.MAX_VALUE`, then a unit factor is automatically used to adjust the value accordingly. If the value does not contain a factor, a factor of one is assumed.

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 Serialization Paged Cache.

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 operation.

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
- nio-file-manager—uses NIO to implement memory-mapped file based 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, see Persistence (long-term storage) .

Elements

Table B-51 describes the subelements of the `paged-external-scheme` element.

Table B-51 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.
`<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.
<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.
<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.
<nio-file-manager>	Optional	Configures the paged external cache to use a memory-mapped file for cache storage.
`<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.

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-52 describes the subelements of the `partitioned-quorum-policy-scheme` element.

Table B-52 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.
`<distribution-quorum>`	Optional	Specifies the minimum number of storage members of a partitioned service that must be present to perform partition distribution. Valid values are non-negative integers.
`<restore-quorum>`	Optional	Specifies the minimum number of storage members of a partitioned service that must be present to restore lost primary partitions from backup. Valid values are non-negative integers.
`<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. Valid values are non-negative integers.
`<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. Valid values are non-negative integers.
`<recover-quorum>`	Optional	Specifies the minimum number of storage members of a partitioned service that must be present in order to recover orphaned partitions from the persistence storage, or assign empty partitions if the persistence storage is unavailable or lost. Valid values are non-negative integers. A value of zero enables the dynamic recovery policy, which ensures availability of all persisted state and is based on the last good cluster membership information to determine how many members must be present for the recovery. The default value is zero.

`<recovery-hosts>`	Optional	Specifies a reference to the set of host-addresses that is configured in an operational configuration file. The set of host-addresses must be represented by the set of storage members in order to recover orphaned partitions from the persistent storage, or assign empty partitions if the persistent storage is unavailable or lost.
		The following example references a host address list that is called `persistence-host-list`.
		`<partitioned-quorum-policy-scheme>`
		`  <recovery-hosts>persistence-host-list</recovery-hosts>`
		`</partitioned-quorum-policy-scheme>`
		See The address-provider Element for Persistence.
		Note: A host address list must not be specified if the `recover-quorum` element is set to `0`.

Table B-52 (Cont.) partitioned-quorum-policy-scheme Subelements

Element	Required/ Optional	Description
<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.

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-53 describes the subelements of the partition-listener element.

Table B-53 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 parameter in the tangosol-coherence.xml descriptor. See DistributedCache Service Parameters.
<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.

persistence

Used in distributed-scheme

Description

The `persistence` element contains the persistence-related configuration for a distributed cache service.

Elements

Table B-55 describes the subelements you can define within the `persistence` element.

Table B-54 persistence Subelements

Element	Required/ Optional	Description

`<environment>`	Optional	Specifies a reference to a persistence environment configuration that is defined in the operational configuration file. The persistence environment is used by the enclosing distributed cache service to persist the contents of backing maps. If configured, a persistence environment enables cache contents to be automatically recovered either after a cluster restart or loss of multiple members, or on-demand from a named snapshot.
		The following example references a persistence environment that is called `environment1`.
		`<persistence>`
		`  <environment>environment1</environment>`
		`</persistence>`
		See persistence-environments.

`<archiver>`	Optional	Specifies a reference to a snapshot archiver definition that is defined in the operational configuration file. The archiver is used to archive, retrieve, or purge persistent snapshots. The following example reference an archiver that is called `archiver1`.
		`<persistence>`
		`  <archiver>archiver1</archiver>`
		`</persistence>`
		See snapshot-archivers.

`<active-failure-mode>`	Optional	Specifies how the service responds to an unexpected failure while performing persistence operations in active persistence mode. Legal values are: • `stop-service` (default): persistence is critical and failures encountered while writing to or recovering from the active persistent store should result in stopping the service rather than continuing without persistence. • `stop-persistence`: persistence is desirable, but the service should continue servicing requests if there is a failure encountered while writing to or recovering from the active persistent store.

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-55 describes the subelements you can define within the `provider` element.

Table B-55 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-56 describes the subelements of the `proxy-config` element.

Table B-56 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-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-58 describes the subelements of the `proxy-quorum-policy-scheme` element.

Table B-57 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.
`<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.

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-58 describes the subelements of the `proxy-scheme` element.

Table B-58 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.
`<scope-name>`	Optional	Specifies the scope name for this configuration. The scope name is added (as a prefix) to all services generated by a cache factory. The name is used to isolate services when using multiple cache factories; thus, avoiding unintended joining of services with similar names from different configurations.
`<service-name>`	Optional	Specifies the name of the service. The slash (`/`) and colon (`:`) are reserved characters and cannot be used in service names.
`<service-priority>`	Optional	Specifies the priority for the service thread. Legal values are from 1 to 10 where 10 is the highest priority. The default value is `10`. The default value is the value specified in the `service-priority` parameter in the `tangosol-coherence.xml` descriptor. See Proxy Service Parameters.
`<event-dispatcher-priority>`	Optional	Specifies the priority for the event dispatcher thread for each service. Legal values are from 1 to 10 where 10 is the highest priority. The default value is the value specified in the `event-dispatcher-priority` parameter in the `tangosol-coherence.xml` descriptor. See Proxy Service Parameters.
`<reliable-transport>`	Optional	Specifies the transport protocol used by this service for reliable point-to-point communication. Specifying a value results in the use of a service-specific transport instance rather then the shared transport instance that is defined by the `<unicast-listener>` element. A service-specific transport instance can result in higher performance but at the cost of increased resource consumption and should be used sparingly for select, high priority services. In general, a shared transport instance uses less resource consumption than service-specific transport instances. Valid values are: • `datagram` – UDP protocol • `tmb` (default) – TCP/IP message bus protocol • `tmbs` – TCP/IP message bus protocol with SSL support. TMBS requires the use of an SSL socket provider. See socket-provider. • `sdmb` – Socket Direct Protocol (SDP) message bus. • `sdmbs` – SDP message bus with SSL support. SDMBS requires the use of an SSL socket provider. See socket-provider. The default value is the `<reliable-transport>` value specified in the `tangosol-coherence.xml` descriptor. See the `reliable-transport` parameter in Proxy Service Parameters.

Table B-58 (Cont.) proxy-scheme Subelements

Element	Required/ Optional	Description
<thread-count>	Optional	**Note:** The thread-count element is deprecated and is replaced by setting the thread-count-min and thread-count-max elements to the same value. Specifies the number of daemon threads used by the partitioned cache service. Legal values are positive integers, 0, or -1. The value 0 indicates that all relevant tasks are performed on the service thread. The value -1 indicates that tasks are performed on the caller's thread where possible.
<thread-count-max>	Optional	Specifies the maximum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers greater or equal to the value of the thread-count-min element. The default value is the value specified in the thread-count-max parameter in the tangosol-coherence.xml descriptor. See Proxy Service Parameters.
<thread-count-min>	Optional	Specifies the minimum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers less than or equal to the value of the thread-count-max element. The default value is the value specified in the thread-count-min parameter in the tangosol-coherence.xml descriptor. See Proxy Service Parameters.
<worker-priority>	Optional	Specifies the priority for the worker threads. Legal values are from 1 to 10 where 10 is the highest priority. The default value is the value specified in the worker-priority parameter in the tangosol-coherence.xml descriptor. See Proxy Service Parameters.
<task-hung-threshold>	Optional	Specifies the amount of time in milliseconds that a task can execute before it is considered hung. Note that 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-min 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 Proxy Service Parameters.
<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-min 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 Proxy Service Parameters.
<request-timeout>	Optional	Specifies the maximum amount of time a proxy waits for requests that are sent to other proxies of the same name. This element should not be used because requests are never sent between proxies.

Table B-58 (Cont.) proxy-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 causes the local node to exit gracefully, halting the process only when the duration of the `coherence.shutdown.timeout` system property exceeds. • `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.
<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.

Table B-58 (Cont.) proxy-scheme Subelements

Element	Required/ Optional	Description
<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.

ramjournal-scheme

Used in: back-scheme, backing-map-scheme, caching-schemes, 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.

This scheme uses the com.tangosol.net.cache.CompactSerializationClass 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-59 describes the subelements of the ramjournal-scheme element.

Table B-59 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.

Table B-59 (Cont.) ramjournal-scheme Subelements

Element	Required/ Optional	Description
<class-name>	Optional	Specifies a custom implementation of the simple serialization map cache. Any custom implementation must extend the com.tangosol.net.cache.CompactSerializationCache 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.
<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 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 ConfigurableCacheMap.EvictionPolicy interface or extend the AbstractEvictionPolicy class.
<high-units>	Optional	Specifies the size limit of the cache. The value represents 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 the <unit-calculator> subelement). When this limit is exceeded, the cache begins the pruning process and evicts entries according to the eviction policy. Valid values are positive integers and 0. The value of this element must be in the following format: (\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: • B or b (byte, 1) • K or k (kilo, 2^{10}) • M or m (mega, 2^{20}) • G or g (giga, 2^{30}) • T or t (tera, 2^{40}) The default value is 0 and implies no limit. If the value exceeds Integer.MAX_VALUE, then a unit factor is automatically used to adjust the value accordingly. If the value does not contain a factor, a factor of one is assumed.

Table B-59 (Cont.) ramjournal-scheme Subelements

Element	Required/ Optional	Description
<low-units>	Optional	Specifies 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 the <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 80% of the high-units setting (that is, for a <high-units> setting of 1000 the default <low-units> is 800).
<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 – 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. • <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.
<unit-factor>	Optional	Specifies the factor by which the <low-units> and <high-units> settings are adjusted. For example, if you use a BINARY unit calculator, a factor of 1048576 can be used to specify megabytes instead of bytes in the <low-units> and <high-units> settings. The unit factor also adjusts the Cache MBean Units attribute. If a <high-units> is specified that would exceed the Integer.MAX_VALUE units, then a unit factor is automatically calculated on start up and does not need to be explicitly set. The default value is 1.

Table B-59 (Cont.) ramjournal-scheme Subelements

Element	Required/ Optional	Description
<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 proactively evicted. The value of this element must be in the following format: (\d)+((.)(\d)+)?[MS|ms|S|s|M|m|H|h|D|d]? where the first non-digits (from left to right) indicate the unit of time duration: • MS or ms (milliseconds) • S or s (seconds) • M or m (minutes) • H or h (hours) • D or d (days) If the value does not contain a unit, a unit of seconds is assumed. A value of zero implies no expiry. 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.

read-write-backing-map-scheme

Used in: caching-schemes, backing-map-scheme.

Description

The read-write-backing-map-scheme defines a backing map which provides a size limited cache of a persistent store. See Caching Data Sources .

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-60 describes the subelements of the `read-write-backing-map-scheme` element.

Table B-60 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.
`<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.
`<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-60 (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-60 (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-60 (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.
<write-behind-remove>	Optional	Specifies whether remove operations should be added to the write-behind queue. If true, and the cachestore is write-behind enabled (write-delay is greater than zero), cache remove operations will be added to the write-behind queue and the cachestore remove operations will take place asynchronously. If false (default), the write-behind delay does not apply to remove operations and remove operations will be synchronous (that is, write-through). Valid values are true or false. You can also override the default behavior by using the coherence.rwbm.writebehind.remove.default system property. For example: To set the default write-behind-remove to true, specify the following: -Dcoherence.rwbm.writebehind.remove.default=true When cachestore remove is write-behind enabled, a cache get() operation returns null while the write-behind remove is pending. After the remove is complete, the get() operation will call cachestore's load() if the entry is not in 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. A TCP/IP initiator uses this information to establish a connection with a proxy service on a 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 connection is established. See Defining a Remote Cache in *Developing Remote Clients for Oracle Coherence*.

> **Note:**
>
> The `name-service-addresses` element can also be used to establish a connection with a proxy service on a a remote cluster. See name-service-addresses.

The following example configuration instructs the initiator to connect to `192.168.0.2:7077` and `192.168.0.3:7077` in a random order:

```
<remote-addresses>
  <socket-address>
    <address>192.168.0.2</address>
    <port>7077</port>
  </socket-address>
  <socket-address>
    <address>192.168.0.3</address>
    <port>7077</port>
  </socket-address>
</remote-addresses>
```

Elements

Table B-61 describes the subelements of the `remote-addresses` element.

Table B-61 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	Specifies the address (IP, or DNS name, and port) on which a TCP/IP acceptor is listening or the configuration for a `com.tangosol.net.AddressProvider` implementation that supplies the address. The `address-provider` element also supports socket address references. 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-62 describes the subelements of the `remote-cache-scheme` element.

Table B-62 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.
`<scope-name>`	Optional	Specifies the scope name for this configuration. The scope name is added (as a prefix) to all services generated by a cache factory. The name is used to isolate services when using multiple cache factories; thus, avoiding unintended joining of services with similar names from different configurations.
`<service-name>`	Optional	Specifies the name of the service which manages caches created from this scheme. The slash (`/`) and colon (`:`) are reserved characters and cannot be used in service names.
`<cluster-name>`	Optional	Contains the name of the cluster to connect to.
`<proxy-service-name>`	Optional	Specifies the proxy service name to which this remote service connects. This element is only used if this remote service uses a name service to lookup a proxy service. See name-service-addresses. The value must match the `<service-name>` value of the proxy service. This element can be omitted if the `<service-name>` value of this remote service matches the `<service-name>` value of the proxy service. See proxy-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`. A value of `false` indicates that key association processing is done by the extend client. If the value is set to `true`, then .NET and C++ clients must include a parallel Java implementation of the key class on the cluster cache servers. The default value is the value specified in the `tangosol-coherence.xml` descriptor. See RemoteCache Service Parameters.
<listener>	Optional	Specifies an implementation of a `com.tangosol.util.MapListener` which is notified of events occurring on the cache.

remote-grpc-cache-scheme

Used in: cachestore-scheme, caching-schemes, back-scheme.

Description

The `remote-grpc-cache-scheme` element contains the configuration information necessary to use a clustered cache from outside the cluster, connecting using gRPC.

Elements

Table B-63 describes the subelements of the `remote-grpc-cache-scheme` element.

Table B-63 remote-grpc-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.
`<scope-name>`	Optional	Specifies the scope name for this configuration. The scope name is added (as a prefix) to all services generated by a cache factory. The name is used to isolate services when using multiple cache factories; thus, avoiding unintended joining of services with similar names from different configurations.
`<remote-scope-name>`	Optional	The `remote-scope-name` element contains the scope name to be used to obtain a session on the proxy server. If this value is not set, but the `scope-name` for the parent cache configuration is set, then the `scope-name` will be used. If neither the `remote-scope-name` element nor the parent `scope-name` are set, then the default (empty) scope will be used on the proxy. Setting this value to `$DEFAULT$` will use the default (empty) scope on the proxy server. This is useful where the parent `scope-name` is configured for this configuration as the `remote-scope-name` cannot be set to an empty value in XML.
`<service-name>`	Optional	The `service-name` element contains the service name configuration information.
`<cluster-name>`	Optional	Contains the name of the cluster to connect to.
`<proxy-service-name>`	Optional	Specifies the proxy service name to which this remote service connects. This element is only used if this remote service uses a name service to lookup a proxy service. See name-service-addresses. The value must match the `<service-name>` value of the proxy service. This element can be omitted if the `<service-name>` value of this remote service matches the `<service-name>` value of the proxy service. See proxy-scheme.
`<grpc-channel>`	Optional	The channels element contains the configuration of a gRPC channel.
`<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).

Table B-63 (Cont.) remote-grpc-cache-scheme Subelements

Element	Required/ Optional	Description
`<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.
`<enable-tracing>`	Optional	Indicates if tracing is enabled.
`<thread-count>`	Optional	**Note:** The `thread-count` element is deprecated and is replaced by setting the `thread-count-min` and `thread-count-max` elements to the same value. Specifies the number of daemon threads used by the service. Legal values are positive integers, `0`, or `-1`. The value `0` indicates that all relevant tasks are performed on the service thread. The value `-1` indicates that tasks are performed on the caller's thread where possible.
`<thread-count-max>`	Optional	Specifies the maximum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers greater or equal to the value of the `thread-count-min` element. The default value is the number of processors.
`<thread-count-min>`	Optional	Specifies the minimum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers less than or equal to the value of the `thread-count-max` element. The default is the number of processors.
`<worker-priority>`	Optional	Specifies the priority for the worker threads. Legal values are from 1 to 10, where 10 is the highest priority. The default value is the `worker-priority` value specified in the `tangosol-coherence.xml` descriptor.
`<listener>`	Optional	Specifies an implementation of a `MapListener` which is notified of events occurring on the cache.

Table B-63 (Cont.) remote-grpc-cache-scheme 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-ack-required>	Optional	The heartbeat-ack-required setting requires the server to send an ack response to a client heart beat message.

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-64 describes the subelements of the remote-invocation-scheme element.

Table B-64 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.
<scope-name>	Optional	Specifies the scope name for this configuration. The scope name is added (as a prefix) to all services generated by a cache factory. The name is used to isolate services when using multiple cache factories; thus, avoiding unintended joining of services with similar names from different configurations.
<cluster-name>	Optional	Contains the name of the cluster to connect to.
<service-name>	Optional	Specifies the name of the service. The slash (/) and colon (:) are reserved characters and cannot be used in service names.

Table B-64 (Cont.) remote-invocation-scheme Subelements

Element	Required/ Optional	Description
<proxy-service-name>	Optional	Specifies the proxy service name to which this remote service connects. This element is only used if this remote service uses a name service to lookup a proxy service. See name-service-addresses. The value must match the <service-name> value of the proxy service. This element can be omitted if the <service-name> value of this remote service matches the <service-name> value of the proxy service. See proxy-scheme.
<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

> **Note:**
>
> The Replicated cache type is deprecated. View, Near or Distributed/Partitioned Cache types should be used in its place. The read and write performance and memory usage characteristics, described in Table 12-1, can be used to determine which cache type to use.

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-65 describes the subelements of the replicated-scheme element.

Table B-65 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.
<scope-name>	Optional	Specifies the scope name for this configuration. The scope name is added (as a prefix) to all services generated by a cache factory. The name is used to isolate services when using multiple cache factories; thus, avoiding unintended joining of services with similar names from different configurations.

Table B-65 (Cont.) replicated-scheme Subelements

Element	Required/ Optional	Description
<service-name>	Optional	Specifies the name of the service which manages caches created from this scheme. The slash (/) and colon (:) are reserved characters and cannot be used in service names. Services are configured from within the <services> element in the tangosol-coherence.xml file. See Operational Configuration Elements.
<service-priority>	Optional	Specifies the priority for the service thread. Legal values are from 1 to 10 where 10 is the highest priority. The default value is the service-priority value specified in the tangosol-coherence.xml descriptor. See ReplicatedCache Service Parameters.
<event-dispatcher-priority>	Optional	Specifies the priority for the event dispatcher thread for each service. Legal values are from 1 to 10 where 10 is the highest priority. The default value is the event-dispatcher-priority value specified in the tangosol-coherence.xml descriptor. See ReplicatedCache Service Parameters.
<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. You cannot change this element during a rolling restart. For example, you cannot change from java to pof serialization or the serializer class.
<reliable-transport>	Optional	Specifies the transport protocol used by this service for reliable point-to-point communication. Specifying a value results in the use of a service-specific transport instance rather then the shared transport instance that is defined by the <unicast-listener> element. A service-specific transport instance can result in higher performance but at the cost of increased resource consumption and should be used sparingly for select, high priority services. In general, a shared transport instance uses less resource consumption than service-specific transport instances. Valid values are: • datagram – UDP protocol • tmb (default) – TCP/IP message bus protocol • tmbs – TCP/IP message bus protocol with SSL support. TMBS requires the use of an SSL socket provider. See socket-provider. • sdmb – Socket Direct Protocol (SDP) message bus. • sdmbs – SDP message bus with SSL support. SDMBS requires the use of an SSL socket provider. See socket-provider. The default value is the <reliable-transport> value specified in the tangosol-coherence.xml descriptor. See the reliable-transport parameter in ReplicatedCache Service Parameters.
<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.

Table B-65 (Cont.) replicated-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 ReplicatedCache Service Parameters.
<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.
<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.) replicated-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 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 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.

resource-config

Used in: http-acceptor

Description

The `resource-config` element contains the configuration information for a class that extends the `com.sun.jersey.api.core.ResourceConfig` class. The instance is used by the HTTP acceptor to load resource and provider classes for the Coherence REST application that is mapped to the specified context path. Multiple resource configuration classes can be configured and mapped to different context paths.

Elements

Table B-66 describes the subelements of the `resource-config` element.

Table B-66 resource-config Subelements

Element	Required/ Optional	Description
`<context-path>`	Optional	Specifies a root URI path for a REST application. The first character of the path must be `/`. The default value is `/`.
<instance>	Optional	Contains the configuration information for a class that extends the `com.sun.jersey.api.core.ResourceConfig` class. The default instance is the `com.tangosol.coherence.rest.server.DefaultResourceConfig` class.

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.

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-67 describes the subelements of the `serializer` element.

Table B-67 serializer Subelements

Element	Required/ Optional	Description
<instance>	Optional	Contains the class configuration information for a `com.tangosol.io.Serializer` implementation.

socket-address

Used in: address-provider, remote-addresses

Description

The `socket-address` element specifies the address and port on which a TCP/IP acceptor is listening.

Elements

Table B-68 describes the subelements of the `socket-address` element.

Table B-68 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. If the address is a bind address, then the address may also be entered using CIDR notation as a subnet and mask (for example, `192.168.1.0/24`), which allows runtime resolution against the available local IP addresses. The bind address can also be an external NAT address that routes to a local address; however, both addresses must use the same port.
<port>	Required	Specifies the port on which a TCP/IP acceptor socket is listening. Legal values are from 0 to 65535. The default value is `0` and indicates that the listener port is automatically assigned from a computer's available ephemeral ports so as to avoid port conflicts with other applications.

socket-provider

Used in: tcp-acceptor, tcp-initiator, defaults, ssl, memcached-acceptor.

Description

The <socket-provider> element contains the configuration information for a socket and channel factory. The socket providers that are configured within the <tcp-acceptor> and <tcp-initiator> elements are for use with Coherence*Extend. A socket provider that is configured within the <http-acceptor> element is used by Coherence REST. A socket provider that is configured within the <memcached-acceptor> element is used by memcached clients. Socket

providers for TCMP are configured in an operational override 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-provider. The following socket providers are available: system (default), ssl, tcp, and sdp. 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>
```

The preconfigured override is coherence.socketprovider.

Elements

Table B-69 describes the subelements you can define within the socket-provider element.

Table B-69 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.
<tcp>	Optional	Specifies a socket provider that produces TCP-based sockets and channel implementations.
<sdp>	Optional	Specifies a socket provider that produce SDP-based sockets and channel implementations provided that the JVM and underlying network stack supports SDP.

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-70 describes the elements you can define within the ssl element.

Table B-70 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.

Table B-70 (Cont.) ssl Subelements

Element	Required/ Optional	Description
<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	Coherence provides a default implementation of HostnameVerifier. Use the <action> subelement to specify the action that the default hostname verifier should take during the SSL handshake if there is a mismatch between the host name in the URL and the host name in the digital certificate that the server sends back as part of the SSL connection. <action> takes one of the following two possible values: • **allow**: Allow all connections. There is no hostname verification. For backwards compatibility, this is the default option. • **default**: Use the default hostname verifier to verify the host. The host name verification passes if the host name in the certificate matches the host name in the URL to which the client connects. It also allows wild-card Subject Alternate Names (SAN) and Common Names (CN). If you do not want a match against CN if SAN DNS names are present, set the system property coherence.security.ssl.verifyCNAfterSAN to false. If the URL specifies localhost, you can set the system property coherence.security.ssl.allowLocalhost to true to allow 127.0.0.1, or the default IP address of the local machine. You can also specify 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. For more information about host name verification, see Using a Host Name Verification in *Securing Oracle Coherence*.
<socket-provider>	Optional	Specifies the configuration information for a delegate provider for SSL. Valid values are tcp and sdp. The default value is tcp.

Table B-70 (Cont.) ssl Subelements

Element	Required/ Optional	Description
<cipher-suites>	Optional	The cipher-suites element lists the names of the ciphers that are allowed or disallowed depending on the usage attribute. If the usage attribute on the element is "black-list", the specified ciphers will be removed from the default enabled cipher list. If the usage attribute on the element is "white-list", the specified ciphers will be the enabled ciphers. If the usage attribute is not defined, the cipher list will be considered a white-list.
<protocol-versions>	Optional	The protocol-versions element lists the names of the protocol versions that are allowed or disallowed depending on usage attribute. If the usage attribute on the element is "black-list", the specified protocol versions will be removed from the default enabled protocol list. If the usage attribute on the element is "white-list", the specified protocol versions will be the enabled protocols. If the usage attribute is not defined, the protocol list will be considered a white-list.

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 Explicitly Configuring Proxy Addresses in *Developing Remote Clients for Oracle Coherence*.

Elements

Table B-71 describes the subelements of the `tcp-acceptor` element.

Table B-71 tcp-acceptor Subelements

Element	Required/ Optional	Description
<socket-provider>	Optional	Specifies the configuration for a socket and channel factory.
<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	Specifies either the local address (IP, or DNS name, and port) on which the TCP/IP server socket is bound or an implementation of the `com.tangosol.net.AddressProvider` interface that programmatically provides a socket address. The `address-provider` element also supports socket address references. A `<tcp-acceptor>` element can include either a `<local-address>` or an `<address-provider>` element but not both.

Table B-71 (Cont.) tcp-acceptor Subelements

Element	Required/ Optional	Description
<keep-alive-enabled>	Optional	Indicates whether the SO_KEEPALIVE option is enabled on a TCP/IP socket. The SO_KEEPALIVE option detects idle sockets and determines whether the socket should be closed. 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, 2^{10}) • M or m (mega, 2^{20}) • G or g (giga, 2^{30}) If the value does not contain a factor, a factor of one is assumed. The default value is operating system 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, 2^{10}) • M or m (mega, 2^{20}) • G or g (giga, 2^{30}) If the value does not contain a factor, a factor of one is assumed. The default value is operating system dependent.
<listen-backlog>	Optional	Configures the size of the TCP/IP server socket backlog queue. Valid values are positive integers. The default value is operating system dependent.
<linger-timeout>	Optional	Specifies a value for the SO_LINGER option on a TCP/IP socket. The SO_LINGER option controls how long to wait before a socket is forcefully closed. 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 -1, which disables the linger timeout. A linger timeout is only set if the value is greater than zero.

Table B-71 (Cont.) tcp-acceptor Subelements

Element	Required/ Optional	Description
<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`.
`<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.
`<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`.

Table B-71 (Cont.) tcp-acceptor Subelements

Element	Required/ Optional	Description
`<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 Defining a Remote Cache in *Developing Remote Clients for Oracle Coherence*.

Elements

Table B-72 describes the subelements of the `tcp-initiator` element.

Table B-72 tcp-initiator Subelements

Element	Required/ Optional	Description
<socket-provider>	Optional	Specifies the configuration for a socket and channel factory.
<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.
<name-service-addresses>	Optional	Contains the address (IP, or DNS name, and port) of one or more name service TCP/IP acceptors. The TCP/IP connection initiator uses this information to establish a connection with a remote cluster. This element cannot be used if the `remote-addresses` element is used to configure remote addresses.

Table B-72 (Cont.) tcp-initiator Subelements

Element	Required/ Optional	Description
<remote-addresses>	Optional	Contains the address (IP, or DNS name, and port) of one or more TCP/IP connection acceptors. The TCP/IP connection initiator uses this information to establish a connection with a remote cluster. This element cannot be used if the `name-service-addresses` element is used.
<keep-alive-enabled>	Optional	Indicates whether the `SO_KEEPALIVE` option is enabled on a TCP/IP socket. The `SO_KEEPALIVE` option detects idle sockets and determines whether the socket should be closed. 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, 2^{10}) • M or m (mega, 2^{20}) • G or g (giga, 2^{30}) If the value does not contain a factor, a factor of one is assumed. The default value is operating system 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, 2^{10}) • M or m (mega, 2^{20}) • G or g (giga, 2^{30}) If the value does not contain a factor, a factor of one is assumed. The default value is operating system dependent.

Table B-72 (Cont.) tcp-initiator Subelements

Element	Required/ Optional	Description
<linger-timeout>	Optional	Specifies a value for the SO_LINGER option on a TCP/IP socket. The SO_LINGER option controls how long to wait before a socket is forcefully closed. 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 -1, which disables the linger timeout. A linger timeout is only set if the value is greater than zero.

topologies

Used in: federated-scheme

Description

The topologies element contains the configuration information for one or more topology definitions.

Elements

Table B-73 describes the subelements of the topologies element.

Table B-73 topologies Subelement

Element	Required/ Optional	Description
<topology>	Required	Specifies a named topology that is defined in the operational override configuration file.

topology

Used in: federated-scheme

Description

The topology element is used specify the topology that is being used by a federated scheme. Topologies are configured in the operational override file and define how data is synchronized between federation participants. See topology-definitions.

Elements

Table B-73 describes the subelements of the topology element.

Table B-74 topology Subelement

Element	Required/ Optional	Description
`<name>`	Required	Specifies the name of a topology that is defined in the operational override configuration file.
`<cache-name>`	Optional	Specifies the destination cache name on the remote participant.

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-75 describes the subelements of the `transactional-scheme` element.

Table B-75 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.
`<scope-name>`	Optional	Specifies the scope name for this configuration. The scope name is added (as a prefix) to all services generated by a cache factory. The name is used to isolate services when using multiple cache factories; thus, avoiding unintended joining of services with similar names from different configurations.
`<service-name>`	Optional	Specifies the name of the service which manages caches created from this scheme. The slash (`/`) and colon (`:`) are reserved characters and cannot be used in service names. The default service name if no service name is provided is `TransactionalCache`.

Table B-75 (Cont.) transactional-scheme Subelements

Element	Required/ Optional	Description
<service-priority>	Optional	Specifies the priority for the service thread. Legal values are from 1 to 10 where 10 is the highest priority. The default value is 10.
<event-dispatcher-priority>	Optional	Specifies the priority for the event dispatcher thread for each service. Legal values are from 1 to 10 where 10 is the highest priority. The default value is 10.
<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.
<reliable-transport>	Optional	Specifies the transport protocol used by this service for reliable point-to-point communication. Specifying a value results in the use of a service-specific transport instance rather then the shared transport instance that is defined by the <unicast-listener> element. A service-specific transport instance can result in higher performance but at the cost of increased resource consumption and should be used sparingly for select, high priority services. In general, a shared transport instance uses less resource consumption than service-specific transport instances. Valid values are: • datagram – UDP protocol • tmb (default) – TCP/IP message bus protocol • tmbs – TCP/IP message bus protocol with SSL support. TMBS requires the use of an SSL socket provider. See socket-provider. • sdmb – Socket Direct Protocol (SDP) message bus. • sdmbs – SDP message bus with SSL support. SDMBS requires the use of an SSL socket provider. See socket-provider. The default value is the <reliable-transport> value specified in the tangosol-coherence.xml descriptor. See the reliable-transport parameter in DistributedCache Service Parameters.
<thread-count>	Optional	**Note:** The thread-count element is deprecated and is replaced by setting the thread-count-min and thread-count-max elements to the same value. Specifies the number of daemon threads used by the partitioned cache service. Legal values are positive integers, 0, or -1. The value 0 indicates that all relevant tasks are performed on the service thread. The value -1 indicates that tasks are performed on the caller's thread where possible. Specifying a thread count changes the default behavior of the Transactional Framework's internal transaction caches that are used for transactional storage and recovery.
<thread-count-max>	Optional	Specifies the maximum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers greater or equal to the value of the thread-count-min element. The default value is the thread-count-max value specified in the tangosol-coherence.xml descriptor. See the thread-count-max parameter in DistributedCache Service Parameters.

Table B-75 (Cont.) transactional-scheme Subelements

Element	Required/ Optional	Description
<thread-count-min>	Optional	Specifies the minimum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers less than or equal to the value of the thread-count-max element. The default value is the thread-count-min value specified in the tangosol-coherence.xml descriptor. See the thread-count-max parameter in DistributedCache Service Parameters.
<worker-priority>	Optional	Specifies the priority for the worker threads. Legal values are from 1 to 10 where 10 is the highest priority. The default value is 5.
<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 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.
<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 in DistributedCache Service Parameters.

Table B-75 (Cont.) transactional-scheme Subelements

Element	Required/ Optional	Description
<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.
<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.
<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. 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.
<partition-assignment-strategy>	Optional	Specifies the strategy that is used by a partitioned service to manage partition distribution. Valid values are simple, or a class that implements the com.tangosol.net.partition.PartitionAssignmentStrategy interface. The simple strategy is a centralized distribution strategy that attempts to balance partition distribution, while ensuring machine-safety. The default value is simple. Enter the custom strategy using the <instance> element.
<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-min 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.

Table B-75 (Cont.) transactional-scheme Subelements

Element	Required/ Optional	Description
<task-timeout>	Optional	Specifies the timeout value in milliseconds for requests executing on the service worker threads. This attribute is applied only if the thread pool is used (the thread-count-min 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.
<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.
<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-75 (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 causes the local node to exit gracefully, halting the process only when the duration of the coherence.shutdown.timeout system property exceeds. • 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-76 describes the elements you can define within the trust-manager element.

Table B-76 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.

Table B-76 (Cont.) trust-manager Subelements

Element	Required/ Optional	Description
<key-store>	Optional	Specifies the configuration for a key store implementation.

view-scheme

Used in: caching-schemes.

Description

The `view-scheme` declaritively creates a `ContinuousQueryCache` backed by a clustered, standalone or remote scheme.

The `view-scheme` can be used as a replacement for replicated cache and thus by having a local cache on each node, the benefits of replicated cache can also be availed along with the benefits of distributed caches on the backend. See Understanding View Caches and View Cache.

Implementation

The view-scheme is implemented by the `com.tangosol.net.cache.ContinuousQueryCache` class.

Elements

Table B-77 describes the subelements of the `view-scheme` element.

Table B-77 view-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.
`<view-filter>`	Optional	Specifies an implementation of a `com.tangosol.util.Filter`, used by the associated view-scheme.
<back-scheme>	Optional	Specifies the back tier cache configuration information.
`<read-only>`	Optional	Specifies a readonly setting for the cachestore. If true the cache will only load data from cachestore for read operations and will not perform any writing to the cachestore when the cache is updated. The valid values are `true` or `false`. The default value is `false`.

Table B-77 (Cont.) view-scheme Subelements

Element	Required/ Optional	Description
`<reconnect-interval>`	Optional	The reconnect-interval indicates the period in which re-synchronization with the underlying cache will be delayed in the case the connection is severed. During this time period, local content can be accessed without triggering re-synchronization of the local content. 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 means that the view cannot be used when not connected. If not configured, a value of zero is assumed. Used in: view-scheme.
`<transformer>`	Optional	Specifies an implementation of a `com.tangosol.util.ValueExtractor` used by the associated view-scheme to transform values retrieved from the underlying cache, before storing them locally. If specified, this view can be set to read-only.
`<listener>`	Optional	Specifies an implementation of a `com.tangosol.util.MapListener` which is notified of events occurring on the cache.
`<service-name>`	Optional	Contains the configuration information of the service name.
`<cache-values>`	Optional	Specifies if the view is to maintain both keys and values or only keys. The valid values are `true` or `false`. The default value is `true`.

Cache Configuration Attribute Reference

The cache configuration attribute reference describes the attributes available in the cache configuration deployment descriptor.

Table B-78 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 System Property Overrides.

C

Topic Configuration Element Reference

The topic configuration element reference includes all non-terminal topic configuration elements.Each section includes instructions on how to use the element and also includes descriptions for all valid subelements.

> **Note:**
>
> Coherence XML configuration files are validated at runtime using the corresponding XML schemas. It is important that any custom configuration files conform to the schema, because something as simple as having elements in the wrong order will cause validation to fail. As validation errors produced by XML schema validators can be difficult to understand, we recommend that custom configuration files are edited in a development environment capable of validating the XML against the schema as it is being written. This can save a lot of time compared to debugging validation errors at runtime.

This appendix includes the following sections:

page-size

Used in: paged-topic-scheme

Description

The page-size element specifies the target page size. The default value is 1MB.

paged-topic-scheme

Used in: caching-schemes

Description

The `paged-topic-scheme` defines topics where the storage for values and metadata is partitioned across cluster nodes.

Elements

Table C-1 describes the subelements of the `paged-topic-scheme` element.

Table C-1 paged-topic-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
`<scope-name>`	Optional	Specifies the scope name for this configuration. The scope name is added (as a prefix) to all services generated by a cache factory. The name is used to isolate services when using multiple cache factories; thus, avoiding unintended joining of services with similar names from different configurations.
`<service-name>`	Optional	Specifies a name for the distributed cache service instance that manages the topic that is created from this scheme. The distributed cache service definition is defined within the `<services>` element in the `tangosol-coherence.xml` file. See DistributedCache Service Parameters . Different distributed schemes can use different partitioned cache service instances to maintain separate topics. The slash (`/`) and colon (`:`) are reserved characters and cannot be used in service names.
`<service-priority>`	Optional	Specifies the priority for the service thread. Legal values are from 1 to 10 where 10 is the highest priority. The default value is the `service-priority` value specified in the `tangosol-coherence.xml` descriptor. See the `service-priority` parameter in DistributedCache Service Parameters.
`<event-dispatcher-priority>`	Optional	Specifies the priority for the event dispatcher thread for each service. Legal values are from 1 to 10 where 10 is the highest priority. The default value is the `event-dispatcher-priority` value specified in the `tangosol-coherence.xml` descriptor. See the `event-dispatcher-priority` parameter in DistributedCache Service Parameters.
`<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. You cannot change this element during a rolling restart. For example, you cannot change from java to pof serialization or the serializer class.

Table C-1 (Cont.) paged-topic-scheme Subelements

Element	Required/ Optional	Description
<reliable-transport>	Optional	Specifies the transport protocol used by this service for reliable point-to-point communication. Specifying a value results in the use of a service-specific transport instance rather then the shared transport instance that is defined by the <unicast-listener> element. A service-specific transport instance can result in higher performance but at the cost of increased resource consumption and should be used sparingly for select, high priority services. In general, a shared transport instance uses less resource consumption than service-specific transport instances. Valid values are: • datagram – UDP . • tmb (default) – TCP/IP message bus protocol. • tmbs – TCP/IP message bus protocol with SSL support. TMBS requires the use of an SSL socket provider. See socket-provider. • sdmb – Socket Direct Protocol (SDP) message bus. • sdmbs – SDP message bus with SSL support. SDMBS requires the use of an SSL socket provider. See socket-provider. The default value is the <reliable-transport> value specified in the tangosol-coherence.xml descriptor. See the reliable-transport parameter in DistributedCache Service Parameters. You cannot change this element from TLS to non-TLS during a rolling restart. You are however able to change between tmb or datagram during a rolling restart.
<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. 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-max>	Optional	Specifies the maximum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers greater or equal to the value of the thread-count-minelement. The default value is the thread-count-max value specified in the tangosol-coherence.xml descriptor. See the thread-count-max parameter in DistributedCache Service Parameters.
<thread-count-min>	Optional	Specifies the minimum number of daemon threads. Usage of daemon threads varies for different service types. If zero or negative, the service does not use daemon threads and all relevant tasks are performed on the service thread. Furthermore, if negative, tasks are performed on the caller's thread where possible. Valid values are integers less than or equal to the value of the thread-count-max element. The default value is the thread-count-min value specified in the tangosol-coherence.xml descriptor. See the thread-count-max parameter in DistributedCache Service Parameters

Table C-1 (Cont.) paged-topic-scheme Subelements

Element	Required/ Optional	Description
<worker-priority>	Optional	Specifies the priority for the worker threads. Legal values are from 1 to 10 where 10 is the highest priority. The default value is the worker-priority value specified in the tangosol-coherence.xml descriptor. See the worker-priority parameter in DistributedCache Service Parameters.
<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.
<partition-count>	Optional	Specifies the number of distributed cache partitions. Each storage-enabled cluster member that is running the distributed cache service manages a balanced number of partitions. Valid values are positive integers between 1 and 32767 and should be a prime number. A list of primes can be found at http://primes.utm.edu/lists/ . The default value is the value specified in the tangosol-coherence.xml descriptor. See the partition-count parameter in DistributedCache Service Parameters. You cannot change this element during a rolling restart.This may appear to work but you will receive a message like the following indicating the change has been ignored: <Warning> (thread=DistributedCache:PartitionedCache, member=8): This node is configured with a 'partition-count' value of 801, but the service senior is using a value of 257; overriding the local configuration.
<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.
<backup-count>	Optional	Specifies the number of members of the paged topic service that hold the backup data for each unit of storage in the topic. A value of 0 means that for abnormal termination, some portion of the data in the topic is lost. The default value is the backup-count value specified in the tangosol-coherence.xmldescriptor. See DistributedCache Service Parameters You cannot change this element during a rolling restart.This may appear to work but you will receive a message like the following indicating the change has been ignored: <Warning> (thread=DistributedCache:PartitionedCache, member=8): This node is configured with a 'backup-count' value of 2, but the service senior is using a value of 1; overriding the local configuration.

Table C-1 (Cont.) paged-topic-scheme Subelements

Element	Required/ Optional	Description
<partition-assignment-strategy>	Optional	Specifies the strategy that is used by a partitioned service to manage partition distribution. The default value is the partition-assignment-strategy value specified in the tangosol-coherence.xml descriptor. See the partition-assignment-strategy parameter in DistributedCache Service Parameters • simple – The simple assignment strategy attempts to balance partition distribution while ensuring machine-safety • mirror:<service-name> – The mirror assignment strategy attempts to co-locate the service's partitions with the partitions of the specified service. This strategy is used to increase the likelihood that key-associated, cross-service cache access remains local to a member. • custom – a class that implements the com.tangosol.net.partition.PartitionAssignmentStrategy interface. Enter a custom strategy using the <instance> element. You cannot change this element during a rolling restart.
<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]?
<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 instance. The MemberListener implementation receives cache service lifecycle events. The element is used as an alternative to programmatically adding a MapListener on a service.

Table C-1 (Cont.) paged-topic-scheme Subelements

Element	Required/ Optional	Description
<storage>	Optional	This enum type specifies the storage scheme used to hold topic values and metadata. Valid values are on-heap, flashjournal or ramjournal. Default value is on-heap.
<transient>	Optional	Specifies whether or not the topic values and metadata should be persisted using a persistence environment. Valid values are true or false. If set to false, a persistence environment is used to persist the contents of the topic values and metadata. If set to true, the topics values and metadata is assumed to be transient and its contents will not be recoverable upon cluster restart. The default value is false.
<persistence>	Optional	Specifies the persistence-related configuration for a partitioned cache service. If this element exists and no persistence.environment value is specified, the default is default-on-demand persistence.
<storage-authorizer>	Optional	Specifies a reference to a storage access authorizer that is defined in an operational configuration file. The storage access authorizer is used by a partitioned cache to authorize access to the underlying cache data. If configured, all read and write access to the data in the topic storage will be validated and/or audited by the configured authorizer. The following example references a storage access authorizer definition with the id attribute auditing. <storage-authorizer>auditing</storage-authorizer> See storage-authorizer.
<partitioned-quorum-policy>	Optional	Specifies quorum policy settings for the partitioned cache service.
<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.
<interceptors>	Optional	Specifies any number of event interceptors that process events for all caches of a specific distributed service.
<page-size>	Optional	The page-size element specifies the target page size.

Table C-1 (Cont.) paged-topic-scheme Subelements

Element	Required/ Optional	Description
`<expiry-delay>`	Optional	Specifies the amount of time since a value was published to a topic that it is available to be received by subscriber(s). An expired value is no longer accessible. 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.
`<high-units>`	Optional	Specifies the size limit of the topic. The value represents the maximum number of bytes for the unprocessed values retained by the topic. Dependent on how the publisher is configured and its threads try-with-resource state, the next send could be throttled by flow control, fail on full or proceed to send values to the topic. Valid values are positive integers and 0. The value of this element must be in the following format: `(\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: • B or b (byte, 1) • K or k (kilo, 2^{10}) • M or m (mega, 2^{20}) • G or g (giga, 2^{30}) • T or t (tera, 2^{40}) The default value is `0` and implies no limit. If the value exceeds `Integer.MAX_VALUE`, then a unit factor is automatically used to adjust the value accordingly. If the value does not contain a factor, a factor of one is assumed.

subscriber-group

Used in: subscriber-groups

Description

The `subscriber-group` element defines a durable subscription group for a topic-mapping.

Elements

Table C-2 describes the subelements of the `subscriber-group` element.

Table C-2 subscriber-group Subelements

Element	Required/ Optional	Description
`<name>`	Required	Subscriber group name.

subscriber-groups

Used in: topic-mapping

Description

The `subscriber-groups` element enables defining one or more durable subscription group(s) in a topic-mapping. These groups will be created along with the topic and as such are ensured to exist before any data is published to the topic.

Elements

Table C-3 describes the subelements of the `subscriber-groups` element.

Table C-3 subscriber-groups Subelements

Element	Required/ Optional	Description
`<subscriber-group>`	Required	One or more subscriber-group elements that define durable subscription group(s) for the topic-mapping.

topic-mapping

Used in: topic-scheme-mapping

Description

Each `topic-mapping` element specifies the `paged-topic-scheme` what are to be used for a given topic name or topic name pattern used by an application.

Elements

Table C-4 describes the subelements of the `topic-mapping` element.

Table C-4 topic-mapping Subelements

Element	Required/ Optional	Description
<topic-name>	Required	Specifies a topic name or name pattern. The name is unique within topics created by a cache factory. The slash (/) and colon (:) are reserved characters and cannot be used in cache names. The following topic name patterns are supported: • Exact match. For example, MyTopic. • Prefix match using a wildcard (prefix*). For example, My* that matches to any topic name starting with My. • Any match using a wildcard (*). Matches to any topic name. If a topic name can be matched to multiple topic mappings, then exact matches are selected over wildcard matches. If no exact match is specified, then the last matching wildcard pattern (based on the order in which they are defined in the file) is selected.
<scheme-name>	Required	Contains the topic scheme name. The name is unique within a configuration file. Topic schemes are configured in caching-scheme element.
<value-type>	Optional	Specifies the fully-qualified name of the Java class for NamedTopic values.
<subscriber-groups>	Optional	The subscriber-group element enables defining one or more durable subscription group(s) in a topic-mapping. These groups will be created along with the topic and as such are ensured to exist before any data is published to the topic.

<init-params>	Optional	Allows specifying replaceable topic scheme parameters. During topic scheme parsing, any occurrence of any replaceable parameter in format param-name is replaced with the corresponding parameter value. Consider the following topic mapping example:
		
		`<topic-mapping>`
		`<topic-name>My*</topic-name>`
		`<scheme-name>my-scheme>/scheme-name>`
		`<value-type>String</value-type>`
		`<init-params>`
		`<init-param>`
		`<param-name>page-size</param-name>`
		`<param-value>10MB</param-value>`
		`</init-param>`
		`</init-params>`
		`</topic-mapping>`
		
		Any occurrence of the literal {*page-size*} will be replaced with the value of 10MB.

topic-scheme-mapping

Used in: cache-config

Description

Defines mappings between topic names, or name patterns, and a paged-topic-scheme.

Elements

Table C-5 describes the subelement you can define within the `topic-scheme-mapping` element.

Table C-5 topic-scheme-mapping Subelements

Element	Required/ Optional	Description
`<topic-mapping>`	Optional	The topic-mapping element contains a single binding between a topic name and a topic scheme this topic will use.

topic-storage-type

Used in: `paged-topic-scheme`

Description

This enum type specifies the storage scheme used to hold topic values and metadata. Valid values are `on-heap`, `flashjournal` or `ramjournal`. Default value is `on-heap`.

D

POF User Type Configuration Elements

The POF user type configuration reference provides a detailed description of the POF configuration deployment descriptor elements.See The PIF-POF Binary Format for details of the binary format.

> **Note:**
>
> Coherence XML configuration files are validated at runtime using the corresponding XML schemas. It is important that any custom configuration files conform to the schema, because something as simple as having elements in the wrong order will cause validation to fail. As validation errors produced by XML schema validators can be difficult to understand, we recommend that custom configuration files are edited in a development environment capable of validating the XML against the schema as it is being written. This can save a lot of time compared to debugging validation errors at runtime.

This appendix includes the following sections:

- POF Configuration Deployment Descriptor
- POF Configuration Element Reference

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/1.3/coherence-pof-config.xsd

The `<pof-config>` element is the root element of the POF configuration deployment descriptor and includes the XSD and namespace declarations. 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">
```

> **Note:**
>
> - 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>
```

POF Configuration Element Reference

The POF configuration element reference includes all non-terminal elements.Each section includes instructions on how to use the element and also includes descriptions for all valid subelements.

- default-serializer
- init-param
- init-params
- pof-config
- serializer
- user-type
- user-type-list

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-1 describes the subelements of the `default-serializer` element.

Table D-1 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-2 describes the subelements of the `init-param` element.

Table D-2 init-param Subelements

Element	Required/ Optional	Description
<param-type>	Required	The param-type element specifies the Java type of initialization parameter. Supported types are: • string—indicates that the value is a java.lang.String • boolean—indicates that the value is a java.lang.Boolean • int—indicates that the value is a java.lang.Integer • long—indicates that the value is a java.lang.Long • double—indicates that the value is a java.lang.Double • decimal—indicates that the value is a java.math.BigDecimal • file—indicates that the value is a java.io.File • date— indicates that the value is a java.sql.Date • time—indicates that the value is a java.sql.Timedatetime • datetime—indicates that the value is a java.sql.Timestamp • xml—indicates that the value is the entire init-param XmlElement. The value is converted to the specified type, and the target constructor or method must have a parameter of that type for the instantiation to succeed.
<param-value>	Required	The param-value element specifies a value of the initialization parameter. The value is in a format specific to the type of the parameter. There are four reserved values that can be specified. Each of these values is replaced at 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-3 describes the subelements of the init-params element.

Table D-3 init-params Subelements

Element	Required/ Optional	Description
<init-param>	Required	The init-param element provides a type for a configuration parameter and a corresponding value to pass as an argument.

pof-config

root element

Description

The `pof-config` element is the root element of the POF user type configuration descriptor.

Elements

Table D-4 describes the subelements of the `pof-config` element.

Table D-4 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-5 describes the subelements of the `serializer` element.

Table D-5 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-6 describes the subelements of the `user-type` element.

Table D-6 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-7 describes the subelements of the user-type-list element.

Table D-7 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 configuration 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

System Property Overrides

The system property reference describes the Coherence system property override feature and lists many of the predefined system properties that are used to change Coherence default settings.
This appendix contains the following sections:

Overview of System Property Overrides

Both the Coherence Operational Configuration and Coherence Cache Configuration deployment descriptors can assign a system property to any element that is 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, 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

Learn how to use the system property override feature by considering an example that overrides the default localhost value.The example configures a system property override that is used to change the IP address of a multi-home server in order to avoid using the default `localhost` value for an interface.
First, add a system-property to the `cluster-config`, `unicast-listener`, or `address` element. For example:

```
<address system-property="coherence.localhost">localhost</address>
```

Then use the system property to specify an IP address instead of the default `localhost`:

```
java -Dcoherence.localhost=192.168.0.301 -jar coherence.jar
```

Preconfigured Override Values

The Coherence system property override reference lists all the preconfigured system properties and the settings they override.

These system properties are defined in the `tangosol-coherence.xml` file and can be overridden in two ways:

- By providing your own override file which applies overrides/specializations to the `tangosol-coherence.xml` file.

  OR

- For service configuration items (a subset of the reference list in Table E-1), the relevant XML element in the cache configuration overrides the service's default value in the `tangosol-coherence.xml` file.

In either case, if the configuration item is specialized, the system-property XML attribute must be present at the point of override to ensure that any value passed through a JVM system property is applied.

The default to determine if a distributed-scheme is storage-enabled is defined in the `tangosol-coherence.xml` file, as seen in the example below:

```
<service id="3">
        <service-type>DistributedCache</service-type>
        <service-component>PartitionedService.PartitionedCache</service-
component>
        ...
        <init-params>
          ...
          <init-param id="4">
            <param-name>local-storage</param-name>
            <param-value system-
property="coherence.distributed.localstorage">true</param-value>
          </init-param>
```

You can specialize this example by providing the local-storage XML element as a child of the distributed-scheme within the cache configuration. The specialization ensures that a value is specified for that instance of the distributed cache. To ensure that the distributed cache honors the system-property in the `tangosol-coherence.xml` file, you should redefine it as shown below:

```
<distributed-scheme>
      ...
      <local-storage system-
property="coherence.distributed.localstorage">false</local-storage>
      ...
    </distributed-scheme>
```

Table E-1 Preconfigured System Property Override Values

System Property	Setting
coherence.authorized.hosts	Comma-separated list of IP addresses and/or host names authorized as cluster members. See the <host-address> subelement of authorized-hosts. Element authorized-hosts is a child element of cluster-config in operational configuration. **Note:** This system property is superceded if a custom authorized-hosts.host-filter is provided in an operational configuration override file, as described in Use a Filter Class to Determine Authorization.
coherence.cacheconfig	Cache configuration descriptor file name. See configurable-cache-factory-config.
coherence.cluster	Cluster name. See member-identity.
coherence.clusteraddress	Cluster (multicast) IP address. See <address> subelement of multicast-listener.
coherence.clusterport	Cluster (multicast) IP port. See <port> subelement of multicast-listener.
coherence.distributed.backup	Data backup storage location. See backup-storage/type subelement in DistributedCache Service Parameters.
coherence.distributed.backupcount	Number of data backups. See backup-count subelement in DistributedCache Service Parameters.
coherence.distributed.localstorage	Local partition management enabled. See local-storage subelement in DistributedCache Service Parameters.
coherence.distributed.threads	Thread pool size. See thread-count subelement in DistributedCache Service Parameters.
coherence.distributed.transfer	Partition transfer threshold. See transfer-threshold subelement inDistributedCache Service Parameters.
coherence.edition	Product edition. See license-config.
coherence.extend.authorized.hosts	Comma-separated list of IP addresses and/or host names authorized as extend clients to join proxy service. See the <host-address subelement of authorized-hosts. Element authorized-hosts is a child element of tcp-acceptor in cache configuration file. This system property is only in the default cache configuration file. An application needs to add the system property to the custom cache configuration file to the element caching-schemes.proxy-scheme.acceptor-config.tcp-acceptor.authorized-hosts.host-address. **Note:** This system property is superceded if a custom proxy-scheme.acceptor-config.tcp-acceptor.authorized-hosts.host-filter is provided in an application-provided cache configuration file.

Table E-1 (Cont.) Preconfigured System Property Override Values

System Property	Setting
coherence.federation.trace.logging	Controls the enabling or disabling of federation trace logging. The default value is false. See TraceLogging in FederationManagerMBean .
coherence.invocation.threads	Invocation service thread pool size. See thread-count subelement in Invocation Service Parameters.
coherence.localhost	Unicast IP address. See <address> subelement in unicast-listener.
coherence.localport	Unicast IP port. See <port> subelement in unicast-listener.
coherence.localport.adjust	Unicast IP port auto assignment. See <port-auto-adjust> subelement in unicast-listener.
coherence.log	Logging destination. See <logging-config-destination> subelement in logging-config.
coherence.log.level	Logging level. See <logging-config-level> subelement in logging-config.
coherence.log.limit	Log output character limit. See <logging-config-limit> subelement in logging-config.
coherence.machine	The computer's name as defined by the machine-name element. See member-identity.
coherence.management	JMX management mode. See management-config.
coherence.management.readonly	JMX management read-only flag. management-config.
coherence.management.remote	Remote JMX management enabled flag. See management-config.
coherence.member	Member name. See member-identity.
coherence.mode	Operational mode. See license-config.
coherence.override	Deployment configuration override filename.
coherence.priority	Priority. See member-identity.
coherence.process	Process name member-identity.
coherence.proxy.threads	Coherence*Extend service thread pool size. See thread-count subelement in Proxy Service Parameters.
coherence.rack	Rack name. See member-identity.
coherence.role	Role name. See member-identity.
coherence.security	Cache access security enabled flag. See security-config.
coherence.security.keystore	Security access controller keystore file name. See security-config.
coherence.security.permissions	Security access controller permissions file name. See security-config.
coherence.service.failure.policy	See <service-failure-policy> child element under service-guardian.
coherence.shutdownhook	Shutdown listener action. See shutdown-listener.
coherence.shutdown.timeout	Time duration string such as "2s" for 2 seconds, "5m" for 5 minutes, or "1h" for 1hour. Default is 2 minutes.
coherence.site	Site name. See member-identity.
coherence.tcmp.enabled	TCMP enabled flag. See <packet-publisher-enabled> subelement in packet-publisher.

Table E-1 (Cont.) Preconfigured System Property Override Values

System Property	Setting
`coherence.ttl`	Multicast packet time to live (TTL). See `<mulitcast-listener-ttl>` subelement in multicast-listener.
`coherence.wka`	Well known IP address. See well-known-addresses.

F

The PIF-POF Binary Format

Learn the binary streams for the Portable Object Format (POF) and the Portable Invocation Format (PIF) that are used to serialize objects in a platform and language neutral way.
This appendix includes the following sections:

Overview of 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 to 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 Using Portable Object Format .

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.

This section includes the following topics:

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 F-1 describes the three regions in the first octet.

Table F-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 F-2 describes the two regions in the trailing octets.

Table F-2 Regions in the Trailing Octet of a Packed Integer

Region Mask	Description
0x80	Continuation indicator
0x7F	integer value (next 7 binary LSDs)

Example F-1 illustrates writing a 32-bit integer value to an octet stream as supported in Coherence.

Example F-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 F-2 illustrates reading a 32-bit integer value from an octet stream as supported in Coherence.

Example F-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 F-3 illustrates some integer value examples.

Table F-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 F-4 lists the predefined identifiers.

Table F-4 Predefined Type Identifiers

Type ID	Description
-1 (0x40)	int16
-2 (0x41)	int32

Table F-4 (Cont.) Predefined Type Identifiers

Type ID	Description
-3 (0x42)	int64
-4 (0x43)	int128*
-5 (0x44)	float32
-6 (0x45)	float64
-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 F-5 lists the type identifiers that combine type and value.

Table F-5 Type Identifiers that Combine a Type and a Value

Type ID	Description
-33 (0x60)	boolean:false

Table F-5 (Cont.) Type Identifiers that Combine a Type and a Value

Type ID	Description
-34 (0x61)	boolean:true
-35 (0x62)	string:zero-length
-36 (0x63)	collection:empty
-37 (0x64)	reference:null
-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

Learn 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. This section includes the following topics:

- Int

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 F-6 illustrates these type identifiers.

Table F-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

Table F-6 (Cont.) Type Identifiers that Combine an int Data Type with a Value

Value	int16	int32	int64	int128
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 F-7 Type IDs of Integer Types that can be Coerced into Other Types

Type IDss	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
-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, the constant type+value identifiers listed in Table F-8 are used to indicate special values supported by IEEE 754r.

Table F-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, the constants in Table F-9 are used to indicate special values supported by IEEE-754

Table F-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, 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 F-10.

Table F-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). The compact form of integer values can be used for Octet values, with the integer value -1 being translated as 0xFF. See Coercion of Integer Types,

Table F-11 lists the integer values that may be used as Octet values.

Table F-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. The compact form of integer values may be used for Char values, with the integer value -1 being translated as 0xFFFF. See Coercion of Integer Types.

> **Note:**
>
> POF optimizes the storage of String data by using only one byte for each character when possible. Custom POF character codecs (ASCII for example) are not required and do not result in better performance.

Example F-3 illustrates writing a character value to an octet stream.

Example F-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 F-4 illustrates reading a character value from an octet stream.

Example F-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 F-12 illustrates the Char String formats.

Table F-12 Values for Char String Formats

Values	Char String Format
zero length	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 F-13 illustrates examples of Collection and Uniform Collection formats for several values.

Table F-13 Collection and Uniform Collection Formats for Various Values

Values	Collection Format	Uniform Collection Format
no value	0x63 (or 0x5500)	not applicable (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 F-14 illustrates examples of Array and Uniform Array formats for several values.

Table F-14 Array and Uniform Array Formats for Various Values

Values	Array Format	Uniform Array Format
no value	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 F-15 illustrates examples of Sparse Array and Uniform Sparse Array formats for several values.

Table F-15 Sparse Array and Uniform Sparse Array Formats for Various Values

Values	Sparse Array format	Uniform Sparse Array format
no value	0x63 (or 0x590040)	0x63 (or 0x5A410040) – This example assumes an element type of Int32.
1	0x5901006A40	0x5A4101000140
1,2,3	0x5903006A016B026C40	0x5A4103000101020203 40
1,,,,5,,,,9	0x5909006A046E087240	0x5A4109000104050809 40
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 F-16 illustrates several examples of key/value pairs and their corresponding binary format.

Table F-16 Binary Formats for Key/Value Pairs

Values	Binary format
no value	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 F-17 illustrates several examples of the binary formats for Key/Value pairs where the Keys are of uniform type.

Table F-17 Binary Formats for Key/Value Pairs where Keys are of Uniform Type

Values	Binary format
no value	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 F-18 illustrates several examples of the binary formats for Key/Value pairs where the Keys and Values are of uniform type.

Table F-18 Binary Formats for Key/Value Pairs where Keys and Values are of Uniform Type

Values	Binary format
no value	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 F-19 illustrates examples of binary formats for several "by reference" semantics.

Table F-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.

This section includes the following topic:

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.