

Oracle® Coherence

Developer's Guide

Release 3.6.1

E15723-03

December 2010

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

Oracle Coherence Developer's Guide, Release 3.6.1

E15723-03

Copyright © 2008, 2010, Oracle and/or its affiliates. All rights reserved.

Primary Author: Joseph Ruzzi

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Preface	xxxiii
Audience.....	xxxiii
Documentation Accessibility	xxxiii
Related Documents	xxxiv
Conventions	xxxiv

Part I Getting Started

1 Introduction

Basic Concepts	1-1
Clustered Data Management.....	1-1
A single API for the logical layer, XML configuration for the physical layer	1-2
Caching Strategies.....	1-2
Data Storage Options.....	1-2
Serialization Options	1-3
Configurability and Extensibility	1-3
Namespace Hierarchy	1-3
Read/Write Caching	1-4
NamedCache.....	1-4
Requirements for Cached Objects.....	1-4
NamedCache Usage Patterns	1-5
Querying the Cache	1-6
Transactions	1-6
HTTP Session Management.....	1-6
Invocation Service	1-7
Events.....	1-7
Object-Relational Mapping Integration	1-7
C++/.NET Integration	1-7
Management and Monitoring.....	1-8

2 Installing Oracle Coherence for Java

System Requirements.....	2-1
Extracting the Distribution.....	2-1
Setting Environment Variables.....	2-2
Running Coherence for the First Time.....	2-2

Create a Basic Cluster	2-2
Create a Cache	2-3

3 Understanding Configuration

Overview of the Default Configuration Files	3-1
Specifying an Operational Configuration File	3-2
Using the Default Operational Override File.....	3-3
Specifying an Operational Override File	3-3
Defining Override Files for Specific Operational Elements.....	3-4
Viewing Which Operational Override Files are Loaded.....	3-5
Specifying a Cache Configuration File	3-5
Using a Default Cache Configuration File.....	3-6
Overriding the Default Cache Configuration File.....	3-7
Using the Cache Configuration File System Property	3-7
Viewing Which Cache Configuration File is Loaded	3-8
Specifying a POF Configuration File	3-8
Using the POF Configuration File System Property	3-9
Combining Multiple POF Configuration Files	3-9
Viewing Which POF Configuration Files are Loaded	3-10
Specifying Management Configuration Files	3-10
Specifying a Custom Report Group Configuration File.....	3-11
Overriding the Default Report Group Configuration File.....	3-11
Using the Report Group Configuration File System Property	3-12
Specifying an MBean Configuration File.....	3-12
Using the Default MBean Configuration Override File	3-13
Using the MBean Configuration File System Property	3-13
Viewing Which Management Configuration Files are Loaded.....	3-13
Understanding the XML Override Feature	3-14
Using the Predefined Override Files.....	3-14
Defining Custom Override Files	3-15
Defining Multiple Override Files for the Same Element.....	3-17
Changing Configuration Using System Properties	3-17
Using Preconfigured System Properties.....	3-18
Creating Custom System Properties	3-20

4 Building Your First Coherence Application

Step 1: Define the Example Cache	4-1
Step 2: Configure and Start the Example Cluster	4-2
Step 3: Create and Run a Basic Coherence Standalone Application	4-3
Create the Sample Standalone Application.....	4-3
Run the Sample Standalone Application.....	4-4
Verify the Example Cache.....	4-4
Step 4: Create and Run a Basic Coherence JavaEE Web Application	4-5
Create the Sample Web Application	4-5
Deploy and Run the Sample Web Application.....	4-6
Verify the Example Cache.....	4-6
Using JDeveloper for Coherence Development	4-7

Running Coherence in JDeveloper	4-7
Viewing Thread Dumps in JDeveloper.....	4-10
Creating Configuration Files in JDeveloper.....	4-10

Part II Using Data Clusters

5 Cluster Services Overview

6 Understanding TCMP

Overview	6-1
Protocol Reliability	6-2
Protocol Resource Utilization	6-2
Protocol Tunability.....	6-2
Multicast Scope.....	6-2
Disabling Multicast	6-2

7 Setting Single Server Mode

Setting Single Server Mode in the Operation Configuration Descriptor	7-1
Setting Single Server Mode on the Command Line.....	7-2

8 Dynamically Managing Cluster Membership

Cluster and Service Objects	8-1
Member object	8-1
Listening to Member Events	8-2

9 Using Network Filters

Compression Filters	9-1
Encryption Filters	9-1
Symmetric Encryption Filter	9-1
Symmetric Encryption Filter Parameters	9-2
PKCS Encryption Filter	9-2
PKCS Encryption Filter Parameters	9-3
Configuring Filters.....	9-4
Creating a Custom Filter	9-5

Part III Using Caches

10 Introduction to Caches

Distributed Cache	10-1
Replicated Cache	10-5
Optimistic Cache	10-7
Near Cache.....	10-7
Local Cache	10-9
Remote Cache.....	10-10

Summary of Cache Types	10-10
11 Configuring Caches	
Overview	11-1
Defining Cache Mappings	11-2
Using One-to-One Cache Mappings	11-2
Using Cache Name Pattern Mappings.....	11-2
Specifying Initialization Parameters in a Mapping.....	11-3
Defining Cache Schemes	11-4
Defining Distributed Cache Schemes.....	11-4
Defining Replicated Cache Schemes	11-5
Defining Optimistic Cache Schemes	11-6
Defining Local Cache Schemes	11-6
Controlling the Growth of a Local Cache.....	11-7
Defining Near Cache Schemes	11-8
Near Cache Invalidation Strategies.....	11-9
Using Scheme Inheritance	11-9
Using Cache Scheme Properties	11-11
Using Parameter Macros	11-12
12 Implementing Storage and Backing Maps	
Cache Layers.....	12-1
Local Storage	12-2
Operations	12-3
Capacity Planning	12-4
Partitioned Backing Maps	12-5
13 Read-Through, Write-Through, Write-Behind, and Refresh-Ahead Caching	
Pluggable Cache Store.....	13-1
Read-Through Caching	13-1
Write-Through Caching	13-2
Write-Behind Caching	13-3
Write-Behind Requirements.....	13-4
Refresh-Ahead Caching	13-5
Selecting a Cache Strategy.....	13-6
Read-Through/Write-Through versus Cache-Aside	13-6
Refresh-Ahead versus Read-Through.....	13-6
Write-Behind versus Write-Through	13-7
Creating a CacheStore Implementation	13-7
Plugging in a CacheStore Implementation	13-7
Sample CacheStore	13-8
Sample Controllable CacheStore	13-14
Implementation Considerations.....	13-18
Idempotency	13-18
Write-Through Limitations.....	13-18
Cache Queries	13-18

Re-entrant Calls	13-18
Cache Server Classpath	13-19
CacheStore Collection Operations	13-19
Connection Pools	13-19
14 Serialization Paged Cache	
Understanding Serialization Paged Cache	14-1
Configuring Serialization Paged Cache	14-1
Optimizing a Partitioned Cache Service	14-2
Configuring for High Availability	14-2
Configuring Load Balancing and Failover	14-2
Supporting Huge Caches	14-2
15 Cache Configurations by Example	
Local Caches (accessible from a single JVM)	15-1
In-memory Cache	15-2
NIO In-memory Cache	15-2
Size Limited In-memory Cache	15-2
In-memory Cache with Expiring Entries	15-2
Cache on Disk	15-3
Size Limited Cache on Disk	15-3
Persistent Cache on Disk	15-3
In-memory Cache with Disk Based Overflow	15-4
Cache of a Database	15-4
Clustered Caches (accessible from multiple JVMs)	15-5
Replicated Cache	15-5
Replicated Cache with Overflow	15-5
Partitioned Cache	15-6
Partitioned Cache with Overflow	15-6
Partitioned Cache of a Database	15-6
Partitioned Cache with a Serializer	15-7
Local Cache of a Partitioned Cache (Near cache)	15-7
Part IV Using the Programming API	
16 Serializing Objects	
17 Using Portable Object Format	
Overview	17-1
Working with POF	17-2
Implementing the PortableObject interface	17-2
Implementing the PofSerializer interface:	17-2
Assigning POF indexes	17-3
Using the ConfigurablePofContext Class	17-3
Configuring Coherence to Use the ConfigurablePofContext Class	17-4

Configure the ConfigurablePofContext Class Per Service.....	17-4
Configure the ConfigurablePofContext Class for All Services	17-5
Configure the ConfigurablePofContext Class For the JVM.....	17-6
Using POF Extractors and POF Updaters	17-6
Navigating a POF object.....	17-6
Using PofExtractors	17-8
Using PofUpdaters.....	17-8
 18 Pre-Loading the Cache	
Performing Bulk Loading and Processing.....	18-1
Bulk Writing to a Cache	18-1
Efficient processing of filter results	18-2
A Bulk Loading and Processing Example	18-4
Performing Distributed Bulk Loading.....	18-9
A Distributed Bulk Loading Example.....	18-9
 19 Using Cache Events	
Listener Interface and Event Object	19-1
Caches and Classes that Support Events	19-3
Signing Up for All Events.....	19-4
Using an Inner Class as a MapListener.....	19-5
Configuring a MapListener for a Cache	19-5
Signing up for Events on specific identities	19-6
Filtering Events	19-6
"Lite" Events	19-7
Advanced: Listening to Queries	19-8
Filtering Events Versus Filtering Cached Data.....	19-9
Advanced: Synthetic Events	19-9
Advanced: Backing Map Events.....	19-10
Producing Readable Backing MapListener Events from Distributed Caches.....	19-11
Advanced: Synchronous Event Listeners	19-13
 20 Querying Data In a Cache	
Query Overview	20-1
Query Concepts	20-1
Performing Simple Queries	20-2
Using Query Indexes	20-3
Creating an Index.....	20-3
Creating User-Defined Indexes.....	20-4
Implementing the MapIndex Interface	20-4
Implementing the IndexAwareExtractor Interface	20-5
Using a Conditional Index.....	20-5
Batching Queries and Memory Usage.....	20-6
Queries Involving Multi-Value Attributes.....	20-8
ChainedExtractor	20-8

21	Using Continuous Query Caching	
	Uses of Continuous Query Caching	21-1
	The Coherence Continuous Query Cache	21-2
	Constructing a Continuous Query Cache	21-2
	Cleaning up the resources associated with a ContinuousQueryCache	21-3
	Caching only keys, or caching both keys and values.....	21-3
	CacheValues Property and Event Listeners	21-3
	Listening to the ContinuousQueryCache	21-3
	Achieving a Stable Materialized View	21-4
	Support for Synchronous and Asynchronous Listeners	21-5
	Making the ContinuousQueryCache Read-Only	21-5
22	Processing Data In a Cache	
	Targeted Execution	22-1
	Parallel Execution	22-1
	Query-Based Execution	22-2
	Data-Grid-Wide Execution	22-2
	Agents for Targeted, Parallel and Query-Based Execution	22-2
	Data Grid Aggregation.....	22-6
	Node-Based Execution.....	22-8
	Work Manager.....	22-10
	Oracle Coherence Work Manager: Feedback from a Major Financial Institution	22-10
23	Managing Map Operations with Triggers	
	A Map Trigger Example	23-2
24	Using Coherence Query Language	
	Understanding Coherence Query Language Syntax	24-1
	Query Syntax Basics.....	24-2
	Using Path-Expressions	24-2
	Using Bind Variables.....	24-3
	Using Key and Value Pseudo-Functions.....	24-3
	Using Aliases	24-3
	Using Quotes with Literal Arguments	24-3
	Retrieving Data.....	24-4
	Retrieving Data from the Cache.....	24-4
	Filtering Entries in a Result Set	24-4
	Managing the Cache Lifecycle.....	24-5
	Creating a Cache	24-5
	Writing a Serialized Representation of a Cache to a File	24-6
	Loading Cache Contents from a File.....	24-6
	Removing a Cache from the Cluster	24-7
	Working with Cache Data.....	24-7
	Aggregating Query Results	24-7
	Changing Existing Values.....	24-7

Inserting Entries in the Cache	24-8
Deleting Entries in the Cache	24-8
Working with Indexes	24-8
Creating an Index on the Cache	24-8
Removing an Index from the Cache	24-9
Issuing Multiple Query Statements	24-9
Processing Query Statements in Batch Mode	24-9
Using the CohQL Command-Line Tool	24-10
Starting the Command-line Tool	24-10
Using Command-Line Tool Arguments	24-11
A Command-Line Example	24-12
Building Filters in Java Programs	24-14
Additional Coherence Query Language Examples.....	24-15
Simple SELECT * FROM Statements that Highlight Filters.....	24-16
Complex Queries that Feature Projection, Aggregation, and Grouping	24-17
UPDATE Examples.....	24-17
Key and Value Pseudo-Function Examples	24-17

25 Performing Transactions

Overview of Transactions	25-1
Using Explicit Locking for Data Concurrency.....	25-2
Using Entry Processors for Data Concurrency.....	25-3
Using the Transaction Framework API	25-5
Defining Transactional Caches.....	25-6
Performing Cache Operations within a Transaction	25-8
Using the NamedCache API	25-8
Using the Connection API	25-9
Creating Transactional Connections	25-10
Using Transactional Connections	25-11
Using Auto-Commit Mode.....	25-11
Setting Isolation Levels	25-12
Using Eager Mode	25-13
Setting Transaction Timeout	25-13
Using the OptimisticNamedCache Interface	25-14
Configuring POF When Performing Transactions.....	25-14
Configuring Transactional Storage Capacity	25-15
Performing Transactions from Java Extend Clients.....	25-16
Create an Entry Processor for Transactions.....	25-16
Configure the Cluster-Side Transaction Caches.....	25-17
Configure the Client-Side Remote Cache	25-18
Use the Transactional Entry Processor from a Java Client	25-19
Viewing Transaction Management Information	25-19
CacheMBeans for Transactional Caches.....	25-19
TransactionManagerBean	25-20
Using the Coherence Resource Adapter	25-21
Performing Cache Operations within a Transaction	25-22
Creating a Coherence Connection	25-23

Getting a Named Cache	25-24
Demarcating Transaction Boundaries.....	25-24
Packaging the Application.....	25-25
Configure the Connection Factory Resource Reference	25-25
Configure the Resource Adapter Module Reference	25-26
Include the Required Libraries	25-26
Using the Coherence Cache Adapter for Transactions.....	25-27
26 Data Affinity	
Specifying Affinity	26-1
Specifying Data Affinity with a KeyAssociation.....	26-2
Specifying Data Affinity with a KeyAssociator.....	26-2
Example of Using Affinity.....	26-3
27 Priority Tasks	
Priority Tasks — Timeouts.....	27-1
Configuring Execution Timeouts.....	27-1
Command Line Options.....	27-3
Priority Task Execution — Custom Objects	27-3
APIs for Creating Priority Task Objects.....	27-4
Errors Thrown by Task Timeouts.....	27-5
28 Specifying a Custom Eviction Policy	
29 Constraints on Re-entrant Calls	
Re-entrancy, Services, and Service Threads	29-1
Parent-Child Object Relationships.....	29-1
Avoiding Deadlock	29-2
Re-entrancy and Listeners	29-2
30 Securing Coherence	
Using the Access Controller	30-1
Overview of the Access Controller.....	30-1
Proof of Identity	30-2
Proof of Trustworthiness	30-3
Enabling the Default Access Controller Implementation	30-4
Working in Applications with Installed Security Manager.....	30-6
Using SSL in Coherence.....	30-6
Overview of SSL.....	30-7
Configuring SSL for TCMP.....	30-8
Defining a SSL Socket Provider	30-8
Using the Pre-Defined SSL Socket Provider	30-10

Part V Deploying Coherence Applications

31 Deploying Coherence

Deploying Coherence with a Standalone Application	31-1
Deploying Coherence to an Application Server	31-1
Deploying Coherence as an Application Server Library.....	31-1
Deploying Coherence in a Java EE Module	31-2
Deploying Coherence Within an EAR	31-2
Deploying Coherence Within a WAR.....	31-3

32 Platform-Specific Deployment Considerations

Deploying to AIX	32-1
Socket Buffers sizes and JVMs	32-1
Multicast and IPv6	32-1
Unique Multicast Addresses and Ports	32-2
Deploying to Oracle JRockit JVMs.....	32-2
JRockit and the Native Posix Thread Library (NPTL).....	32-2
OutOfMemoryError.....	32-2
Deploying to Cisco Switches	32-2
Buffer Space and Packet Pauses	32-2
Multicast Connectivity on Large Networks.....	32-2
Multicast Outages	32-3
Deploying to Foundry Switches.....	32-5
Multicast Connectivity	32-5
Deploying to IBM BladeCenters	32-5
MAC Address Uniformity and Load Balancing.....	32-5
Deploying to IBM JVMs	32-6
UDP Socket Buffer Sizes.....	32-6
OutOfMemoryError.....	32-6
Heap Sizing	32-6
Deploying to Linux	32-7
Native POSIX Thread Library (NPTL)	32-7
TSC High Resolution Timesource	32-7
Deploying to OS X	32-8
Multicast and IPv6	32-8
Unique Multicast Addresses and Ports	32-8
Socket Buffer Sizing	32-8
Deploying to Solaris	32-8
Solaris 10 (x86 and SPARC)	32-8
Solaris 10 Networking	32-8
Deploying to Sun JVMs	32-9
Heap Sizes	32-9
AtomicLong	32-9
OutOfMemoryError.....	32-9
Deploying to Virtual Machines	32-10
Supported Deployment.....	32-10
Multicast Connectivity	32-10
Performance	32-10
Fault Tolerance	32-10

Deploying to Windows	32-10
Performance Tuning	32-10
Personal Firewalls	32-11
Disconnected Network Interface	32-11
Deploying to z OS	32-11
EBCDIC.....	32-11
Multicast	32-12

33 Production Checklist

Network.....	33-2
Hardware.....	33-4
Operating System.....	33-7
JVM	33-8
Java Security Manager.....	33-10
Application Instrumentation	33-10
Coherence Editions and Modes	33-10
Ensuring that RTC nodes don't use Coherence TCMP.....	33-11
Coherence Operational Configuration.....	33-12
Coherence Cache Configuration	33-12
Large Cluster Configuration	33-14
Death Detection	33-14
tangosol-license.xml Deprecated.....	33-16

Part VI Managing Coherence

34 How to Manage Coherence Using JMX

Configuring the Coherence Management Framework	34-1
Enabling JMX Management.....	34-2
Filtering MBeans	34-2
Configuring Management Refresh	34-3
Setting the Management Refresh Expiry	34-3
Setting the Management Refresh Policy	34-4
Setting the Management Refresh Timeout	34-4
Accessing Coherence MBeans	34-4
Viewing MBeans Using the JConsole Utility	34-5
Viewing MBeans Using the HTML Adapter Application.....	34-5
Using Coherence MBeanConnector to Access MBeans	34-6

35 JMX Reporter

Basic Configuration	35-1
Administration.....	35-1
Data Analysis	35-3
Advanced Configuration	35-4
Creating Custom Reports.....	35-4
Running Reporter in a Distributed Configuration.....	35-4

36 How to Create a Custom Report

Configuring a Report File	36-1
file-name Element	36-1
file-name Macros	36-1
file-name Macro Examples	36-2
Specifying Data Columns	36-2
How to Include an Attribute	36-2
How to Include Part of the Key	36-3
How to Include Information from Composite Attributes	36-3
How to Include Information from Multiple MBeans	36-3
Including Multiple MBean Information Example	36-3
How to Use Report Macros	36-4
How to Include Constant Values	36-5
Including Queries in a Report	36-5
Using Filters to Construct Reports	36-6
Using Functions to Construct a Report	36-9
Function Examples	36-9
Using Aggregates to Construct a Report	36-10
Aggregate Examples	36-11
Constructing Delta Functions	36-11
Delta Function Examples	36-12

37 How to Modify Report Batch

Report Batch Deployment Descriptor	37-1
Document Location	37-1
Document Root	37-1
System Properties	37-1
Document Format	37-2
Report Batch Element Index	37-3
frequency	37-4
location	37-5
init-param	37-6
init-params	37-7
output-directory	37-8
param-name	37-9
param-type	37-10
param-value	37-11
report-config	37-12
report-group	37-13
report-list	37-14

38 Analyzing Reporter Content

Network Health Report	38-1
Network Health Detail Report	38-2
Memory Status Report	38-3
Cache Size Report	38-4

Cache Usage Report	38-4
Service Report	38-6
Node List Report.....	38-6
Proxy Report	38-7
39 How to Run a Report on Demand	
How to Run ReportControl MBean at Node Startup	39-1
How to Configure the ReportControl MBean	39-2
40 Configuring Custom MBeans	
Creating an MBean XML Configuration File	40-1
Configuring Standard MBeans.....	40-1
Configuring MXBeans	40-1
Configuring JMX MBeans	40-2
Enabling a Custom MBean Configuration File	40-3
Setting a System Property	40-3
Adding a Custom MBean Configuration File to the Class Path.....	40-3
41 How to Manage Custom MBeans Within the Cluster	
Custom MBean Configuration.....	41-1
How to Add a Standard MBean to Coherence	41-1
How to Programmatically Add a Standard MBean to Coherence.....	41-1
Using Static MBean Names	41-2
How to Add the Results of a JMX Query to Coherence	41-2
Part VII Tuning Coherence	
42 Evaluating Performance and Scalability	
Measuring Latency and Throughput.....	42-1
Demonstrating Scalability	42-1
Tuning Your Environment	42-2
Measurements on a Large Cluster.....	42-2
43 Performing a Multicast Connectivity Test	
Running the Multicast Test Utility	43-1
Sample Commands.....	43-1
Multicast Test Example	43-2
Troubleshooting Multicast Communications	43-3
44 Performing a Datagram Test for Network Performance	
Running the Datagram Test Utility.....	44-1
Sample Commands for a Listener and a Publisher	44-2
Datagram Test Example	44-2
Reporting	44-3

Publisher Statistics	44-3
Listener Statistics	44-4
Throttling	44-5
Bidirectional Testing.....	44-5
Distributed Testing	44-5

45 Performance Tuning

Operating System Tuning	45-1
Socket Buffer Sizes	45-1
High Resolution timesource (Linux)	45-2
Datagram size (Microsoft Windows)	45-3
Thread Scheduling (Microsoft Windows)	45-3
Swapping.....	45-4
Network Tuning	45-4
Network Interface Settings	45-4
Bus Considerations	45-5
Network Infrastructure Settings	45-5
Ethernet Flow-Control.....	45-6
Path MTU	45-6
JVM Tuning	45-7
Basic Sizing Recommendation	45-7
Heap Size Considerations	45-7
General Guidelines	45-8
Moving the Cache Out of the Application Heap.....	45-10
GC Monitoring & Tuning.....	45-11
Coherence Network Tuning	45-11
Validation	45-11
Data Access Patterns	45-12
Data Access Distribution (hot spots).....	45-12
Cluster-node Affinity.....	45-12
Read/Write Ratio and Data Sizes.....	45-12
Interleaving Cache Reads and Writes	45-13

46 Using the Service Guardian

Overview	46-1
Configuring the Service Guardian	46-2
Setting the Guardian Timeout.....	46-2
Setting the Guardian Timeout for All Threads.....	46-3
Setting the Guardian Timeout Per Service Type	46-3
Setting the Guardian Timeout Per Service Instance	46-4
Using the Timeout Value From the PriorityTask API	46-4
Setting the Guardian Service Failure Policy.....	46-4
Setting the Guardian Failure Policy for All Threads	46-5
Setting the Guardian Failure Policy Per Service Type.....	46-5
Setting the Guardian Failure Policy Per Service Instance	46-6
Enabling a Custom Guardian Failure Policy	46-6
Issuing Manual Guardian Heartbeats	46-7

47 Using Quorum

Overview	47-1
Using the Cluster Quorum	47-1
Configuring the Cluster Quorum Policy	47-2
Using the Partitioned Cache Quorums	47-2
Configuring the Partitioned Cache Quorum Policy	47-3
Using the Proxy Quorum	47-4
Configuring the Proxy Quorum Policy	47-4
Enabling Custom Action Policies	47-5

48 Scaling Out Your Data Grid Aggregations Linearly

The Data	48-1
Configure a Partitioned Cache	48-3
Add an Index to the Price Property	48-4
Code to perform a Parallel Aggregation	48-4
The Testing Environment and Process	48-4
Performing a "Test Run"	48-5
This "Test Suite" (and Subsequent Results) Includes Data from Four "Test Runs":	48-5
JDK Version	48-5
The Results	48-5
Conclusion	48-7

A Operational Configuration Elements

Operational Configuration Deployment Descriptors	A-1
Document Location	A-1
Document Root	A-1
Document Format	A-1
Operational Override File (tangosol-coherence-override.xml)	A-2
Command Line Override	A-2
Element Index	A-3
access-controller	A-5
address-provider	A-6
authorized-hosts	A-7
cache-factory-builder-config	A-8
callback-handler	A-9
cluster-config	A-10
cluster-quorum-policy	A-12
coherence	A-13
configurable-cache-factory-config	A-14
filters	A-16
Compression Filter Parameters	A-17
flow-control	A-18
host-range	A-19
identity-asserter	A-20
identity-manager	A-21
identity-transformer	A-22

incoming-message-handler.....	A-23
init-param	A-24
init-params	A-25
instance	A-26
key-store	A-27
license-config	A-28
logging-config	A-29
management-config	A-32
mbean.....	A-34
mbeans	A-36
mbean-filter	A-37
member-identity	A-38
message-pool	A-40
multicast-listener	A-42
notification-queueing.....	A-45
outgoing-message-handler	A-46
outstanding-packets.....	A-47
packet-buffer	A-48
packet-bundling	A-49
packet-delivery	A-50
packet-pool.....	A-51
packet-publisher.....	A-52
packet-size	A-54
packet-speaker	A-55
pause-detection.....	A-56
provider	A-57
reporter	A-58
security-config	A-59
serializers.....	A-61
service-guardian.....	A-62
services.....	A-63
Initialization Parameter Settings.....	A-64
DistributedCache Service Parameters	A-65
ReplicatedCache Service Parameters	A-69
InvocationService Parameters	A-70
ProxyService Parameters	A-71
shutdown-listener	A-73
socket-address	A-74
socket-provider.....	A-75
socket-providers	A-77
ssl	A-78
tcp-ring-listener	A-79
traffic-jam	A-81
trust-manager	A-82
unicast-listener.....	A-83
volume-threshold	A-85
well-known-addresses.....	A-86

Element Attributes	A-88
--------------------------	------

B Cache Configuration Elements

Cache Configuration Deployment Descriptor.....	B-1
Element Reference.....	B-2
acceptor-config	B-5
address-provider	B-6
async-store-manager.....	B-7
authorized-hosts.....	B-9
backing-map-scheme	B-10
backup-storage	B-11
bdb-store-manager	B-13
bundle-config.....	B-15
cache-config.....	B-16
cache-mapping	B-17
cache-service-proxy	B-18
cachestore-scheme.....	B-19
caching-scheme-mapping	B-20
caching-schemes.....	B-21
class-scheme.....	B-23
custom-store-manager.....	B-24
defaults	B-25
disk-scheme.....	B-26
distributed-scheme.....	B-27
external-scheme	B-34
identity-manager	B-38
initiator-config	B-39
init-param	B-40
init-params	B-41
instance	B-42
invocation-scheme	B-43
invocation-service-proxy.....	B-46
jms-acceptor	B-47
jms-initiator	B-48
key-associator	B-49
key-partitioning.....	B-50
key-store	B-51
lh-file-manager	B-52
listener.....	B-53
local-address	B-54
local-scheme.....	B-55
near-scheme	B-58
nio-file-manager	B-61
nio-memory-manager.....	B-62
operation-bundling.....	B-64
optimistic-scheme	B-65
outgoing-message-handler	B-68

overflow-scheme	B-70
paged-external-scheme.....	B-73
partition-listener	B-76
partitioned	B-77
partitioned-quorum-policy-scheme	B-78
provider	B-79
proxy-config.....	B-80
proxy-scheme.....	B-81
proxy-quorum-policy-scheme.....	B-84
read-write-backing-map-scheme	B-85
remote-addresses.....	B-89
remote-cache-scheme.....	B-90
remote-invocation-scheme.....	B-91
replicated-scheme.....	B-92
serializer	B-95
socket-address	B-96
socket-provider.....	B-97
ssl	B-98
tcp-acceptor	B-99
tcp-initiator.....	B-103
transactional-scheme	B-105
trust-manager	B-110
version-persistent-scheme	B-111
version-transient-scheme	B-112
versioned-backing-map-scheme	B-113
versioned-near-scheme.....	B-117

C Command Line Overrides

Override Example.....	C-1
Preconfigured Override Values	C-2

D POF User Type Configuration Elements

POF User Type Deployment Descriptor	D-1
Document Location.....	D-1
Document Root.....	D-1
Document Format	D-1
Command Line Override	D-2
Element Index	D-3
allow-interfaces	D-4
allow-subclasses	D-5
class-name	D-6
default-serializer.....	D-7
include	D-8
init-param	D-9
init-params	D-10
param-type	D-11
param-value	D-12

pof-config	D-13
serializer	D-14
type-id	D-16
user-type	D-17
user-type-list	D-18

E Coherence MBeans Reference

Overview	E-1
MBean Index	E-2
CacheMBean	E-3
ClusterMBean	E-6
ClusterNodeMBean	E-8
ConnectionManagerMBean	E-13
ConnectionMBean	E-14
ManagementMBean	E-15
PointToPointMBean	E-16
ReporterMBean	E-18
ServiceMBean	E-20
StorageManagerMBean	E-24
TransactionManagerMBean	E-26

F The PIF-POF Binary Format

Stream Format	F-1
Integer Values	F-2
Type Identifiers	F-3
Binary Formats for Predefined Types	F-5
Int	F-5
Coercion of Integer Types	F-6
Decimal	F-7
Floating Point	F-7
Boolean	F-8
Octet	F-8
Octet String	F-9
Char	F-9
Char String	F-10
Date	F-10
Year-Month Interval	F-11
Time	F-11
Time Interval	F-11
Date-Time	F-11
Coercion of Date and Time Types	F-11
Day-Time Interval	F-11
Collections	F-11
Arrays	F-12
Sparse Arrays	F-13
Key-Value Maps (Dictionaries)	F-13

Identity	F-14
Reference	F-15
Binary Format for User Types	F-15
Versioning of User Types.....	F-16

G Log Message Glossary

TCMP Log Messages	G-1
Configuration Log Messages	G-6
Partitioned Cache Service Log Messages.....	G-7

List of Examples

1-1	Methods in the InvocationService API	1-7
4-1	The Sample HelloWorld Standalone Application.....	4-3
4-2	The Sample Hello World JSP.....	4-5
7-1	Single Server Mode Configuration.....	7-1
7-2	Command to Start Coherence in Single Server Mode.....	7-2
8-1	Determining Services Running in the Cluster	8-1
8-2	A Sample MemberListener Implementation.....	8-3
8-3	Using Event Type Information in a MemberEvent Object	8-3
9-1	Enabling a Filter for all Network Traffic	9-1
9-2	Declaring a Filter in the tangosol-coherence.xml File	9-4
9-3	Attaching the Filter to a Service.....	9-4
9-4	Adding the Filter to All Services.....	9-5
9-5	Configuration for a Custom Filter	9-5
9-6	Configuring a setConfig Call for a Filter	9-6
11-1	Sample One-to-One Cache Mapping	11-2
11-2	Sample Cache Name Pattern Mapping.....	11-2
11-3	Initialization Parameters in a Cache Mapping	11-3
11-4	Sample Distributed Cache Definition	11-5
11-5	Sample Replicated Cache Definition.....	11-5
11-6	Sample Optimistic Cache Definition.....	11-6
11-7	Sample Local Cache Definition	11-7
11-8	Sample Near Cache Definition.....	11-8
11-9	Using Cache Scheme References	11-9
11-10	Multiple Cache Schemes Using Scheme Inheritance	11-10
11-11	Setting Cache Properties	11-11
13-1	Cache Configuration Specifying a Refresh-Ahead Factor	13-5
13-2	A Cache Configuration with a Cachestore Module.....	13-7
13-3	Implementation of the CacheStore Interface.....	13-9
13-4	Main.java - Interacting with a Controllable CacheStore	13-14
13-5	CacheStoreAware.java interface	13-17
15-1	Configuration for a Local, In-memory Cache	15-2
15-2	Configuration for a NIO In-memory Cache.....	15-2
15-3	Configuration for a Size Limited, In-memory, Local Cache.....	15-2
15-4	Configuration for an In-memory Cache with Expiring Entries	15-2
15-5	Configuration to Define a Cache on Disk.....	15-3
15-6	Configuration for a Size Limited Cache on Disk.....	15-3
15-7	Configuration for Persistent cache on disk	15-3
15-8	Configuration for Persistent cache on disk with Berkeley DB.....	15-4
15-9	Configuration for In-memory Cache with Disk Based Overflow.....	15-4
15-10	Configuration for the Cache of a Database	15-4
15-11	Configuration for a Replicated Cache.....	15-5
15-12	Configuration for a Replicated Cache with Overflow.....	15-5
15-13	Configuration for a Partitioned Cache.....	15-6
15-14	Configuration for a Partitioned Cache with Overflow.....	15-6
15-15	Configuration for a Partitioned Cache of a Database	15-7
15-16	Configuration for a Partitioned Cache with a Serializer	15-7
15-17	Configuration for a Local Cache of a Partitioned Cache	15-7
17-1	Implementation of the PortableObject Interface	17-2
17-2	Implementation of the PofSerializer Interface	17-3
18-1	Pre-Loading a Cache.....	18-1
18-2	Pre-Loading a Cache Using ConcurrentMap.putAll	18-2
18-3	Using a Filter to Query a Cache	18-2
18-4	Processing Query Results in Batches	18-3
18-5	A Sample Bulk Loading Program.....	18-4

18-6	Terminal Output from the Bulk Loading Program.....	18-8
18-7	Retrieving Storage-Enabled Members of the Cache	18-10
18-8	Routine to Get a List of Files Assigned to a Cache Member.....	18-10
18-9	Class to Load Each Member of the Cache	18-10
19-1	Excerpt from the MapListener API	19-1
19-2	Excerpt from the MapEvent API	19-2
19-3	Methods on the ObservableMap API.....	19-4
19-4	Sample MapListener Implementation	19-4
19-5	Holding a Reference to a Listener	19-4
19-6	Removing a Listener.....	19-5
19-7	Inner Class that Prints Only Cache Insert Events	19-5
19-8	Routing All Events to a Single Method for Handling	19-5
19-9	Triggering an Event when a Specific Integer Key is Inserted or Updated	19-6
19-10	Adding a Listener with Filter for Deleted Events	19-6
19-11	Inserting, Updating, and Removing a Value from the Cache	19-7
19-12	Sample Output	19-7
19-13	Listening for Events from a Cache	19-8
19-14	Listening for Events on an Object.....	19-8
19-15	Using MapEventFilter to Filter on Various Events.....	19-8
19-16	Determining Synthetic Events	19-10
19-17	An AbstractMultiplexingBackingMapListener Implementation.....	19-11
19-18	Cache Configuration Specifying a Verbose Backing Map Listener.....	19-12
20-1	Equality Filter	20-2
20-2	Filter that Constructs a ReflectionExtractor.....	20-2
20-3	Selecting Cache Entries that Satisfy a Filter	20-2
20-4	Selecting and Sorting Cache Entries that Satisfy a Filter.....	20-2
20-5	Querying the Cache with a Filter.....	20-2
20-6	Sample Code to Create an Index.....	20-3
20-7	Using a keySet Query Format	20-6
20-8	Using a Limit Filter.....	20-6
20-9	Querying on Multi-Value Attributes	20-8
20-10	Chaining Invocation Methods.....	20-8
21-1	A Query for a Continuous Query Cache.....	21-2
21-2	Getting Data for the Continuous Query Cache	21-3
21-3	Constructing the Continuous Query Cache.....	21-3
21-4	A Constructor that Allows the CacheValues Property	21-3
21-5	Setting the CacheValues Property	21-3
21-6	Adding a Listener to a Continuous Query Cache.....	21-4
21-7	Processing Continuous Query Cache Entries and Adding a Listener	21-4
21-8	Adding a Listener Before Processing Continuous Query Cache Entries.....	21-4
21-9	Providing a Listener When Constructing the Continuous Query Cache	21-4
21-10	Making the Continuous Query Cache Read-Only	21-5
22-1	Querying Across a Data Grid.....	22-2
22-2	Methods in the EntryProcessor Interface	22-3
22-3	InvocableMap.Entry API	22-4
22-4	Aggregation in the InvocableMap API.....	22-6
22-5	EntryAggregator API	22-7
22-6	ParallelAwareAggregator API for running Aggregation in Parallel	22-7
22-7	Simple Agent to Request Garbage Collection.....	22-8
22-8	Agent to Support a Grid-Wide Request and Response Model	22-8
22-9	Printing the Results from a Grid-Wide Request or Response	22-9
22-10	Stateful Agent Operations	22-9
22-11	Using a Work Manager	22-10
23-1	Creating a MapTriggerListener in the coherence-cache-config.xml File.....	23-2
23-2	A MapTriggerListener Registering a MapTrigger with a Named Cache.....	23-2

23-3	A MapTrigger Class.....	23-2
23-4	Calling a MapTrigger and Passing it to a Named Cache.....	23-3
24-1	A Command-Line Query Exercise	24-12
25-1	Applying Locking Operations on a Cache.....	25-2
25-2	Concurrency Control without Using EntryProcessors	25-4
25-3	Concurrency Control Using EntryProcessors.....	25-4
25-4	Example Transactional Cache Definition	25-7
25-5	Performing an Auto-Commit Transaction.....	25-9
25-6	Performing a Non Auto-Commit Transaction.....	25-9
25-7	Transaction Across Multiple Caches.....	25-10
25-8	Entry Processor for Extend Client Transaction	25-16
25-9	Performing a Transaction When Using CMT	25-22
25-10	Performing a User-Controlled Transaction.....	25-22
25-11	Using the CacheAdapter Class When Using coherence-transaction.rar	25-27
25-12	Using the CacheAdapter Class When Using coherence-tx.rar.....	25-28
26-1	Creating a Key Association	26-2
26-2	A Custom KeyAssociator.....	26-2
26-3	Configuring a Key Associator	26-3
26-4	Using Affinity for a More Efficient Query	26-3
27-1	Sample Task Time and Task Hung Configuration	27-2
27-2	Sample Client Request Timeout Configuration	27-2
27-3	Exception Thrown by a TaskTimeout.....	27-5
28-1	Implementing a Custom Eviction Policy.....	28-1
28-2	Custom Eviction Policy in a coherence-cache-config.xml File	28-3
30-1	Sample SSL Configuration for TCMP Communication	30-9
32-1	Log for a Multicast Outage	32-3
33-1	Verifying Hardware Configuration	33-11
33-2	Sample Coherence License Configuration	33-11
35-1	System Properties for Reporter on the "Management" Node.....	35-1
35-2	System Properties for Reporter on the "Managed" Node	35-1
35-3	System Properties for Reporter in Distributed Mode on the "Managing" Node.....	35-4
35-4	System Properties for Reporter in Distributed Mode on the "Managed" Node	35-4
36-1	Including an Attribute Obtained from a Query Pattern	36-2
36-2	Including Part of an ObjectName Key in a Report	36-3
36-3	Including Information from a Composite Attribute in a Report	36-3
36-4	Including Information from Multiple MBeans in a Report	36-4
36-5	Including Execution Time in a Report.....	36-4
36-6	Including the Report Batch/Count in a Report.....	36-4
36-7	Including the Execution Node	36-5
36-8	Including a Constant Numeric Value in a Report.....	36-5
36-9	Including a Constant String in a Report	36-5
36-10	Including a List of the Cluster's NodeIDs and RoleNames in a Report	36-5
36-11	Using an Equals Filter for a Report	36-6
36-12	Defining a "Greater Than" Filter for a Report.....	36-7
36-13	Defining a "Less Than" Filter for a Report.....	36-7
36-14	Defining an "And" Filter for a Report	36-7
36-15	Defining an "Or" Filter for a Report	36-8
36-16	Defining a "Not Equals" Filter for a Report.....	36-8
36-17	Adding Column Values and Including Results in a Different Column	36-9
36-18	Subtracting Column Values and Including Results in a Different Column.....	36-9
36-19	Multiplying Column Values and Including Results in a Different Column.....	36-10
36-20	Dividing Column Values and Including Results in a Different Column	36-10
36-21	Adding the Values in a Column	36-11
36-22	Calculating the Average of Values in a Column	36-11
36-23	Finding the Maximum Value in a Column	36-11

36-24	Finding the Minimum Value in a Column.....	36-11
36-25	Delta Calculation for an Attribute	36-12
36-26	Delta Calculation for an Attribute with an Alternate Delta Key	36-12
37-1	Format of a Report Batch Configuration File (report-group.xml)	37-2
39-1	tangosol.coherence.management.report.autostart System Property	39-2
39-2	tangosol.coherence.management.report.group System Property	39-2
40-1	Using an MBean to Create a Query Node.....	40-1
40-2	Getting an MBean for the Memory System of a Java Virtual Machine.....	40-2
40-3	Executing a JMX Query and Creating an MBean on the MBean Server	40-2
40-4	System Property to Load an MBean.....	40-3
41-1	Adding a Standard MBean to Coherence Programatically	41-1
43-1	Command to Determine a Multicast Address.....	43-2
43-2	Sequential Multicast Packets Sent by the Multicast Test Utility	43-2
43-3	Sample Multicast Test Results from Server A	43-2
43-4	Sample Multicast Test Results on Server B	43-3
43-5	Command to Set Machine Routing Table	43-4
44-1	Command to Start a Listener	44-2
44-2	Output from Starting a Listener.....	44-2
44-3	Command to Start a Publisher.....	44-3
44-4	Datagram Test—Starting a Listener and a Publisher on a Server	44-3
44-5	Sample Publisher Report	44-3
44-6	Sample Lifetime Statistics	44-4
44-7	Running Datagram Test in Bi-Directional Mode	44-5
45-1	Message Indicating OS Failed to Allocate the Full Buffer Size	45-1
45-2	Log Message from a Linux Timesource.....	45-2
45-3	Message Indicating a Communication Delay	45-5
45-4	Disabling Partition Storage.....	45-10
45-5	Message Indicating Target Cluster Node is in Garbage Collection Mode	45-11
48-1	Trade Object Defining Three Properties.....	48-1
48-2	Mapping a cache-mapping to a caching-scheme with Unlimited Capacity	48-3
48-3	Adding an Index to the Price Property	48-4
48-4	Perform a Parallel Aggregation Across all JVMs in the Grid.....	48-4
A-1	Operational Configuration Deployment Descriptor DOCTYPE Declaration	A-1
A-2	Sample init-param Configuration.....	A-65
A-3	Another Sample init-param Configuration.....	A-65
A-4	Configuration for Two Well-Known-Addresses	A-86
D-1	Format of a POF User Type Configuration File (pof-config.xml).....	D-1
F-1	Writing a 32-bit Integer Value to an Octet Stream	F-2
F-2	Reading a 32-bit Integer Value from an Octet Stream.....	F-2
F-3	Writing a Character Value to an Octet Stream	F-9
F-4	Reading a Character Value from an Octet Stream	F-9

List of Figures

10-1	Get Operations in a Partitioned Cache Environment	10-2
10-2	Put Operations in a Partitioned Cache Environment	10-3
10-3	Failover in a Partitioned Cache Environment	10-4
10-4	Local Storage in a Partitioned Cache Environment	10-5
10-5	Get Operation in a Replicated Cache Environment	10-6
10-6	Put Operation in a Replicated Cache Environment	10-7
10-7	Put Operations in a Near Cache Environment	10-8
10-8	Get Operations in a Near Cache Environment	10-9
12-1	Backing Map Storage	12-2
12-2	Conventional Backing Map Implementation	12-6
12-3	Partitioned Backing Map Implementation	12-6
13-1	Read-Through Caching	13-2
13-2	Write-Through Caching	13-2
13-3	Write-Behind Caching	13-4
34-1	JConsole Utility to Access Coherence MBeans	34-5
34-2	Viewing the HttpAdapter Web Application in a Browser	34-6
35-1	Reporter Attributes in JConsole	35-2
35-2	Reporter Operations in JConsole	35-3
39-1	Reporter Operations in JConsole	39-1
40-1	MBean Query Displayed in the JConsole	40-3
41-1	JMX Query Run in JConsole	41-3
42-1	Coherence Throughput versus Number of Machines	42-2
42-2	Coherence Latency versus Number of Machines	42-3
48-1	Average Aggregation Time	48-6
48-2	Aggregation Scale-Out	48-7

List of Tables

3-1	Preconfigured System Properties	3-18
9-1	Symmetric Encryption Filter Parameters	9-2
9-2	PKCS Encryption Filter Parameters	9-3
10-1	Summary of Cache Types and Characteristics	10-10
11-1	Near Cache Invalidation Strategies	11-9
11-2	Parameter Macros for Cache Configuration	11-12
24-1	Coherence Query Language Statements	24-2
24-2	Coherence Query Language Command-Line Tool Arguments	24-11
25-1	Coherence Transaction Options	25-2
25-2	Transactional Cache Supported Attributes	25-19
25-3	TransactionManagerMBean Attributes	25-20
27-1	Execution Timeout Elements	27-2
27-2	Command Line Options for Setting Service Type	27-3
27-3	Methods to Support Task Timeout	27-4
33-1	Valid tangosol.coherence.edition Values	33-11
34-1	Refresh Policies	34-4
34-2	Optional Properties that can be used for JMX RMI Configuration	34-7
34-3	Optional Properties that can be used for Http Configuration	34-7
35-1	File Names Generated by Reporter	35-3
36-1	Elements to Configure an Output File	36-1
36-2	Macros that can be Used with the file-name Element	36-2
36-3	Reporter Aggregate Types	36-10
37-1	System Properties for Controlling Report Batch	37-1
37-2	Report Batch Elements	37-3
38-1	Contents of the Network Health Report	38-1
38-2	Contents of the Network Health Detail Report	38-2
38-3	Contents of the Memory Status Report	38-3
38-4	Contents of the Cache Size Report	38-4
38-5	Contents of the Cache Usage Report	38-4
38-6	Contents of the Service Report	38-6
38-7	Contents of the Node List Report	38-7
38-8	Contents of the Proxy Report	38-7
43-1	Command Line Options for the Multicast Test Utility	43-1
44-1	Command Line Options for the Datagram Test Utility	44-1
44-2	Listener Statistics	44-4
A-1	Non-Terminal Operational Configuration Elements	A-3
A-2	access-controller Subelements	A-5
A-3	address-provider Subelements	A-6
A-4	authorized-hosts Subelements	A-7
A-5	cache-factory-builder-config Subelements	A-8
A-6	callback-handler Subelement	A-9
A-7	cluster-config Subelements	A-10
A-8	cluster-quorum-policy-scheme Subelements	A-12
A-9	coherence Subelements	A-13
A-10	configurable-cache-factory-config Subelements	A-15
A-11	filters Subelements	A-16
A-12	Compression Filter Parameters	A-17
A-13	flow-control Subelements	A-18
A-14	host-range Subelements	A-19
A-15	identity-asserter Subelements	A-20
A-16	identity-manager Subelements	A-21
A-17	identity-transformer Subelements	A-22
A-18	incoming-message-handler Subelements	A-23

A-19	init-param Subelement.....	A-24
A-20	init-params Subelement	A-25
A-21	instance Subelements	A-26
A-22	key-store Subelements.....	A-27
A-23	license-config Subelements.....	A-28
A-24	logging-config Subelements	A-29
A-25	management-config Subelements.....	A-32
A-26	Subelements of mbean	A-34
A-27	Subelement of mbeans	A-36
A-28	mbean-filter Subelements	A-37
A-29	member-identity Subelements	A-38
A-30	message-pool Subelements.....	A-40
A-31	multicast-listener Subelements	A-43
A-32	notification-queuing Subelements.....	A-45
A-33	outgoing-message-handler Subelement	A-46
A-34	outstanding-packets Subelements	A-47
A-35	packet-buffer Subelements	A-48
A-36	packet-bundling Subelements.....	A-49
A-37	packet-delivery Subelements	A-50
A-38	packet-pool Subelements	A-51
A-39	packet-publisher Subelements	A-52
A-40	packet-size Subelement.....	A-54
A-41	packet-speaker Subelements	A-55
A-42	pause-detection Subelements.....	A-56
A-43	provider Subelements	A-57
A-44	reporter Subelements.....	A-58
A-45	security-config Subelements.....	A-59
A-46	serializer Subelements.....	A-61
A-47	service-guardian Subelements	A-62
A-48	services Subelements	A-64
A-49	Sample Table Entry	A-65
A-50	DistributedCache Service Parameters.....	A-66
A-51	ReplicatedCache Service Parameters	A-70
A-52	InvocationService Parameters.....	A-71
A-53	ProxyService Parameters	A-72
A-54	shutdown-listener Subelements.....	A-73
A-55	socket-address Subelements.....	A-74
A-56	socket-provider Subelements	A-76
A-57	socket-providers Subelements	A-77
A-58	ssl Subelements	A-78
A-59	tcp-ring-listener Subelements	A-79
A-60	traffic-jam Subelements.....	A-81
A-61	trust-manager Subelements.....	A-82
A-62	unicast-listener Subelements.....	A-83
A-63	packet-speaker Subelements	A-85
A-64	well-known-addresses Subelements	A-87
A-65	Elements that can use id or xml-override, or Both.....	A-88
A-66	id and xml-override Attribute Descriptions	A-88
B-1	Cache Configuration Elements	B-2
B-2	acceptor-config Subelements.....	B-5
B-3	address-provider Subelements	B-6
B-4	async-store-manager Subelements	B-7
B-5	authorized-hosts Subelements	B-9
B-6	backing-map-scheme Subelements	B-10
B-7	backup-storage Subelements.....	B-11

B-8	bdb-store-manager Subelements	B-13
B-9	bundle-config Subelements	B-15
B-10	cache-config Subelements	B-16
B-11	cache-mapping Subelements	B-17
B-12	cache-service-proxy Subelements	B-18
B-13	cachestore-scheme Subelements	B-19
B-14	caching-scheme-mapping Subelement	B-20
B-15	caching-schemes Subelements	B-21
B-16	class-scheme Subelements	B-23
B-17	custom-store-manager Subelements	B-24
B-18	defaults Subelements	B-25
B-19	distributed-scheme Subelements	B-28
B-20	external-scheme Subelements	B-35
B-21	identity-manager Subelements	B-38
B-22	initiator-config Subelements	B-39
B-23	init-param Subelements	B-40
B-24	init-params Subelements	B-41
B-25	instance Subelements	B-42
B-26	invocation-scheme Subelements	B-43
B-27	invocation-service-proxy Subelement	B-46
B-28	jms-acceptor Subelements	B-47
B-29	jms-initiator Subelements	B-48
B-30	key-associator Subelements	B-49
B-31	key-partitioning Subelements	B-50
B-32	key-store Subelements	B-51
B-33	lh-file-manager Subelements	B-52
B-34	listener Subelement	B-53
B-35	local-address Subelements	B-54
B-36	local-scheme Subelements	B-55
B-37	near-scheme Subelements	B-58
B-38	nio-file-manager Subelements	B-61
B-39	nio-memory-manager Subelements	B-62
B-40	operation-bundling Subelement	B-64
B-41	optimistic-scheme Subelements	B-65
B-42	outgoing-message-handler Subelements	B-69
B-43	overflow-scheme Subelements	B-70
B-44	paged-external-scheme Subelements	B-74
B-45	partition-listener Subelements	B-76
B-46	partitioned-quorum-policy-scheme Subelements	B-78
B-47	provider Subelements	B-79
B-48	proxy-config Subelements	B-80
B-49	proxy-scheme Subelements	B-81
B-50	proxy-quorum-policy-scheme Subelements	B-84
B-51	read-write-backing-map-scheme Subelements	B-85
B-52	remote-addresses Subelements	B-89
B-53	remote-cache-scheme Subelements	B-90
B-54	remote-invocation-scheme Subelements	B-91
B-55	replicated-scheme Subelements	B-92
B-56	serializer Subelements	B-95
B-57	socket-address Subelements	B-96
B-58	socket-provider Subelements	B-97
B-59	ssl Subelements	B-98
B-60	tcp-acceptor Subelements	B-99
B-61	tcp-initiator Subelements	B-103
B-62	transactional-scheme Subelements	B-105

B-63	trust-manager Subelements.....	B-110
B-64	persistent-scheme Subelements	B-111
B-65	transient-scheme Subelements.....	B-112
B-66	versioned-backing-map-scheme Subelement	B-114
B-67	versioned-near-scheme Subelements.....	B-117
C-1	Preconfigured System Property Override Values.....	C-2
D-1	POF Configuration Elements	D-3
D-2	default-serializer Subelements.....	D-7
D-3	init-param Subelements	D-9
D-4	init-params Subelements	D-10
D-5	pof-config Subelements.....	D-13
D-6	serializer Subelements.....	D-15
D-7	user-type Subelements	D-17
D-8	user-type-list Subelements.....	D-18
E-1	Coherence MBeans.....	E-2
E-2	CacheMBean Attributes	E-3
E-3	ClusterMBean Attributes	E-6
E-4	ClusterMBean Operations	E-7
E-5	ClusterNodeMBean Attributes	E-8
E-6	ClusterNodeMBean Operations.....	E-12
E-7	ConnectionManagerMBean Attributes.....	E-13
E-8	ConnectionMBean Attributes.....	E-14
E-9	ConnectionMBean Operations	E-14
E-10	ManagementMBean Attributes.....	E-15
E-11	PointToPointMBean Attributes.....	E-16
E-12	PointToPointMBean Operations.....	E-17
E-13	ReporterMBean Attributes	E-18
E-14	ReporterMBean Operations.....	E-19
E-15	ServiceMBean Attributes	E-20
E-16	ServiceMBean Operations.....	E-23
E-17	StorageManagerMBean Attributes.....	E-24
E-18	TransactionManagerMBean Attributes	E-26
F-1	Regions in the First Octet of a Packed Integer.....	F-2
F-2	Regions in the Trailing Octet of a Packed Integer.....	F-2
F-3	Binary Formats for Integer Values Without a Type Identifier	F-3
F-4	Predefined Type Identifiers.....	F-3
F-5	Type Identifiers that Combine a Type and a Value	F-4
F-6	Type Identifiers that Combine an int Data Type with a Value	F-6
F-7	Type IDs of Integer Types that can be Coerced into Other Types.....	F-6
F-8	Type Identifiers that can Indicate Decimal Values	F-7
F-9	Type Identifiers that can Indicate IEEE 754 Special Values.....	F-8
F-10	Type Identifiers that can Indicate Boolean Values.....	F-8
F-11	Integer Values that may be Used for Octet Values	F-8
F-12	Values for Char String Formats	F-10
F-13	Collection and Uniform Collection Formats for Various Values.....	F-12
F-14	Array and Uniform Array Formats for Various Values.....	F-12
F-15	Sparse Array and Uniform Sparse Array Formats for Various Values	F-13
F-16	Binary Formats for Key/Value Pairs	F-14
F-17	Binary Formats for Key/Value Pairs where Keys are of Uniform Type	F-14
F-18	Binary Formats for Key/Value Pairs where Keys and Values are of Uniform Type....	F-14
F-19	Binary Formats for "By Reference" Semantics	F-15

Preface

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

Audience

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

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

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

Documentation Accessibility

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

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/support/contact.html> or visit <http://www.oracle.com/accessibility/support.html> if you are hearing impaired.

Related Documents

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

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

Conventions

The following text conventions are used in this document:

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

Part I

Getting Started

Part I contains the following chapters:

- [Chapter 1, "Introduction"](#)
- [Chapter 2, "Installing Oracle Coherence for Java"](#)
- [Chapter 3, "Understanding Configuration"](#)
- [Chapter 4, "Building Your First Coherence Application"](#)

Introduction

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

The following sections are included in this chapter:

- [Basic Concepts](#)
- [Read/Write Caching](#)
- [Querying the Cache](#)
- [Transactions](#)
- [HTTP Session Management](#)
- [Invocation Service](#)
- [Events](#)
- [Object-Relational Mapping Integration](#)
- [C++/.NET Integration](#)
- [Management and Monitoring](#)

Basic Concepts

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

Clustered Data Management

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

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

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

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

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

Caching Strategies

Coherence provides several cache implementations:

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

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

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

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

See [Part III, "Using Caches"](#) for detailed information on configuring and using caches.

Data Storage Options

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

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

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

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

See [Chapter 12, "Implementing Storage and Backing Maps,"](#) for detailed information.

Serialization Options

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

- `com.tangosol.io.pof.PofSerializer` – The Portable Object Format (also referred to as POF) is a language agnostic binary format. POF was designed to be incredibly efficient in both space and time and is the recommended serialization option in Coherence. See [Chapter 17, "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 physical machine). A cluster is defined by the combination of multicast address and port. A TTL (network packet time-to-live; that is, the number of network hops) setting can be used to restrict the cluster to a single machine, or the machines attached to a single switch.

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

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

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

See [Chapter 5, "Cluster Services Overview,"](#) for more information on the cluster service as well the other cluster-based service provided by Coherence.

Read/Write Caching

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

NamedCache

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

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

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

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

- `java.util.Map`—basic Map methods such as `get()`, `put()`, `remove()`.
- `com.tangosol.util.ObservableMap`—methods for listening to cache events. (See [Chapter 19, "Using Cache Events"](#).)
- `com.tangosol.net.cache.CacheMap`—methods for getting a collection of keys (as a `Map`) that are in the cache and for putting objects in the cache. Also supports adding an expiry value when putting an entry in a cache.
- `com.tangosol.util.QueryMap`—methods for querying the cache. (See "Query the Cache" in the *Oracle Coherence Developer's Guide*)
- `com.tangosol.util.ConcurrentMap`—methods for concurrent access such as `lock()` and `unlock()`.
- `com.tangosol.util.InvocableMap`—methods for sever-side processing of cache data.

Requirements for Cached Objects

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

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

NamedCache Usage Patterns

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

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

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

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

See [Chapter 13, "Read-Through, Write-Through, Write-Behind, and Refresh-Ahead Caching"](#) for more information on inline caching.

Write-back options:

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

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

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

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

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

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

Querying the Cache

Coherence provides the ability to query cached data. With partitioned caches, the queries are indexed and parallel. This means that adding servers to a partitioned cache not only increases throughput (total queries per second) but also reduces latency, with queries taking less user time. To query against a `NamedCache`, all objects should implement a common interface (or base class). Any field of an object can be queried; indexes are optional, and used to increase performance. With a replicated cache, queries are performed locally, and do not use indexes. See [Chapter 20, "Querying Data In a Cache,"](#) for detailed information.

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

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

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

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

Transactions

Coherence provides various transaction options. The options include: basic data concurrency using the `ConcurrentMap` interface and `EntryProcessor` API, atomic transactions using the Transaction Framework API, and atomic transactions with full XA support using the Coherence resource adapter. See [Chapter 25, "Performing Transactions"](#) for detailed instructions.

HTTP Session Management

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

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

Invocation Service

The Coherence invocation service can deploy computational agents to various nodes within the cluster. These agents can be either execute-style (deploy and asynchronously listen) or query-style (deploy and synchronously listen). See [Chapter 22, "Processing Data In a Cache,"](#) for more information on using the invocation service.

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

Example 1–1 Methods in the InvocationService API

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

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

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

Events

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

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

See [Chapter 19, "Using Cache Events,"](#) for more detailed information on using events.

Object-Relational Mapping Integration

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

C++/.NET Integration

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

embedded JVMs or language bridges). The wire protocol supports event feeds and coherent in-process caching for all client platforms. See *Oracle Coherence Client Guide* for complete instructions on using Coherence*Extend to support remote C++ and .NET clients.

Management and Monitoring

Coherence offers management and monitoring facilities by using Java Management Extensions (JMX). A detailed set of statistics and commands is maintained in the API documentation for `com.tangosol.net.management.Registry`. See [Part VI](#), "[Managing Coherence](#)" for detailed information.

Installing Oracle Coherence for Java

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

The following sections are included in this chapter:

- [System Requirements](#)
- [Extracting the Distribution](#)
- [Setting Environment Variables](#)
- [Running Coherence for the First Time](#)

System Requirements

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

- 65 MB disk space for installation
- 1 GB of RAM (assuming a maximum Java heap size of 512MB) – This amount of RAM can ideally support a maximum cache size of 150MB on a single node that is configured to store a backup of all data (150MB x 2) and leaves more than a 1/3 of the heap available for scratch and JVM tasks. Refer to "[JVM Tuning](#)" on page 45-7 for recommendations on calculating cache size.
- 1.5.x JVM or later
- Windows or UNIX-based system that supports the required Java Version
- Network adapter

Extracting the Distribution

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

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

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

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

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

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

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

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

Setting Environment Variables

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

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

Running Coherence for the First Time

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

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

Create a Basic Cluster

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

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

Note: The Coherence default behavior is to use multicast to find cluster members. Coherence can be configured to use unicast if a network does not allow the use of multicast. See [Part II, "Using Data Clusters"](#) for information on configuring clusters to use unicast.

To create a basic cluster:

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

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

Replace `cluster_name` and `port` with values that are unique for this cluster. For example, use your name for the cluster name and the last four digits of your phone number for the port.

3. Save and close the `cache-server` script.
4. Repeat steps 1 to 3 for the `COHERENCE_HOME/bin/coherence` script.
5. Run the `cache-server` script. The cache server starts and output is emitted that provides information about this cluster member.
6. Run 2 instances of the `coherence` script. As each instance is started, output is emitted that provides information about the respective cluster members. Each instance returns a command prompt for the command-line tool.

Create a Cache

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

To create a cache:

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

```
cache Test
```

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

```
put key1 Hello
```

The command returns and displays `null`. The `put` command always returns the previous value for a given key. The `null` value is returned because this is the first value entered for this key.

3. Switch to the other cache factory instance and from the command prompt create the `Test` cache using the `cache` command:

`cache Test`

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

`get key1`

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

Understanding Configuration

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

The following sections are included in this chapter:

- [Overview of the Default Configuration Files](#)
- [Specifying an Operational Configuration File](#)
- [Specifying a Cache Configuration File](#)
- [Specifying a POF Configuration File](#)
- [Specifying Management Configuration Files](#)
- [Understanding the XML Override Feature](#)
- [Changing Configuration Using System Properties](#)

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 Document Type Definition (DTD) files 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 DTD for this file is the `coherence.dtd` file. For a complete reference of the elements in this file, see [Appendix A, "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.

- `tangosol-coherence-override-eval.xml` – This file overrides operational settings in the `tangosol-coherence.xml` file when Coherence is started in evaluation mode. The settings in this file are suitable for evaluating Coherence.
- `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.
- `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 DTD for this file is the `cache-config.dtd` file. For a complete reference of the elements in this file, see [Appendix B, "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 DTD for this file is the `pof-config.dtd` file. For a complete reference of the elements in this file, see [Appendix D, "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 the Coherence JAR. The files include a report group configuration file (`report-group.xml`, by default), which refers to any number of report definition files. Each report definition file results in the creation of a report file that displays management information for a particular metric. The DTD for these files are the `report-group.dtd` and `report-config.dtd`, respectively. For a complete reference of the elements in this file, see the ["management-config"](#) element in [Appendix A, "Operational Configuration Elements."](#)
- License configuration files – A set of files that are used to license specific Coherence server and client editions.

Specifying an Operational Configuration File

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

The default operational deployment descriptor that is shipped with Coherence is located in the root of the `coherence.jar` library. This file can be changed as required; however, overriding this file is recommended when configuring the operational run time. See ["Understanding the XML Override Feature"](#) on page 3-14 for detailed information about the XML override feature.

The following topics are included in this section:

- [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](#)

Refer to [Part II, "Using Data Clusters"](#) for detailed instructions on configuring the operational run time.

Using the Default Operational Override File

Elements in the default `tangosol-coherence.xml` file are overridden by placing an operational override file named `tangosol-coherence-override.xml` in the classpath at run time. The structure of the override file is the same as the operational deployment descriptor 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'?>

<!DOCTYPE coherence SYSTEM "coherence.dtd">

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

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

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

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

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

Specifying an Operational Override File

The `tangosol.coherence.override` system property specifies an operational override file to be used instead of the default `tangosol-coherence-override.xml` file. The structure of the specified file is the same as the operational deployment descriptor except that all elements are optional. Any missing elements are loaded from the `tangosol-coherence.xml` file.

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

To specify an operational override file:

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

The following example configures the multicast port number:

```
<coherence>
  <cluster-config>
    <multicast-listener>
      <port system-property="tangosol.coherence.clusterport">3059</port>
    </multicast-listener>
  </cluster-config>
</coherence>
```

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

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

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

Defining Override Files for Specific Operational Elements

Override files can be created to override the contents of specific operational elements. The override files follow the same structure as the operational deployment descriptor except that their root element must match the element that is to be overridden. See ["Defining Custom Override Files"](#) on page 3-15 for detailed information on defining override files for specific operational elements.

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

To define override files for specific operational elements:

1. Create a `tangosol-coherence-override.xml` file as described in ["Using the Default Operational Override File"](#) on page 3-3.
2. Add an `xml-override` attribute to an element that is to be overridden. The value of the `xml-override` attribute is the name of an override file.

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

```
<coherence>
  <cluster-config xml-override="/cluster-config.xml">
    ...
  </cluster-config>
</coherence>
```

```

    </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 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.

```

<cluster-config>
  <multicast-listener>
    <join-timeout-milliseconds>4000</join-timeout-milliseconds>
  </multicast-listener>
</cluster-config>

```

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

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

```

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

```

Viewing Which Operational Override Files are Loaded

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

The following example output demonstrates typical messages that are emitted:

```

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

```

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

Specifying a Cache Configuration File

The `coherence-cache-config.xml` cache configuration deployment descriptor file is used to specify the various types of caches that can be used within a cluster. At

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

Note:

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

The following topics are included in this section:

- [Using a Default Cache Configuration File](#)
- [Overriding the Default Cache Configuration File](#)
- [Using the Cache Configuration File System Property](#)
- [Viewing Which Cache Configuration File is Loaded](#)

Refer to [Part III, "Using Caches"](#) for detailed instructions on configuring caches.

Using a Default Cache Configuration File

Coherence is configured out-of-box to use the first `coherence-cache-config.xml` file that is found on the classpath. To use a `coherence-cache-config.xml` file, the file must be located on the classpath and must precede the `coherence.jar` library; otherwise, the sample `coherence-cache-config.xml` file that is located in the `coherence.jar` will be 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 should be used as a starting point for creating solution-specific cache configurations.
2. Make sure that the location where the `coherence-cache-config.xml` file is saved is in the classpath at runtime and that the location precedes the `coherence.jar` file in the classpath.

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

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

Overriding the Default Cache Configuration File

The default name and location of the cache configuration deployment descriptor is specified in the operational deployment descriptor within the `<configurable-cache-factory-config>` element. This element can be overridden to specify a different name 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 as described in ["Using the Default Operational Override File"](#) on page 3-3.
3. Edit the operational override file and enter a `<configurable-cache-factory-config>` node that specifies the name of the cache configuration file created in step 1. If the cache configuration file is not located in the classpath, enter the full (or relative) path to the file as well. The element also supports the use of a URL when specifying the location of a cache configuration file.

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

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

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

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

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

Using the Cache Configuration File System Property

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

To specify a custom cache configuration file, enter the name of the file as a value of the `tangosol.coherence.cacheconfig` system property. This is typically done as a

-D Java option when starting a Coherence node. If the file is not located in the classpath, enter the full (or relative) path to the file as well as the name. The system property also supports the use of a URL when specifying the location of a cache configuration file.

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

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

Viewing Which Cache Configuration File is Loaded

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

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

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

Specifying a POF Configuration File

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

Note:

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

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

The following topics are included in this section:

- [Using the POF Configuration File System Property](#)
- [Combining Multiple POF Configuration Files](#)

- [Viewing Which POF Configuration Files are Loaded](#)

Refer to [Chapter 17, "Using Portable Object Format"](#) for detailed instructions on configuring POF user types.

Using the POF Configuration File System Property

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

To specify a custom POF configuration file:

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

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

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

<pof-config>
  <user-type-list>
    <include>coherence-pof-config.xml</include>
  </user-type-list>
</pof-config>
```

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

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

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

Combining Multiple POF Configuration Files

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

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

To combine multiple POF configuration files:

1. Open an existing POF configuration file that is being loaded at startup.

2. Add an `<include>` element whose value is the name of a POF configuration file. If the file is not located in the classpath, enter the full (or relative) path to the file as well as 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:

```
<pof-config>
  <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, make sure that the location of the POF configuration files are located in the classpath at run time.

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

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

Viewing Which POF Configuration Files are Loaded

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

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

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

Specifying Management Configuration Files

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

- `report-group.xml` – The default report group configuration file that 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, this file is located in the `/reports` directory of the `coherence.jar`. An additional report group definition file called `report-all.xml` is also located in the `/reports` directory and references a comprehensive set of reports. Custom report group definition files can be created as required.

- report definition files – A set of default report definition files that each results in the creation of a report file that displays management information for a particular metric. Report definition files must be referenced in a report group definition file in order to be used at run time. The default report definition files are located in the `/reports` directory of the `coherence.jar` and are referenced by the default report group definition file. Custom report definition 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/or monitored from any node within the cluster. Custom MBeans can be defined within any operational override file. However, it is recommended that the MBean configuration override file be used instead.

The following topics are included in this section:

- [Specifying a Custom Report Group Configuration File](#)
- [Specifying an MBean Configuration File](#)
- [Viewing Which Management Configuration Files are Loaded](#)

Refer to [Part VI, "Managing Coherence"](#) for detailed instructions on configuring management.

Specifying a Custom Report Group Configuration File

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

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

Overriding the Default Report Group Configuration File

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

To override the default report group configuration file:

1. Create an XML file.
2. Edit the file and create an empty `<report-group>` node as follows:

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

<!DOCTYPE report-group SYSTEM "report-group.dtd">

<report-group>
</report-group>
```

3. Save and close the file.
4. Create a `tangosol-coherence-override.xml` file as described in ["Using the Default Operational Override File"](#) on page 3-3.

5. Edit the file and enter a `<management-config>` node that specifies the name of the report group configuration file. If the report group configuration file is not located in the classpath, enter the full (or relative) path to the file as well. The element also supports the use of a URL when specifying the location of a report group configuration file.

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

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

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

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

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

Using the Report Group Configuration File System Property

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

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

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

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

Specifying an MBean Configuration File

The `tangosol-coherence.xml` operational deployment descriptor defines an operational override file that is named `custom-mbeans.xml` and is specifically used

to define custom MBeans. A name and location of the override file may also be specified using the MBean configuration file system property.

Using the Default MBean Configuration Override File

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

To use the default MBean configuration override file:

1. Create a file named `custom-mbeans.xml`.
2. Edit the file and create an empty `<mbeans>` node as follows:


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

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

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

Using the MBean Configuration File System Property

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

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

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

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

Viewing Which Management Configuration Files are Loaded

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

Report Group Configuration File

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

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

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

MBean Configuration Override File

The MBean configuration message is emitted together 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"
```

Understanding the XML Override Feature

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

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

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

The following topics are included in this section:

- [Using the Predefined Override Files](#)
- [Defining Custom Override Files](#)
- [Defining Multiple Override Files for the Same Element](#)

Using the Predefined Override Files

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

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

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

```
<coherence>
  <cluster-config>
    <member-identity>
      <cluster-name>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.

```
<coherence>
  <services>
    ...
    <service id="3">
      <init-params>
        ...
        <init-param id="4">
          <param-name>local-storage</param-name>
          <param-value system-property="tangosol.coherence.distributed.
            localstorage">false</param-value>
        </init-param>
        ...
      </init-params>
    </service>
    ...
  </services>
</coherence>
```

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

```
<mbeans>
  <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 wish 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. [Table A-65](#) lists the operational elements which can contain `xml-override` and `id` attributes.

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, this means that 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:

```
<coherence>
  <cluster-config xml-override="/cluster-config.xml">
    ...
  </cluster-config>
</coherence>
```

To use this override file, create a document named `cluster-config.xml` and make sure 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.

```
<cluster-config>
  <multicast-listener>
    <join-timeout-milliseconds>4000</join-timeout-milliseconds>
  </multicast-listener>
</cluster-config>
```

An `id` attribute is used to distinguish elements that can occur more than once.

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

```
<coherence>
  <services>
    <service id="3" xml-override="/dist-service-config.xml">
    </service>
  </services>
</coherence>
```

To use this override file, create a document named `dist-service-config.xml` and make sure 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.

```
<service id="3">
  <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 form a chain of 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:

```
<coherence xml-override="{tangosol.coherence.override
/tangosol-coherence-override-{mode}.xml}">
  ...
</coherence>
```

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

```
<coherence 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:

```
<coherence xml-override="/tangosol-coherence-override-staging.xml">
  ...
</coherence>
```

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

Changing Configuration Using System Properties

The command-line override feature allows operational and cache settings to be overridden using system properties. System properties are typically specified on the Java command line using the Java `-D` option. This allows configuration to be customized for each node in a cluster while using the same operational configuration file and cache configuration file across the nodes. System properties are also a convenient and quick way to change settings during development.

The following topics are included in this section:

- [Using Preconfigured System Properties](#)
- [Creating Custom System Properties](#)

Using Preconfigured System Properties

Coherence includes many preconfigured system properties that can be used to override different operational and cache settings. [Table 3–1](#) lists all the preconfigured system properties.

To use a preconfigured system property, add the system property as a Java `-D` option at startup. The following example demonstrates using the `tangosol.coherence.log.level` system property to specify a log level when starting a cache server:

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

Table 3–1 Preconfigured System Properties

System Property	Setting
<code>tangosol.coherence.cacheconfig</code>	Cache configuration descriptor filename. See "configurable-cache-factory-config" on page A-14.
<code>tangosol.coherence.cluster</code>	Cluster name. See "member-identity" on page A-38.
<code>tangosol.coherence.clusteraddress</code>	Cluster (multicast) IP address. See <code><multicast-listener-address></code> subelement of "multicast-listener" on page A-42.
<code>tangosol.coherence.clusterport</code>	Cluster (multicast) IP port. See <code><multicast-listener-port></code> subelement of "multicast-listener" on page A-42.
<code>tangosol.coherence.distributed.backup</code>	Data backup storage location. See <code>backup-storage/type</code> subelement in "DistributedCache Service Parameters" on page A-65.
<code>tangosol.coherence.distributed.backupcount</code>	Number of data backups. See <code>backup-count</code> subelement in "DistributedCache Service Parameters" on page A-65.
<code>tangosol.coherence.distributed.localstorage</code>	Local partition management enabled. See <code>local-storage</code> subelement in "DistributedCache Service Parameters" on page A-65.
<code>tangosol.coherence.distributed.threads</code>	Thread pool size. See <code>thread-count</code> subelement in "DistributedCache Service Parameters" on page A-65.
<code>tangosol.coherence.distributed.transfer</code>	Partition transfer threshold. See <code>transfer-threshold</code> subelement in "DistributedCache Service Parameters" on page A-65.
<code>tangosol.coherence.edition</code>	Product edition. See "license-config" on page A-28.
<code>tangosol.coherence.invocation.threads</code>	Invocation service thread pool size. See <code>thread-count</code> subelement in "InvocationService Parameters" on page A-70.
<code>tangosol.coherence.localhost</code>	Unicast IP address. See <code><unicast-listener-address></code> subelement in "unicast-listener" on page A-83.
<code>tangosol.coherence.localport</code>	Unicast IP port. See <code><unicast-listener-port></code> subelement in "unicast-listener" on page A-83.
<code>tangosol.coherence.localport.adjust</code>	Unicast IP port auto assignment. See <code><unicast-listener-auto></code> subelement in "unicast-listener" on page A-83.

Table 3–1 (Cont.) Preconfigured System Properties

System Property	Setting
<code>tangosol.coherence.log</code>	Logging destination. See <code><logging-config-destination></code> subelement in "logging-config" on page A-29.
<code>tangosol.coherence.log.level</code>	Logging level. See <code><logging-config-level></code> subelement in "logging-config" on page A-29.
<code>tangosol.coherence.log.limit</code>	Log output character limit. See <code><logging-config-limit></code> subelement in "logging-config" on page A-29.
<code>tangosol.coherence.machine</code>	Machine name. See "member-identity" on page A-38.
<code>tangosol.coherence.management</code>	JMX management mode. See "management-config" on page A-32.
<code>tangosol.coherence.management.readonly</code>	JMX management read-only flag. "management-config" on page A-32.
<code>tangosol.coherence.management.remote</code>	Remote JMX management enabled flag. See "management-config" on page A-32.
<code>tangosol.coherence.member</code>	Member name. See "member-identity" on page A-38.
<code>tangosol.coherence.mode</code>	Operational mode. See "license-config" on page A-28.
<code>tangosol.coherence.override</code>	Operational configuration override filename.
<code>tangosol.coherence.priority</code>	Priority. See "member-identity" on page A-38.
<code>tangosol.coherence.process</code>	Process name "member-identity" on page A-38.
<code>tangosol.coherence.proxy.threads</code>	Coherence*Extend service thread pool size. See <code>thread-count</code> subelement in "ProxyService Parameters" on page A-71.
<code>tangosol.coherence.rack</code>	Rack name. See "member-identity" on page A-38.
<code>tangosol.coherence.role</code>	Role name. See "member-identity" on page A-38.
<code>tangosol.coherence.security</code>	Cache access security enabled flag. See "security-config" on page A-59.
<code>tangosol.coherence.security.keystore</code>	Security access controller keystore file name. See "security-config" on page A-59.
<code>tangosol.coherence.security.password</code>	Keystore or cluster encryption password. "Encryption Filters" on page 9-1.
<code>tangosol.coherence.security.permissions</code>	Security access controller permissions file name. See "security-config" on page A-59.
<code>tangosol.coherence.shutdownhook</code>	Shutdown listener action. See "shutdown-listener" on page A-73.
<code>tangosol.coherence.site</code>	Site name. See "member-identity" on page A-38.
<code>tangosol.coherence.tcnp.enabled</code>	TCMP enabled flag. See <code><packet-publisher-enabled></code> subelement in "packet-publisher" on page A-52.
<code>tangosol.coherence.tcpring</code>	!TCP ring enabled flag. See "tcp-ring-listener" on page A-79.

Table 3–1 (Cont.) Preconfigured System Properties

System Property	Setting
<code>tangosol.coherence.ttl</code>	Multicast packet time to live (TTL). See <code><multicast-listener-ttl></code> subelement in "multicast-listener" on page A-42.
<code>tangosol.coherence.wka</code>	Well known IP address. See "well-known-addresses" on page A-86.
<code>tangosol.coherence.wka.port</code>	Well known IP port. See "well-known-addresses" on page A-86.

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:

```
<coherence>
  <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:

```
<cache-config>
  <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:

```
<coherence>
  <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 that does not exist, Coherence disregards the system property.

Building Your First Coherence Application

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

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

The following sections are included in this chapter:

- [Step 1: Define the Example Cache](#)
- [Step 2: Configure and Start the Example Cluster](#)
- [Step 3: Create and Run a Basic Coherence Standalone Application](#)
- [Step 4: Create and Run a Basic Coherence JavaEE Web Application](#)
- [Using JDeveloper for Coherence Development](#)

Step 1: Define the Example Cache

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

To define the example cache:

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

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

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

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

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

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

3. Save and close the file.

Step 2: Configure and Start the Example Cluster

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

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

To configure and start the example cluster:

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

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

<!DOCTYPE coherence SYSTEM "coherence.dtd">

<coherence>
  <cluster-config>
    <member-identity>
      <cluster-name>cluster_name</cluster-name>
    </member-identity>

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



```

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

```

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

```

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

```

Step 3: Create and Run a Basic Coherence Standalone Application

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

Create the Sample Standalone Application

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

Example 4-1 The Sample HelloWorld Standalone Application

```

package com.examples;

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

public class HelloWorld {

    public static void main(String[] args) {

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

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

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

```

```
        CacheFactory.shutdown();  
    }  
}
```

Run the Sample Standalone Application

To run the standalone application example:

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

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

2. Run the Hello World application and include the location of the `coherence.jar` library and the configuration files as a Java `-cp` option. For example:

```
java -cp COHERENCE_HOME\lib\coherence.jar;COHERENCE_HOME\config  
com.example.HelloWorld
```

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

Verify the Example Cache

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

This step uses the connection 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 connection 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;COHERENCE_HOME\config  
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!
```

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

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

Create the Sample Web Application

To create the sample Web application:

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

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

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

Example 4–2 The Sample Hello World JSP

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

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

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

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

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

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

4. Copy the `coherence.jar` file to the `WEB-INF/lib` directory.

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

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

The archive should contain the following files

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

Deploy and Run the Sample Web Application

To deploy and run the Web application example:

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

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

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

Verify the Example Cache

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

This step uses the connection 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 connection 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;COHERENCE_HOME\config  
com.tangosol.net.CacheFactory
```

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

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

```
cache hello-example
```

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

```
list
```

The command returns and displays:

```
k2 = Hello World!
```

Using JDeveloper for Coherence Development

This section provides basic instructions on how to setup JDeveloper for Coherence development:

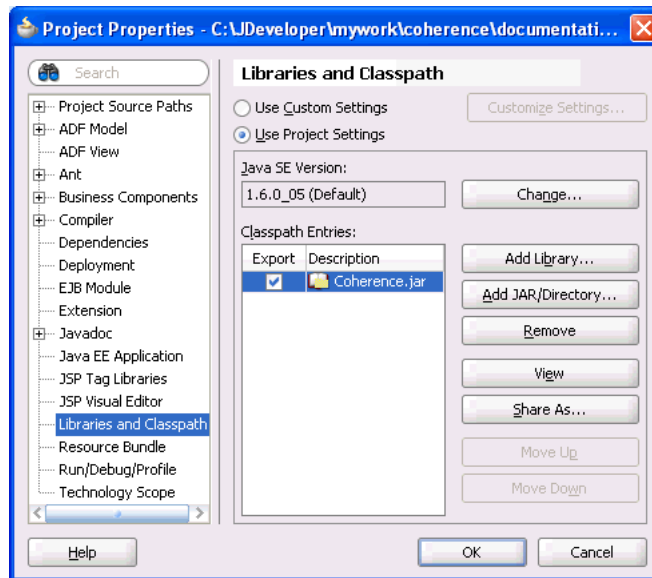
- [Running Coherence in JDeveloper](#)
- [Viewing Thread Dumps in JDeveloper](#)
- [Creating Configuration Files in JDeveloper](#)

Running Coherence in JDeveloper

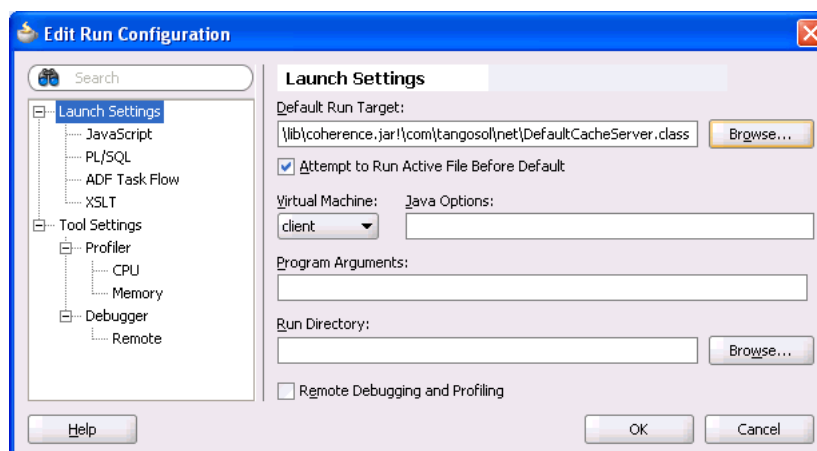
JDeveloper can be used to run cache server (`DefaultCacheServer`) and cache (`CacheFactory`) instances. Each instance is started as a separate Java process and emits standard output to the process' log. Input (such as cache commands) can be entered directly in the process as if it were started from the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

To run Coherence in JDeveloper:

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

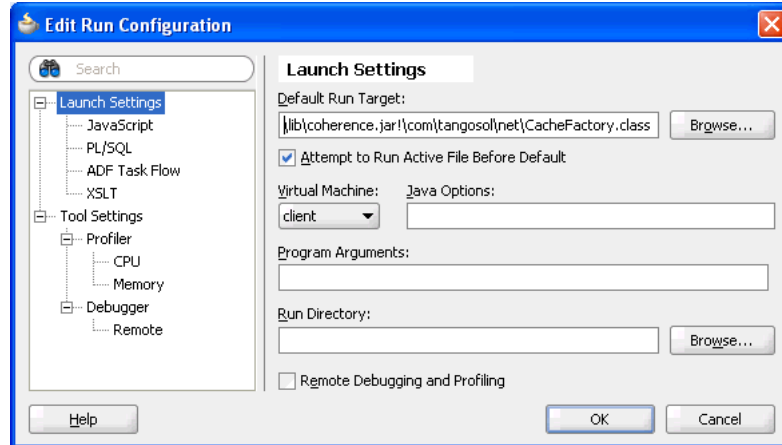


6. From the Project Properties dialog box, select the **Run/Debug/Profile** node. The Run/Debug/Profile page displays.
7. From the Run/Debug/Profile page, click **New**. The Create Run Configuration dialog box displays.
8. In the Name text box, enter a name for the new run configuration. In the Copy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

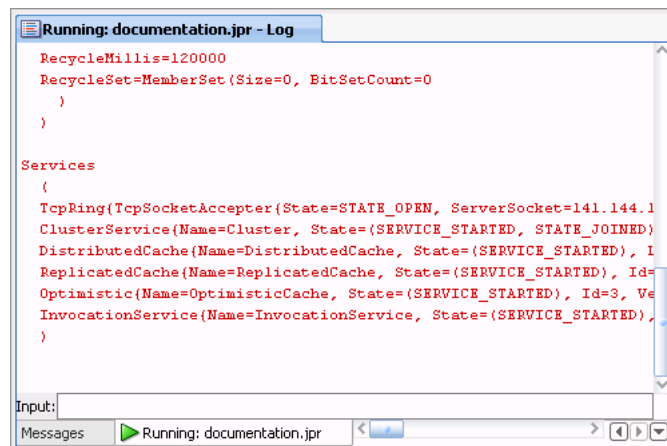


Tip: Use the Java Options text box to set Coherence system properties.

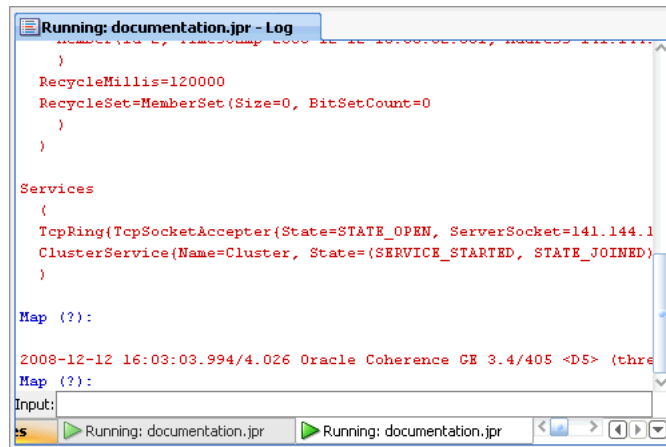
12. Select the Tool Settings Node. The Tool Settings page displays.
13. From the Additional Runner Options section, click the **Allow Program Input** check box. A check mark in the box indicates that the option is selected.
14. Click **OK**.
15. Repeat Steps 6 through 14 and select `COHERENCE_HOME\lib\coherence.jar\com\tangosol\net\CacheFactory.class`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



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



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



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

Viewing Thread Dumps in JDeveloper

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

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

To view thread dumps in JDeveloper:

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

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

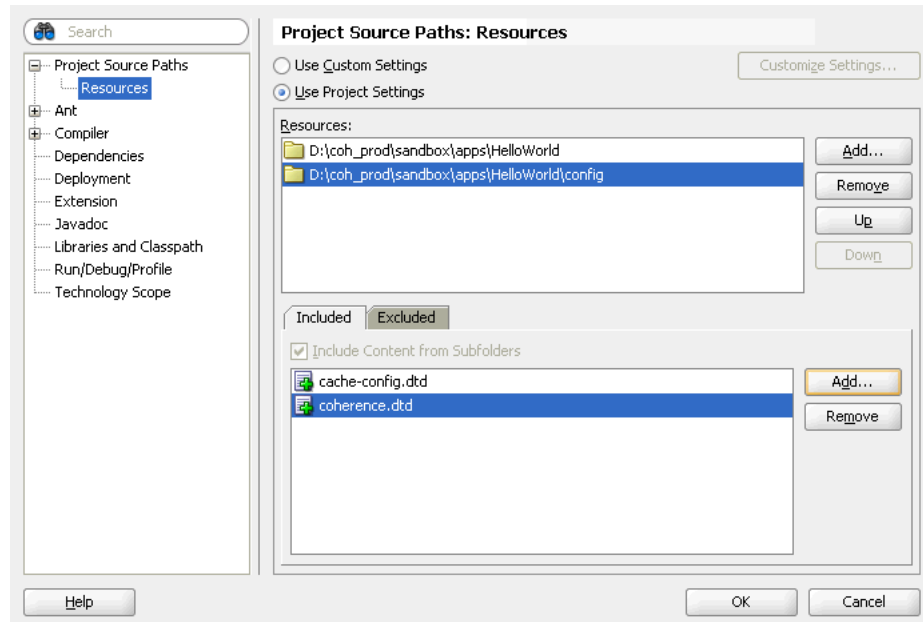
Creating Configuration Files in JDeveloper

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

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

1. Extract `cache-config.dtd` and `coherence.dtd` from the `COHERENCE_HOME\lib\coherence.jar` library to a directory on your computer.
2. In the JDeveloper Application Navigator, double-click your coherence project. The Project Properties dialog box displays.
3. Expand the **Project Source Paths** node and click **Resources**. The Resources page displays.

4. In the Resources section, click **Add** to find and select the directory where you extracted the DTD files.
5. In the Included tab, click **Add** and select the DTD files. Alternatively, you can allow JDeveloper to include all files in this directory and not explicitly add each file.
6. Click **OK**. The DTDs are listed in the Included tab as shown below.



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

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

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

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

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

```
<!DOCTYPE coherence SYSTEM "coherence.dtd">
```

The Component Palette refreshes and lists all the elements available from the `coherence.dtd` file.

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

Part II

Using Data Clusters

Part II contains the following chapters:

- [Chapter 5, "Cluster Services Overview"](#)
- [Chapter 6, "Understanding TCMP"](#)
- [Chapter 7, "Setting Single Server Mode"](#)
- [Chapter 8, "Dynamically Managing Cluster Membership"](#)
- [Chapter 9, "Using Network Filters"](#)

Cluster Services Overview

Coherence functionality is based on the concept of cluster services. Each cluster node can participate in (which implies both the ability to *provide* and to *consume*) any number of named services. These named services may already exist, which is to say that they may already be running on one or more other cluster nodes, or a cluster node can register new named services. Each named service has a service name that uniquely identifies the service within the cluster, and a service type, which defines what the service can do. There may be multiple named instances of each service type (other than the root Cluster service). By way of analogy, a service instance corresponds roughly to a database schema, and in the case of data services, a hosted named cache corresponds roughly to a database table. While services can be configured, many applications will only need to use the default set of services shipped with Coherence. There are several service types that are supported by Coherence.

Connectivity Services

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

Processing Services

- **Invocation Service:** This service provides clustered invocation and supports grid computing architectures. Using the Invocation Service, application code can 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 is the distributed cache service, which 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 will

be backed up by one or more other cluster nodes. Lastly, some cluster nodes can be configured to hold no data at all; this is useful, for example, to limit the Java heap size of an application server process, by setting the application server processes to not hold any distributed data, and by running additional cache server JVMs to provide the distributed cache storage. For more information on distributed caches, see ["Distributed Cache"](#) on page 10-1.

- **Replicated Cache Service:** This is the synchronized replicated cache service, which fully replicates all of its data to all cluster nodes that run the service. Furthermore, it supports pessimistic locking so that data can be modified in a cluster without encountering the classic missing update problem. With the introduction of near caching and continuous query caching, almost all of the functionality of replicated caches is available on top of the Distributed cache service (and with better robustness). But replicated caches are often used to manage internal application metadata. For more information on distributed caches, see ["Replicated Cache"](#) on page 10-5.
- **Optimistic Cache Service:** This is the optimistic-concurrency version of the Replicated Cache Service, which fully replicates all of its data to all cluster nodes, and employs an optimization similar to optimistic database locking to maintain coherency. Coherency refers to the fact that all servers will end up with the same "current" value, even if multiple updates occur at the same exact time from different servers. The Optimistic Cache Service does not support pessimistic locking, so in general it should only be used for caching "most recently known" values for read-only uses. This service is rarely used. For more information on distributed caches, see ["Optimistic Cache"](#) on page 10-7.

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

It is important to note that these are only the basic clustered 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 can provide an extremely flexible, configurable and powerful set of options for clustered applications. For example, the Near Cache functionality uses a Distributed Cache as one of its components.

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`. The `Map` interface is the same API that is implemented by the Java `Hashtable` class, for example.

Understanding TCMP

The following sections are included in this chapter:

- [Overview](#)
- [Protocol Reliability](#)
- [Protocol Resource Utilization](#)
- [Protocol Tunability](#)
- [Multicast Scope](#)
- [Disabling Multicast](#)

Overview

Coherence uses Tangosol Cluster Management Protocol (TCMP), a clustered IP-based protocol, for server discovery, cluster management, service provisioning and data transmission. To ensure true scalability, the TCMP protocol is completely asynchronous, meaning that communication is never blocking, even when many threads on a server are communicating at the same time. Further, the asynchronous nature 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 will affect the speed of certain operations.

TCMP uses a combination of UDP/IP multicast, UDP/IP unicast and TCP/IP as follows:

- **Multicast**
 - Cluster discovery: Is there a cluster already running that a new member can join?
 - Cluster heartbeat: The most senior member in the cluster issues a periodic heartbeat through multi-cast; the rate can be configured and defaults to once per second.
 - Message delivery: Messages that need to be delivered to multiple cluster members will often be sent through multicast, instead of unicasting the message one time to each member.
- **Unicast**
 - Direct member-to-member ("point-to-point") communication, including messages, 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.
- **TCP**
 - An optional TCP/IP ring is used as an additional "death detection" mechanism, to differentiate between actual node failure and an unresponsive node, such as when a JVM conducts a full GC.
 - TCP/IP is **not** used as a data transfer mechanism due to the intrinsic overhead of the protocol and its synchronous nature.

Protocol Reliability

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

Protocol Resource Utilization

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

The optional TCP/IP ring will use a few additional TCP/IP sockets, and a total of one additional thread.

Protocol Tunability

The TCMP protocol is very tunable to take advantage of specific network topologies, or to add tolerance for low-bandwidth and/or high-latency segments in a geographically distributed cluster. Coherence comes with a pre-set configuration. Some TCMP attributes are dynamically self-configuring at runtime, but can also be overridden and locked down for deployment purposes.

Multicast Scope

Multicast UDP/IP packets are configured with a time-to-live value (TTL) that designates how far those packets can travel on a network. The TTL is expressed in terms of how many "hops" a packet will survive; each network interface, router and managed switch is considered one hop. Coherence provides a TTL setting to limit the scope of multicast messages.

Disabling Multicast

In most WAN environments, and some LAN environments, multicast traffic is disallowed. To prevent Coherence from using multicast, configure a list of `well-known-addresses` (WKA). This will disable multicast discovery, and also disable multicast for all data transfer. Coherence is designed to use point-to-point communication as much as possible, so most application profiles will not see a substantial performance impact.

Setting Single Server Mode

If you want to perform unit testing or quick restarts, you might find it more convenient to avoid the network and run in single-server mode. To constrain Coherence to run on a single server, set the multicast packet time-to-live to 0, and set the unicast IP address.

You can configure these properties either by declaring system properties on the command line or by editing the values in the operational configuration descriptor, `tangosol-coherence.xml` file.

Setting Single Server Mode in the Operation Configuration Descriptor

In the `tangosol-coherence.xml` file, the multicast packet time to live value is defined by the `<time-to-live>` subelement of the `<multicast-listener>` element. The `<time-to-live>` value determines the maximum number of "hops" a packet may traverse between network segments. Setting this subelement to 0 keeps the packets from leaving the originating machine.

The unicast IP address is defined by the `<address>` subelement of the `<unicast-listener>` element. This subelement specifies the IP address that a Socket will listen or publish on and must be routed to loopback as well. Setting this subelement to an IP address that is never used will prevent Coherence from joining the network.

Note: The "localhost" setting may not work on systems that define localhost as the loopback address; in that case, specify the machine name or the specific IP address.

The following XML code fragment illustrates a single server mode configuration in the `tangosol-coherence.xml` file.

Example 7-1 Single Server Mode Configuration

```
<coherence>
  <cluster-config>
    ...
    <multicast-listener>
      <time-to-live>0<\time-to-live>
    ...
    <multicast-listener>
    ...
    <unicast-listener>
      <address>127.0.0.1<\address>
```

```
...
<\unicast-listener>
...
<\cluster-config>
<\coherence>
```

Setting Single Server Mode on the Command Line

Coherence defines system properties that allow you to set the multicast packet time-to-live and the unicast IP address for single server mode on the command line. This feature is useful when you need to change the settings for a single JVM, or if you want to start an application with settings that differ from those in the descriptor files.

The following system properties can be used to define single server mode.

- `tangosol.coherence.ttl`—Multicast packet time to live. Set to "0" to keep the packets from leaving the originating machine.
- `tangosol.coherence.localhost`—Unicast IP address.

The sample command line in [Example 7-2](#) illustrates starting coherence in single server mode:

Example 7-2 Command to Start Coherence in Single Server Mode

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

See [Appendix C, "Command Line Overrides"](#) for more information on system properties defined by Coherence.

Dynamically Managing Cluster Membership

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

Cluster and Service Objects

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

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

From the `Cluster` object, the application can determine the set of services that run in the cluster. This is illustrated in [Example 8-1](#).

Example 8-1 Determining Services Running in the Cluster

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

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

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

Member object

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

- The Member's IP address

- What date/time the Member joined the cluster

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

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

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

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

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

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

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

Listening to Member Events

Applications must create a class that implements the `MemberListener` interface (see [Example 8-2](#)) in order to listen to cluster and/or service membership changes. The listener class is then added on a service by either using the service's `addMemberListener` method or by adding a `<member-listener>` element to a cache scheme definition.

There are two advantages to using the configuration approach versus the programmatic approach. First, programmatically, listeners can only be added to a service that is already 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>  
  <backing-map-scheme>  
    <local-scheme>  
      <scheme-ref>example-binary-backing-map</scheme-ref>  
    </local-scheme>  
  </backing-map-scheme>  
  <member-listener>package.MyMemberListener</member-listener>
```

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

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

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

[Example 8-2](#) demonstrates a `MemberListener` implementation that prints out all the membership events that it receives:

Example 8-2 A Sample `MemberListener` Implementation

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

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

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

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

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

[Example 8-3](#) illustrates how information encapsulated in a `MemberEvent` object can be used.

Example 8-3 Using Event Type Information in a `MemberEvent` Object

```
public class RejoinEventPrinter
    extends Base
```

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

        public void memberLeaving(MemberEvent evt)
        {
        }

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

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

Using Network Filters

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

Compression Filters

The compression filter is based on the `java.util.zip` package and compresses message contents thus reducing the network load. This is useful when there is ample CPU available but insufficient network bandwidth. See ["Configuring Filters"](#) on page 9-4 for information on enabling this filter.

Encryption Filters

Coherence ships with two JCA-based encryption filters which can be used to protect the clustered communications for privacy and authenticity: The Symmetric Encryption Filter and the PKCS Encryption Filter.

Note: Using SSL is strongly recommended over encryption filters. See ["Using SSL in Coherence"](#) on page 30-6.

Symmetric Encryption Filter

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

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

Example 9-1 Enabling a Filter for all Network Traffic

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

The shared password may either be specified in the `<filters>` section of the operational configuration file, or by using the `tangosol.coherence.security.password` system property. See "Symmetric Encryption Filter Parameters" on page 9-2 for additional configuration options.

Symmetric Encryption Filter Parameters

The symmetric encryption filter supports the parameters listed in Table 9–1. See the `com.tangosol.net.security.PasswordBasedEncryptionFilter` Javadoc for additional configuration details.

Table 9–1 Symmetric Encryption Filter Parameters

Parameter Name	Value Description
algorithm	Specifies the mechanism to use in deriving a secret key from the above material. Default value is <code>PBEWithMD5AndDES</code> .
iterations	Specifies the iteration count to use in deriving the key. Default value is 32.
password	Specifies the raw material used to generate the secret key. The system property override is <code>tangosol.coherence.security.password</code> .
salt	Specifies the salt to use in deriving the key. Default value is <code>nosecret</code> .

PKCS Encryption Filter

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

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

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

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

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

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

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

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

```
keytool -import -alias remote_1 -storepass secret -keystore ./keystore.jks -file
local_1.cert
keytool -import -alias remote_2 -storepass secret -keystore ./keystore.jks -file
local_2.cert
keytool -import -alias remote_3 -storepass secret -keystore ./keystore.jks -file
local_3.cert
```

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

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

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

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

The keystore and alias password can be specified either in the `<filters>` node of the operational configuration file, or by using the `tangosol.coherence.security.password` system property.

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

Note: Using this filter may require a change to the packet size configuration depending on the size of certificates used. Set the `<maximum-length>` to a value larger than the certificate size (allowing some overhead). See ["packet-size"](#) on page A-54.

PKCS Encryption Filter Parameters

The PKCS encryption filter supports the following parameters, see ["Encryption Filters"](#) on page 9-1 section for examples, or the `com.tangosol.net.security.ClusterEncryptionFilter` Javadoc for additional configuration details.

Table 9-2 PKCS Encryption Filter Parameters

Parameter Name	Description
<code>asymmetricFilterClassName</code>	Specifies the asymmetric filter implementation. Default value is <code>com.tangosol.net.security.AsymmetricEncryptionFilter</code> .
<code>keyAlias</code>	Specifies the alias to use in reading the key from the keystore.

Table 9–2 (Cont.) PKCS Encryption Filter Parameters

Parameter Name	Description
keyPassword	Specifies the password to use in reading the key. The preconfigured system property is <code>tangosol.coherence.security.password</code> .
store	Specifies the path to the KeyStore Default value is <code>.keystore</code> .
sharedKeySize	Specifies the size of shared key. Default value is 112.
sharedKeyType	Specifies the type of shared key. Default value is <code>DESede</code> .
storePassword	Specifies the password to use to access the store If unspecified value of <code>keyPassword</code> parameter will be used.
storeType	Specifies the type of KeyStore. Default value is <code>JKS</code> .
transformation	Specifies the transformation to use. Default value is <code>RSA/NONE/PKCS1Padding</code> .

Configuring Filters

There are two steps to configuring a filter.

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

Example 9–2 Declaring a Filter in the tangosol-coherence.xml File

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

See [Appendix A, "Operational Configuration Elements"](#) for more information on the structure of the `<filters>` XML element.

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

Example 9–3 Attaching the Filter to a Service

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

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

Example 9-4 Adding the Filter to All Services

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

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

Creating a Custom Filter

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

If the filter class implements the `XmlConfigurable` interface, then Coherence will configure the filter after instantiating it. [Example 9-5](#) illustrates a filter declaration in the `tangosol-coherence.xml` file. If the filter is associated with a service type, every time a new service is started of that type, Coherence will instantiate the `CompressionFilter` class and will hold it with the service until the service stops. If the filter is associated with all outgoing messages, Coherence will instantiate the filter on startup and will hold it until the cluster stops.

Example 9-5 Configuration for a Custom Filter

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

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

Example 9-6 Configuring a *setConfig* Call for a Filter

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

Part III

Using Caches

Part III contains the following chapters:

- [Chapter 10, "Introduction to Caches"](#)
- [Chapter 11, "Configuring Caches"](#)
- [Chapter 12, "Implementing Storage and Backing Maps"](#)
- [Chapter 13, "Read-Through, Write-Through, Write-Behind, and Refresh-Ahead Caching"](#)
- [Chapter 14, "Serialization Paged Cache"](#)
- [Chapter 15, "Cache Configurations by Example"](#)

Introduction to Caches

This chapter provides an overview and comparison of basic cache types offered by Coherence. The chapter includes the following sections:

- [Distributed Cache](#)
- [Replicated Cache](#)
- [Optimistic Cache](#)
- [Near Cache](#)
- [Local Cache](#)
- [Remote Cache](#)
- [Summary of Cache Types](#)

Distributed Cache

A distributed, or partitioned, cache is a clustered, fault-tolerant cache that has linear scalability. Data is partitioned among all the machines of the cluster. For fault-tolerance, partitioned caches can be configured to keep each piece of data on one or more unique machines within a cluster. Distributed caches are the most commonly used caches in Coherence.

Coherence defines a distributed cache as a collection of data that is distributed (or, *partitioned*) across any number of cluster nodes such that exactly one node in the cluster is responsible for each piece of data in the cache, and the responsibility is distributed (or, load-balanced) among the cluster nodes.

There are several key points to consider about a distributed cache:

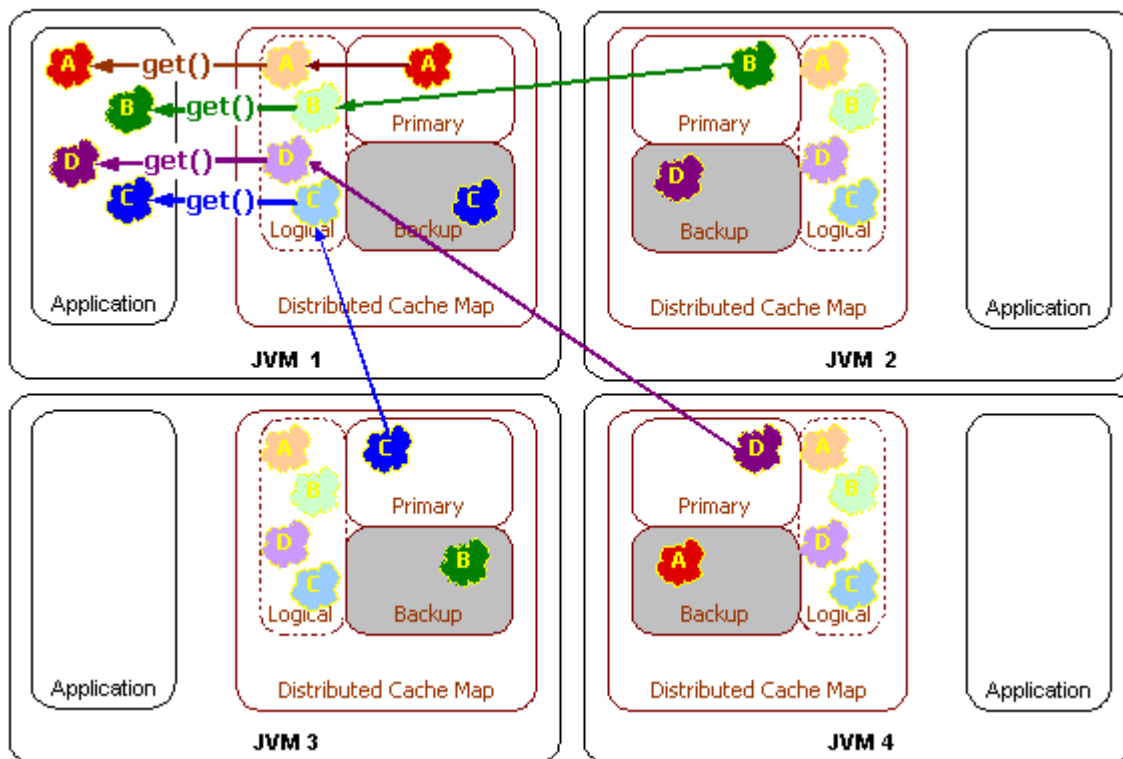
- **Partitioned:** The data in a distributed cache is spread out over all the servers in such a way that no two servers are responsible for the same piece of cached data. This means that the size of the cache and the processing power associated with the management of the cache can grow linearly with the size of the cluster. Also, it means that operations against data in the cache can be accomplished with a "single hop," in other words, involving at most one other server.
- **Load-Balanced:** Since the data is spread out evenly over the servers, the responsibility for managing the data is automatically load-balanced across the cluster.
- **Location Transparency:** Although the data is spread out across cluster nodes, the exact same API is used to access the data, and the same behavior is provided by each of the API methods. This is called location transparency, which means that the developer does not have to code based on the topology of the cache, since the

API and its behavior will be 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.

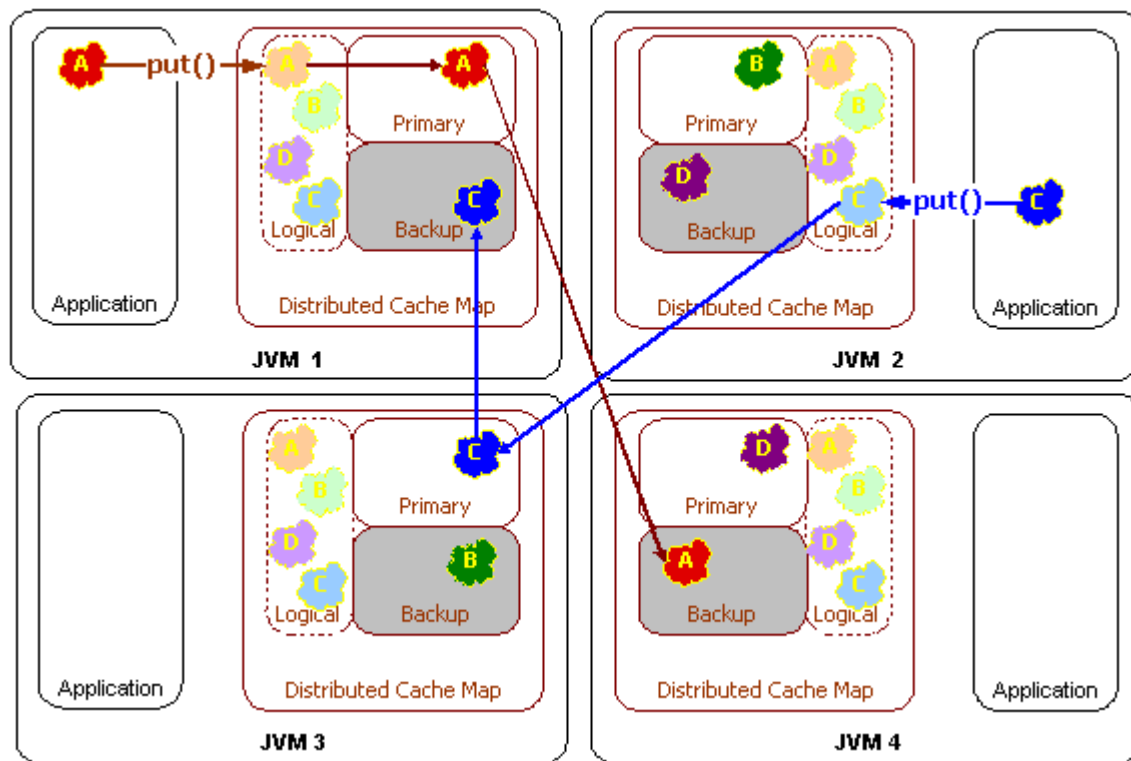
Access to the distributed cache will often need to go over the network to another cluster node. All other things equals, if there are n cluster nodes, $(n - 1) / n$ operations will go over the network:

Figure 10–1 *Get Operations in a Partitioned Cache Environment*



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

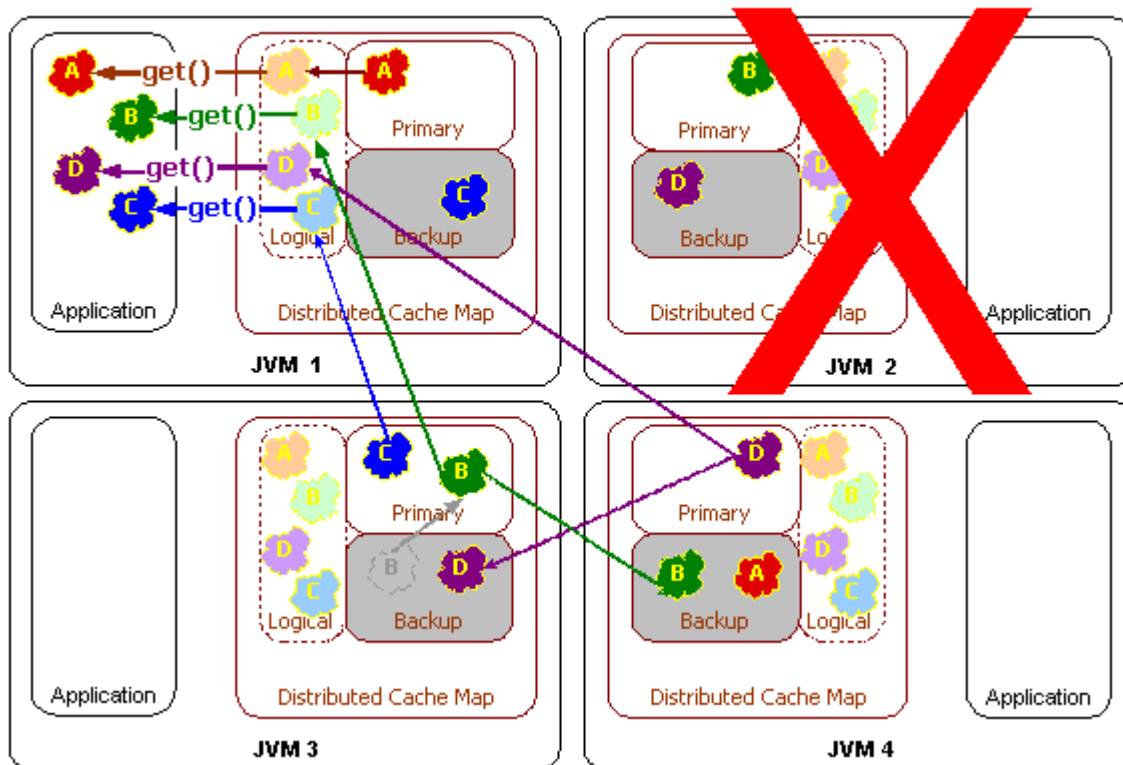
Similarly, a cache update operation can use the same single-hop point-to-point approach, which addresses one of the two known limitations of a replicated cache, the need to push cache updates to all cluster nodes.

Figure 10–2 Put Operations in a Partitioned Cache Environment

In the figure above, the data is being sent to a primary cluster node and a backup cluster node. This is for failover purposes, and corresponds to a backup count of one. (The default backup count setting is one.) If the cache data were not critical, which is to say that it could be re-loaded from disk, the backup count could be set to zero, which would allow some portion of the distributed cache data to be lost in the event of a cluster node failure. 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. This means that 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 will be lost.

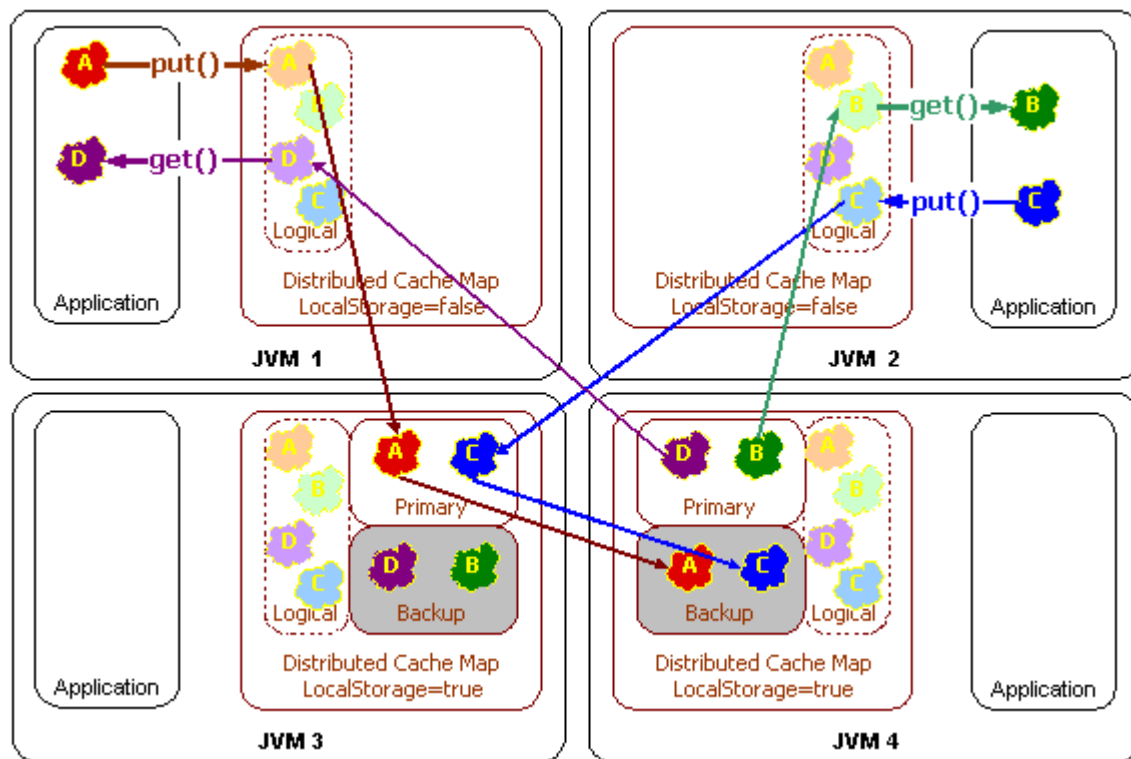
Failover of a distributed cache involves promoting backup data to be primary storage. When a cluster node fails, all remaining cluster nodes determine what data each holds in backup that the failed cluster node had primary responsible for when it died. Those data becomes the responsibility of whatever cluster node was the backup for the data:

Figure 10–3 Failover in a Partitioned Cache Environment

If there are multiple levels of backup, the first backup becomes responsible for the data; the second backup becomes the new first backup, and so on. Just as with the replicated cache service, lock information is also retained in the case of 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 will provide the cache storage and the backup storage for the distributed cache. Regardless of this setting, all cluster nodes will have the same exact view of the data, due to location transparency.

Figure 10–4 Local Storage in a Partitioned Cache Environment



There are several benefits to the local storage enabled option:

- The Java heap size of the cluster nodes that have turned off local storage enabled will not be affected at all by the amount of data in the cache, because that data will be cached on other cluster nodes. This is particularly useful for application server processes running on older JVM versions with large Java heaps, because those processes often suffer from garbage collection pauses that grow exponentially with the size of the heap.
- Coherence allows each cluster node to run any supported version of the JVM. That means that cluster nodes with local storage enabled turned on could be running a newer JVM version that supports larger heap sizes, or Coherence's off-heap storage using the Java NIO features.
- The local storage enabled option allows some cluster nodes to be used just for storing the cache data; such cluster nodes are called Coherence cache servers. Cache servers are commonly used to scale up Coherence's distributed query functionality.

Replicated Cache

A replicated cache is a clustered, fault tolerant cache where data is fully replicated to every member in the cluster. This cache offers the fastest read performance with linear performance scalability for reads but poor scalability for writes (as writes must be processed by every member in the cluster). Because data is replicated to all machines, adding servers does not increase aggregate cache capacity.

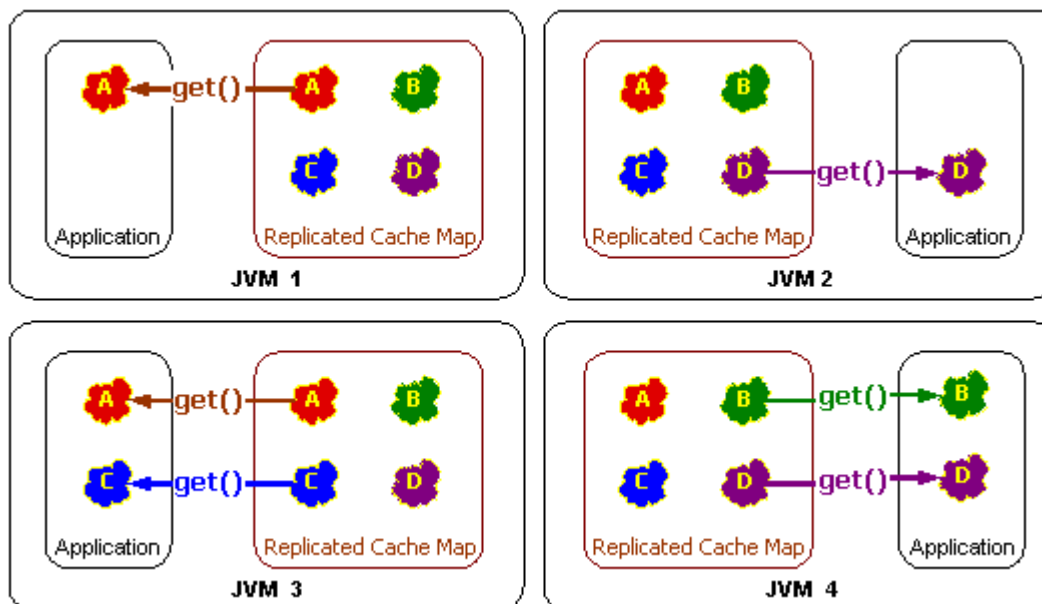
The replicated cache excels in its ability to handle data replication, concurrency control and failover in a cluster, all while delivering in-memory data access speeds. A

clustered replicated cache is exactly what it says it is: a cache that replicates its data to all cluster nodes.

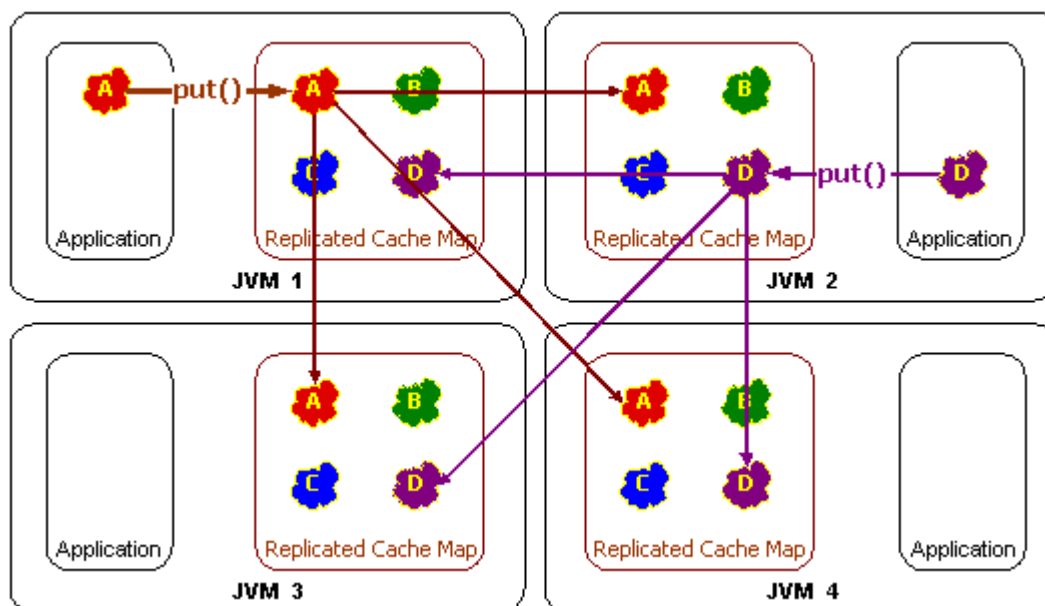
There are several challenges to building a reliable replicated cache. The first is how to get it to scale and perform well. Updates to the cache have to be sent to all cluster nodes, and all cluster nodes have to end up with the same data, even if multiple updates to the same piece of data occur at the same time. Also, if a cluster node requests a lock, it should not have to get all cluster nodes to agree on the lock, otherwise it will scale extremely poorly; yet in the case of cluster node failure, all of the data and lock information must be kept safely. Coherence handles all of these scenarios transparently, and provides the most scalable and highly available replicated cache implementation available for Java applications.

The best part of a replicated cache is its access speed. Since the data is replicated to each cluster node, it is available for use without any waiting. This is referred to as "zero latency access," and is perfect for situations in which an application requires the highest possible speed in its data access. Each cluster node (JVM) accesses the data from its own memory:

Figure 10–5 *Get Operation in a Replicated Cache Environment*



In contrast, updating a replicated cache requires pushing the new version of the data to all other cluster nodes:

Figure 10–6 Put Operation in a Replicated Cache Environment

Coherence implements its replicated cache service in such a way that all read-only operations occur locally, all concurrency control operations involve at most one other cluster node, and only update operations require communicating with all other cluster nodes. The result is excellent scalable performance, and as with all of the Coherence services, the replicated cache service provides transparent and complete failover and failback.

The limitations of the replicated cache service should also be carefully considered. First, however much data is managed by the replicated cache service is on each and every cluster node that has joined the service. That means that memory utilization (the Java heap size) is increased for each cluster node, which can impact performance. Secondly, replicated caches with a high incidence of updates will not scale linearly as the cluster grows; in other words, the cluster will suffer diminishing returns as cluster nodes are added.

Optimistic Cache

An optimistic cache is a clustered cache implementation similar to the replicated cache implementation but without any concurrency control. This implementation offers higher write throughput than a replicated cache. It also allows an alternative underlying store for the cached data (for example, a MRU/MFU-based cache). However, if two cluster members are independently pruning or purging the underlying local stores, it is possible that a cluster member may have a different store content than that held by another cluster member.

Near Cache

A near cache is a hybrid cache; it typically fronts a distributed cache or a remote cache with a local cache. Near cache invalidates front cache entries, using a configured invalidation strategy, and provides excellent performance and synchronization. Near cache backed by a partitioned cache offers zero-millisecond local access for repeat data access, while enabling concurrency and ensuring coherency and fail-over, effectively combining the best attributes of replicated and partitioned caches.

The objective of a Near Cache is to provide the best of both worlds between the extreme performance of the [Replicated Cache](#) and the extreme scalability of the [Distributed Cache](#) by providing fast read access to Most Recently Used (MRU) and Most Frequently Used (MFU) data. To achieve this, 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.

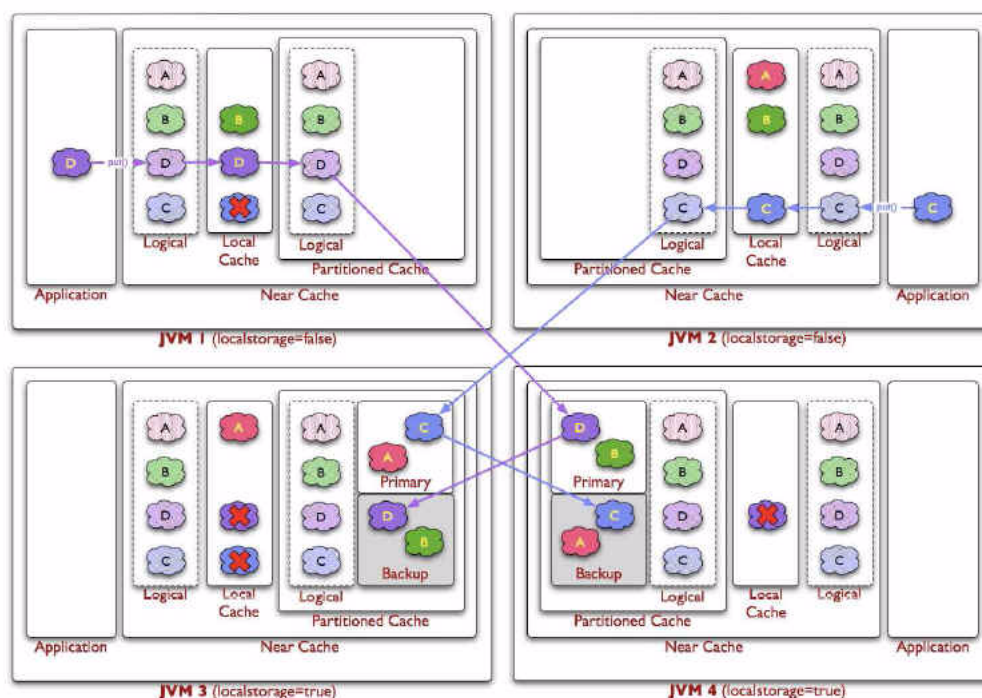
The "front cache" provides local cache access. It is assumed to be inexpensive, in that it is fast, and is limited in terms of size. The "back cache" can be a centralized or multi-tiered cache that can load-on-demand in case of local cache misses. The "back cache" is assumed to be complete and correct in that it has much higher capacity, but more expensive in terms of access speed. The use of a Near Cache is not confined to Coherence*Extend; it also works with TCMP.

This design allows Near Caches to configure cache coherency, from the most basic expiry-based caches and invalidation-based caches, up to advanced caches that version data and provide guaranteed coherency. The result is a tunable balance between the preservation of local memory resources and the performance benefits of truly local caches.

The typical deployment uses a [Local Cache](#) for the "front cache". A Local Cache is a reasonable choice because it is thread safe, highly concurrent, size-limited and/or auto-expiring and stores the data in object form. For the "back cache", a remote, partitioned cache is used.

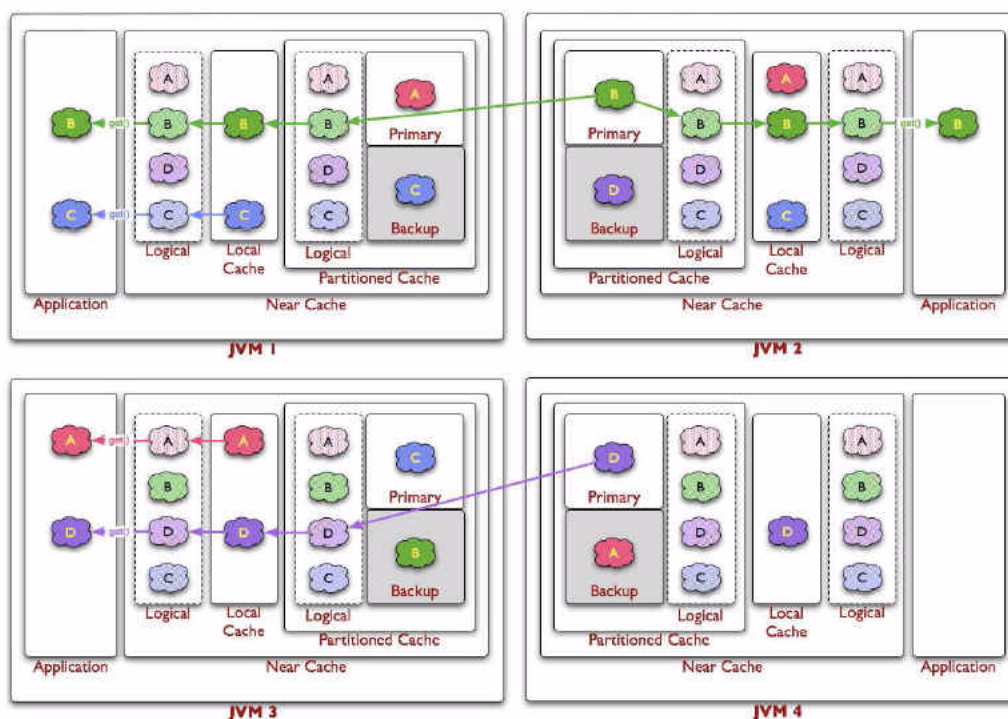
The following figure illustrates the data flow in a Near Cache. If the client writes an object D into the grid, the object is placed in the local cache inside the local JVM and in the partitioned cache which is backing it (including a backup copy). If the client requests the object, it can be obtained from the local, or "front cache", in object form with no latency.

Figure 10–7 Put Operations in a Near Cache Environment



If the client requests an object that has been expired or invalidated from the "front cache", then Coherence will automatically retrieve the object from the partitioned cache. The updated object will be written to the "front cache" and then delivered to the client.

Figure 10–8 Get Operations in a Near Cache Environment



Local Cache

While it is not a clustered service, the Coherence local cache implementation is often used in combination with various Coherence clustered cache services. The Coherence local cache is just that: a cache that is local to (completely contained within) a particular cluster node. There are several attributes of the local cache that are particularly interesting:

- The local cache implements the same standard collections interface that the clustered caches implement, meaning that there is no programming difference between using a local or a clustered cache. Just like the clustered caches, the local cache is tracking to the JCache API, which itself is based on the same standard collections API that the local cache is based on.
- The local cache can be size-limited. This means that the local cache can restrict the number of entries that it caches, and automatically evict entries when the cache becomes full. Furthermore, both the sizing of entries and the eviction policies can be customized. For example, the cache can be size-limited based on the memory used by the cached entries. The default eviction policy uses a combination of Most Frequently Used (MFU) and Most Recently Used (MRU) information, scaled on a logarithmic curve, to determine what cache items to evict. This algorithm is the best general-purpose eviction algorithm because it works well for short duration and long duration caches, and it balances frequency versus recentness to avoid

cache thrashing. The pure LRU and pure LFU algorithms are also supported, and the ability to plug in custom eviction policies.

- The local cache supports automatic expiration of cached entries, meaning that each cache entry can be assigned a time to live in the cache.
- The local cache is thread safe and highly concurrent, allowing many threads to simultaneously access and update entries in the local cache.
- The local cache supports cache notifications. These notifications are provided for additions (entries that are put by the client, or automatically loaded into the cache), modifications (entries that are put by the client, or automatically reloaded), and deletions (entries that are removed by the client, or automatically expired, flushed, or evicted.) These are the same cache events supported by the clustered caches.
- The local cache maintains hit and miss statistics. These runtime statistics can be used to accurately project the effectiveness of the cache, and adjust its size-limiting and auto-expiring settings accordingly while the cache is running.

The local cache is important to the clustered cache services for several reasons, including as part of Coherence's near cache technology, and with the modular backing map architecture.

Remote Cache

A remote cache describes any out of process cache accessed by a Coherence*Extend client. All cache requests are sent to a Coherence proxy where they are delegated to one of the other Coherence cache types (Replicated, Optimistic, Partitioned). See *Oracle Coherence Client Guide* for more information on using remote caches.

Summary of Cache Types

Numerical Terms:

- **JVMs** = number of JVMs
- **DataSize** = total size of cached data (measured without redundancy)
- **Redundancy** = number of copies of data maintained
- **LocalCache** = size of local cache (for near caches)

Table 10–1 Summary of Cache Types and Characteristics

	Replicated Cache	Optimistic Cache	Partitioned Cache	Near Cache backed by partitioned cache	LocalCache not clustered
Topology	Replicated	Replicated	Partitioned Cache	Local Caches + Partitioned Cache	Local Cache
Read Performance	Instant 5	Instant 5	Locally cached: instant 5 Remote: network speed 1	Locally cached: instant 5 Remote: network speed 1	Instant 5
Fault Tolerance	Extremely High	Extremely High	Configurable 4 Zero to Extremely High	Configurable 4 Zero to Extremely High	Zero

Table 10–1 (Cont.) Summary of Cache Types and Characteristics

	Replicated Cache	Optimistic Cache	Partitioned Cache	Near Cache backed by partitioned cache	LocalCache not clustered
Write Performance	Fast 2	Fast 2	Extremely fast 3	Extremely fast 3	Instant 5
Memory Usage (Per JVM)	DataSize	DataSize	DataSize/JVMs x Redundancy	LocalCache + [DataSize / JVMs]	DataSize
Coherency	fully coherent	fully coherent	fully coherent	fully coherent 6	n/a
Memory Usage (Total)	JVMs x DataSize	JVMs x DataSize	Redundancy x DataSize	[Redundancy x DataSize] + [JVMs x LocalCache]	n/a
Locking	fully transactional	none	fully transactional	fully transactional	fully transactional
Typical Uses	Metadata	n/a (see Near Cache)	Read-write caches	Read-heavy caches w/ access affinity	Local data

Notes:

1. As a rough estimate, with 100mb Ethernet, network reads typically require ~20ms for a 100KB object. With gigabit Ethernet, network reads for 1KB objects are typically sub-millisecond.
2. Requires UDP multicast or a few UDP unicast operations, depending on JVM count.
3. Requires a few UDP unicast operations, depending on level of redundancy.
4. Partitioned caches can be configured with as many levels of backup as desired, or zero if desired. Most installations use one backup copy (two copies total)
5. Limited by local CPU/memory performance, with negligible processing required (typically sub-millisecond performance).
6. Listener-based Near caches are coherent; expiry-based near caches are partially coherent for non-transactional reads and coherent for transactional access.

Configuring Caches

This chapter provides detailed instructions on how to configure caches within a cache configuration deployment descriptor. Refer to [Appendix B, "Cache Configuration Elements,"](#) for a complete reference of all the elements available in the descriptor. In addition, see [Chapter 15, "Cache Configurations by Example,"](#) for various sample cache configurations that can be used to create and learn about caches.

The following sections are included in this chapter:

- [Overview](#)
- [Defining Cache Mappings](#)
- [Defining Cache Schemes](#)
- [Using Scheme Inheritance](#)
- [Using Cache Scheme Properties](#)
- [Using Parameter Macros](#)

Overview

Caches are configured in a cache configuration deployment descriptor. By default, Coherence attempts to load the first `coherence-cache-config.xml` deployment descriptor that is found in the classpath. Coherence includes a sample `coherence-cache-config.xml` file in the `coherence.jar`. To use a different `coherence-cache-config.xml` file, the file must be located on the classpath and must be loaded before the `coherence.jar` library; otherwise, the sample cache configuration deployment descriptor is used. See ["Specifying a Cache Configuration File"](#) on page 3-5 for alternate methods that are available for specifying a cache configuration deployment descriptor.

The cache configuration descriptor allows caches to be defined independently from the application code. At run time, applications get an instance of a cache by referring to a cache using the name that is defined in the descriptor. This allows application code to be written independent of the cache definition. Based on this approach, cache definitions can be modified without making any changes to the application code. This approach also maximizes cache definition reuse.

The document type definition of the cache configuration descriptor is the `cache-config.dtd` file. This file is located in the root of the `coherence.jar` file. A cache configuration deployment descriptor consists of two primary elements that are detailed in this chapter: the `<caching-scheme-mapping>` element and the `<caching-schemes>` element. These elements are used to define caches schemes and to define cache names that map to the cache schemes.

Defining Cache Mappings

Cache mappings map a cache name to a cache scheme definition. The mappings provide a level of separation between applications and the underlying cache definitions. The separation allows cache implementations to be changed as required without having to change application code. Cache mappings can also be used to set initialization parameters that are applied to the underlying cache scheme definition.

Cache mappings are defined using a `<cache-mapping>` element within the `<cache-scheme-mapping>` node. Any number of cache mappings can be created. The cache mapping must include the cache name and the scheme name to which the cache name is mapped. See ["cache-mapping"](#) on page B-17 for a detailed reference of the `<cache-mappings>` element.

Using One-to-One Cache Mappings

One-to-one cache mappings map a specific cache name to a cache scheme definition. An applications must provide the exact name as specified in the mapping in order to use a cache. [Example 11-1](#) creates a single cache mapping that maps the cache name `example` to a distributed cache scheme definition with the scheme name `distributed`.

Example 11-1 Sample One-to-One Cache Mapping

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

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>example</cache-name>
      <scheme-name>distributed</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <distributed-scheme>
      <scheme-name>distributed</scheme-name>
      ...
    </distributed-scheme>
  </caching-schemes>
</cache-config>
```

Using Cache Name Pattern Mappings

Cache name pattern mappings allow applications to use patterns when specifying a cache name. Patterns are built together with the asterisk (*) wild card. Cache name patterns alleviate an application from having to know the exact name of a cache. [Example 11-2](#) creates two cache mappings. The first mapping uses the wild card (*) to map any cache name to a distributed cache scheme definition with the scheme name `distributed`. The second mapping maps the cache name pattern `account-*` to the cache scheme definition with the scheme name `account-distributed`.

Example 11-2 Sample Cache Name Pattern Mapping

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

```

<cache-config>
  <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>

```

For the first mapping, an application can use any name when creating a cache and the name will be 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 will use the cache scheme definition with the scheme name `account-distributed`.

Specifying Initialization Parameters in a Mapping

Cache mappings support the use of initialization parameters to override the properties of the underlying cache scheme definition. Initialization parameters are typically used to facilitate cache scheme definition reuse. In such cases, multiple cache names map to the same cache scheme definition, but each mapping overrides cache properties as required.

Initialization parameters are defined using an `<init-param>` element within the `<init-params>` node. The `<init-param>` element must include the `<param-name>` element and the `<param-value>` element. Any number of parameters can be specified. See ["init-param"](#) on page B-40 for a detailed reference of the `<init-param>` element.

[Example 11-3](#) creates two cache mappings that map to the same cache scheme definition. However, the first mapping overrides the `back-size-limit` property on the underlying cache scheme definition; while, the second mapping uses the `back-size-limit` as configured in the underlying cache scheme definition.

Example 11-3 Initialization Parameters in a Cache Mapping

```

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

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

```

```
<scheme-name>distributed</scheme-name>
<init-params>
  <init-param>
    <param-name>back-size-limit</param-name>
    <param-value>8MB</param-value>
  </init-param>
</init-params>
</cache-mapping>
<cache-mapping>
  <cache-name>account-*</cache-name>
  <scheme-name>distributed</scheme-name>
</cache-mapping>
</caching-scheme-mapping>
...
```

See ["Using Cache Scheme Properties"](#) on page 11-11 for more information on how cache scheme properties are configured for a cache scheme definition.

Defining Cache Schemes

Cache schemes are used to define the caches that are available to an application. Cache schemes provide a declarative mechanism that allows caches to be defined independent of the applications that use them. This removes the responsibility of defining caches from the application and allows caches to change without having to change an application's code. Cache schemes also promote cache definition reuse by allowing many applications to use the same cache definition.

Cache schemes are defined within the `<caching-schemes>` element. Each cache type (distributed, replicated, and so on) has a corresponding scheme element as well as properties that are used to define a cache of that type. Cache schemes can also be nested to allow further customized and composite caches such as near caches. See ["caching-schemes"](#) on page B-21 for a detailed reference of the `<caching-schemes>` element.

This section describes how to define cache schemes for the most often used cache types and does not represent the full set of cache types provided by Coherence. Instructions for defining cache schemes for additional cache types are found throughout this guide and are discussed as part of the features that they support. The following topics are included in this section:

- [Defining Distributed Cache Schemes](#)
- [Defining Replicated Cache Schemes](#)
- [Defining Optimistic Cache Schemes](#)
- [Defining Local Cache Schemes](#)
- [Defining Near Cache Schemes](#)

Defining Distributed Cache Schemes

The `<distributed-scheme>` element is used to define distributed caches. A distributed cache utilizes a distributed (partitioned) cache service instance. Any number of distributed caches can be defined in a cache configuration file. See ["distributed-scheme"](#) on page B-27 for a detailed reference of the `<distributed-scheme>` element.

[Example 11-4](#) defines a basic distributed cache that uses distributed as the scheme name and is mapped to the cache name example. The <autostart> element is set to true in order to start the service on a cache server node.

Example 11-4 Sample Distributed Cache Definition

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

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>example</cache-name>
      <scheme-name>distributed</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

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

In the example, the distributed cache defines a local cache to be used as the backing map. See [Chapter 12, "Implementing Storage and Backing Maps"](#) for more information on configuring backing maps.

Defining Replicated Cache Schemes

The <replicated-scheme> element is used to define replicated caches. A replicated cache utilizes a replicated cache service instance. Any number of replicated caches can be defined in a cache configuration file. See ["replicated-scheme"](#) on page B-92 for a detailed reference of the <replicated-scheme> element.

[Example 11-5](#) defines a basic replicated cache that uses replicated as the scheme name and is mapped to the cache name example. The <autostart> element is set to true in order to start the service on a cache server node.

Example 11-5 Sample Replicated Cache Definition

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

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>example</cache-name>
      <scheme-name>replicated</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <replicated-scheme>
      <scheme-name>replicated</scheme-name>
      <backing-map-scheme>
```

```
        <local-scheme/>
      </backing-map-scheme>
    <autostart>true</autostart>
  </replicated-scheme>
</caching-schemes>
</cache-config>
```

In the example, the replicated cache defines a local cache to be used as the backing map. See [Chapter 12, "Implementing Storage and Backing Maps"](#) for more information on configuring backing maps.

Defining Optimistic Cache Schemes

The `<optimistic-scheme>` element is used to define optimistic caches. An optimistic cache utilizes an optimistic cache service instance. Any number of optimistic caches can be defined in a cache configuration file. See ["optimistic-scheme"](#) on page B-65 for a detailed reference of the `<optimistic-scheme>` element.

[Example 11-6](#) defines a basic optimistic cache that uses `optimistic` as the scheme name and is mapped to the cache name `example`. The `<autostart>` element is set to `true` in order to start the service on a cache server node.

Example 11-6 Sample Optimistic Cache Definition

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

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>example</cache-name>
      <scheme-name>optimistic</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

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

In the example, the optimistic cache defines a local cache to be used as the backing map. See [Chapter 12, "Implementing Storage and Backing Maps"](#) for more information on configuring backing maps.

Defining Local Cache Schemes

The `<local-scheme>` element is used to define local caches. Local caches are generally nested within other cache schemes, for instance as the front-tier of a near cache. Thus, this element can appear as a sub-element of any of the following elements: `<caching-schemes>`, `<distributed-scheme>`, `<replicated-scheme>`, `<optimistic-scheme>`, `<near-scheme>`, `<versioned-near-scheme>`, `<overflow-scheme>`, `<read-write-backing-map-scheme>`, `<versioned-backing-map-scheme>`,

and `<backing-map-scheme>`. See ["local-scheme"](#) on page B-55 for a detailed reference of the `<local-scheme>` element.

[Example 11-7](#) defines a local cache that uses `local` as the scheme name and is mapped to the cache name `example`.

Note: A local cache is not typically used as a standalone cache on a cache server; moreover, a cache server will not start if the only cache definition in the cache configuration file is a local cache.

Example 11-7 Sample Local Cache Definition

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

<cache-config>
  <cache-scheme-mapping>
    <cache-mapping>
      <cache-name>example</cache-name>
      <scheme-name>local</scheme-name>
    </cache-mapping>
  </cache-scheme-mapping>

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

See ["Configuring a Local Cache for C++ Clients"](#) and ["Configuring a Local Cache for .NET Clients"](#) in the *Oracle Coherence Client Guide* when using Coherence*Extend.

Controlling the Growth of a Local Cache

As shown in [Table 11-7](#), the `<local-scheme>` provides several optional sub-elements that control the growth of the cache. For example, the `<low-units>` and `<high-units>` sub-elements limit the cache in terms of size. When the cache reaches its maximum allowable size it prunes itself back to a specified smaller size, choosing which entries to evict according to a specified eviction-policy (`<eviction-policy>`). The entries and size limitations are measured in terms of units as calculated by the scheme's unit-calculator (`<unit-calculator>`).

Local caches use the `<expiry-delay>` cache configuration element to configure the amount of time that items may remain in the cache before they expire. Client threads initiate these actions while accessing the cache. This means that the `<expiry-delay>` time may be reached, but not initiated until a client thread accesses the cache. For example, if the `<expiry-delay>` value is set at 10 seconds (10s) and a client accesses the cache after 15 seconds, then expiry occurs after 15 seconds.

Note: The client thread performs the evictions, not a background thread. In addition, the expiry delay parameter (`cExpiryMillis`) is defined as an integer and is expressed in milliseconds. Therefore, the maximum amount of time can never exceed `Integer.MAX_VALUE` (2147483647) milliseconds or approximately 24 days.

Defining Near Cache Schemes

The `<near-scheme>` element is used to define a near cache. A near cache is a composite cache because it is comprised of 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 as well as any of the standalone cache types. See ["near-scheme"](#) on page B-58 for a detailed reference of the `<near-scheme>` element.

[Example 11-8](#) defines of a near cache that uses `near` as the scheme name and is mapped to the cache name `example`. The front-tier is a local cache and the back-tier is a distributed cache.

Note: Near caches are used for cache clients and are not typically used on a cache server; moreover, a cache server will not start if the only cache definition in the cache configuration file is a near cache.

Example 11-8 Sample Near Cache Definition

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

<cache-config>
  < 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>
```

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

See "Defining a Near Cache for C++ Clients" and "Defining a Near Cache for .NET Clients" in the *Oracle Coherence Client Guide* when using Coherence*Extend.

Near Cache Invalidation Strategies

The `<invalidation-strategy>` is an optional subelement for a near cache. An invalidation strategy is used to specify how the front-tier and back-tier objects will be kept synchronous. A near cache can be configured to listen to certain events in the back cache and automatically update or invalidate entries in the front cache. Depending on the interface that the back cache implements, the near cache provides four different strategies of invalidating the front cache entries that have changed by other processes in the back cache.

[Table 11–1](#) describes the invalidation strategies. You can find more information on the invalidation strategies and the read-through/write-through approach in [Chapter 13, "Read-Through, Write-Through, Write-Behind, and Refresh-Ahead Caching."](#)

Table 11–1 Near Cache Invalidation Strategies

Strategy Name	Description
None	This strategy instructs the cache not to listen for invalidation events at all. This is the best choice for raw performance and scalability when business requirements permit the use of data which might not be absolutely current. Freshness of data can be guaranteed by use of a sufficiently brief eviction policy for the front cache.
Present	This strategy instructs a near cache to listen to the back cache events related only to the items currently present in the front cache. This strategy works best when each instance of a front cache contains distinct subset of data relative to the other front cache instances (for example, sticky data access patterns).
All	This strategy instructs a near cache to listen to all back cache events. This strategy is optimal for read-heavy tiered access patterns where there is significant overlap between the different instances of front caches.
Auto	This strategy instructs a near cache to switch automatically between <code>Present</code> and <code>All</code> strategies based on the cache statistics.

Using Scheme Inheritance

Scheme inheritance allows cache schemes to be created by inheriting another scheme and selectively overriding the inherited scheme's properties as required. This flexibility enables cache schemes to be easily maintained and promotes cache scheme reuse. The `<scheme-ref>` element is used within a cache scheme definition and specifies the name of the cache scheme from which to inherit.

[Example 11–9](#) 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 11–9 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 11–9](#), 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 11–10](#) demonstrates multiple schemes reusing the same `LocalSizeLimited` base definition and overriding the `expiry-delay` property.

Example 11–10 Multiple Cache Schemes Using Scheme Inheritance

```
<distributed-scheme>
  <scheme-name>DistributedInMemoryCache</scheme-name>
  <service-name>DistributedCache</service-name>
  <backing-map-scheme>
    <local-scheme>
      <scheme-ref>LocalSizeLimited</scheme-ref>
    </local-scheme>
  </backing-map-scheme>
</distributed-scheme>

<replicated-scheme>
  <scheme-name>ReplicatedInMemoryCache</scheme-name>
  <service-name>ReplicatedCache</service-name>
  <backing-map-scheme>
    <local-scheme>
      <scheme-ref>LocalSizeLimited</scheme-ref>
      <expiry-delay>10m</expiry-delay>
    </local-scheme>
  </backing-map-scheme>
</replicated-scheme>

<local-scheme>
  <scheme-name>LocalSizeLimited</scheme-name>
  <eviction-policy>LRU</eviction-policy>
  <high-units>1000</high-units>
```

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

Using Cache Scheme Properties

Cache scheme properties modify cache behavior as required for a particular application. Each cache scheme type contains its own set of properties that are valid for the cache. Cache properties are set within a cache scheme definition using their respective elements. See [Appendix B, "Cache Configuration Elements,"](#) for a reference of all the properties that are supported for each cache scheme type.

Many cache properties use default values unless a different value is explicitly given within the cache scheme definition. The clustered caches (distributed, replicated and optimistic) use the default values as specified by their respective cache service definition. Cache services are defined in the operational deployment descriptor. While it is possible to change property values using an operational override file, cache properties are most often set within the cache scheme definition.

[Example 11-11](#) 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 in order 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 11-11 Setting Cache Properties

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

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>example</cache-name>
      <scheme-name>DistributedInMemoryCache</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <distributed-scheme>
      <scheme-name>DistributedInMemoryCache</scheme-name>
      <service-name>DistributedCache</service-name>
      <thread-count>4</thread-count>
      <request-timeout>60s</request-timeout>
      <backing-map-scheme>
        <local-scheme>
          <scheme-ref>LocalSizeLimited</scheme-ref>
        </local-scheme>
      </backing-map-scheme>
    </distributed-scheme>

    <local-scheme>
      <scheme-name>LocalSizeLimited</scheme-name>
      <eviction-policy>LRU</eviction-policy>
      <high-units>1000</high-units>
      <expiry-delay>1h</expiry-delay>
    </local-scheme>
  </caching-schemes>
</cache-config>
```

Using Parameter Macros

The cache configuration deployment descriptor supports parameter *macros* to minimize custom coding and enable specification of commonly used attributes when configuring class constructor parameters. The macros should be entered enclosed in curly braces as shown below, without any quotes or spaces.

[Table 11–2](#) describes the parameter macros that may be specified:

Table 11–2 Parameter Macros for Cache Configuration

<param-type>	<param-value>	Description
java.lang.String	{cache-name}	Used to pass the current cache name as a constructor parameter. For example: <pre><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></pre>
java.lang.ClassLoader	{class-loader}	Used to pass the current classloader as a constructor parameter. For example: <pre><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></pre>

Table 11-2 (Cont.) Parameter Macros for Cache Configuration

<param-type>	<param-value>	Description
com.tangosol.net. BackingMapManager Context	{manager-context}	Used to pass the current BackingMapManagerContext object as a constructor parameter. For example: <pre> <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> </pre>
{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: <pre> <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> </pre>

Table 11–2 (Cont.) Parameter Macros for Cache Configuration

<param-type>	<param-value>	Description
{cache-ref}	cache name	<p>Used to obtain a NamedCache reference for the specified cache name. Consider the following configuration example:</p> <pre> <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> </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> </pre> <p>The <code>CacheFactory.getCache("london-test")</code> call would result in a standard partitioned cache reference. Conversely, the <code>CacheFactory.getCache("boston-test")</code> call would resolve the value of the <code>delegate-cache-name</code> parameter to <code>london-test</code> and would construct an instance of the <code>WrapperNamedCache</code> delegating to the <code>NamedCache</code> returned by the <code>CacheFactory.getCache("london-test")</code> call.</p>

Implementing Storage and Backing Maps

This chapter provides information on coherence storage using backing maps. The following sections are included in this chapter:

- [Cache Layers](#)
- [Operations](#)
- [Capacity Planning](#)
- [Partitioned Backing Maps](#)

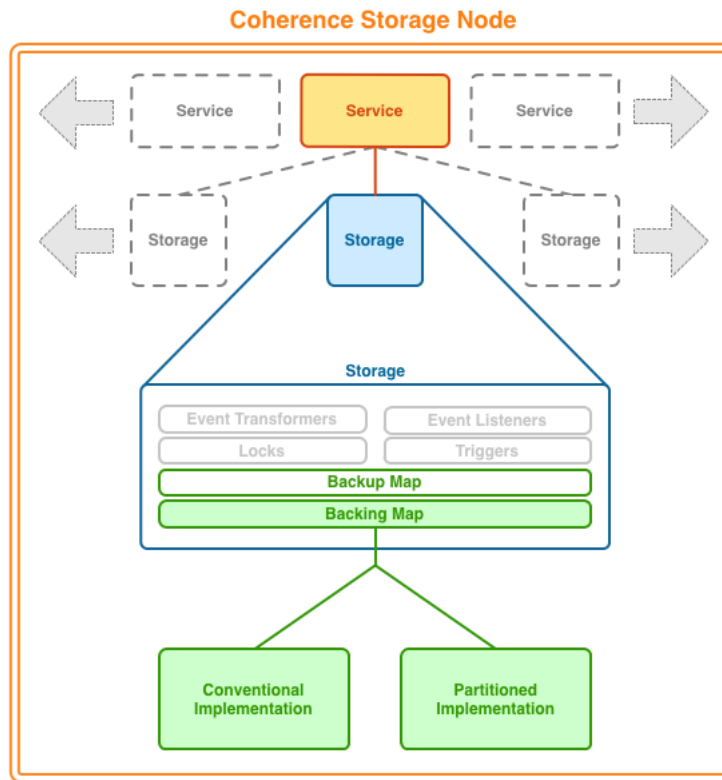
Cache Layers

Partitioned (Distributed) cache service in Coherence has three distinct layers:

- **Client View** – The client view represents a virtual layer that provides access to the underlying partitioned data. Access to this tier is provided using the `NamedCache` interface. In this layer you can also create synthetic data structures such as `NearCache` or `ContinuousQueryCache`.
- **Storage Manager** – The storage manager is the server-side tier that is responsible for processing cache-related requests from the client tier. It manages the data structures that hold the actual cache data (primary and backup copies) and information about locks, event listeners, map triggers etc.
- **Backing Map** – The Backing Map is the server-side data structure that holds actual data.

Coherence allows users to configure a number of out-of-the-box backing map implementations as well as custom ones. Basically, the only constraint that all these Map implementation have to be aware of, is the understanding that the Storage Manager provides all keys and values in internal (Binary) format. To deal with conversions of that internal data to and from an Object format, the Storage Manager can supply Backing Map implementations with a `BackingMapManagerContext` reference.

[Figure 12-1](#) shows a conceptual view of backing maps.

Figure 12–1 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.

Typically, Coherence uses one of the following local storage implementations:

- **Safe HashMap:** This is the default lossless implementation. A lossless implementation is one, like Java's `Hashtable` class, that is neither size-limited nor auto-expiring. In other words, it is an implementation that never evicts ("loses") cache items on its own. This particular `HashMap` implementation is optimized for extremely high thread-level concurrency. (For the default implementation, use class `com.tangosol.util.SafeHashMap`; when an implementation is required that provides cache events, use `com.tangosol.util.ObservableHashMap`. These implementations are thread-safe.)
- **Local Cache:** This is the default size-limiting and/or auto-expiring implementation. The local cache is covered in more detail below, but the primary points to remember about it are that it can limit the size of the cache, and it can automatically expire cache items after a certain period. (For the default implementation, use `com.tangosol.net.cache.LocalCache`; this implementation is thread safe and supports cache events, `com.tangosol.net.CacheLoader`, `CacheStore` and configurable/pluggable eviction policies.)

- **Read/Write Backing Map:** This is the default backing map implementation for caches that load from a database on a cache miss. It can be configured as a read-only cache (consumer model) or as either a write-through or a write-behind cache (for the consumer/producer model). The write-through and write-behind modes are intended only for use with the distributed cache service. If used with a near cache and the near cache must be kept synchronous with the distributed cache, it is possible to combine the use of this backing map with a Seppuku-based near cache (for near cache invalidation purposes); however, given these requirements, it is suggested that the versioned implementation be used. (For the default implementation, use class `com.tangosol.net.cache.ReadWriteBackingMap`.)
- **Versioned Backing Map:** This is an optimized version of the read/write backing map that optimizes its handling of the data by utilizing a data versioning technique. For example, to invalidate near caches, it simply provides a version change notification, and to determine whether cached data must be written back to the database, it can compare the persistent (database) version information with the transient (cached) version information. The versioned implementation can provide very balanced performance in large scale clusters, both for read-intensive and write-intensive data. (For the default implementation, use class `com.tangosol.net.cache.VersionedBackingMap`; with this backing map, you can optionally use the `com.tangosol.net.cache.VersionedNearCache` as a near cache implementation.)
- **Binary Map (Java NIO):** This is a backing map implementation that can store its information in memory but outside of the Java heap, or even in memory-mapped files, which means that it does not affect the Java heap size and the related JVM garbage-collection performance that can be responsible for application pauses. This implementation is also available for distributed cache backups, which is particularly useful for read-mostly and read-only caches that require backup for high availability purposes, because it means that the backup does not affect the Java heap size yet it is immediately available in case of failover.
- **Serialization Map:** This is a backing map implementation that translates its data to a form that can be stored on disk, referred to as a serialized form. It requires a separate `com.tangosol.io.BinaryStore` object into which it stores the serialized form of the data; usually, this is the built-in LH disk store implementation, but the Serialization Map supports any custom implementation of `BinaryStore`. (For the default implementation of Serialization Map, use `com.tangosol.net.cache.SerializationMap`.)
- **Serialization Cache:** This is an extension of the `SerializationMap` that supports an LRU eviction policy. This can be used to limit the size of disk files, for example. (For the default implementation of Serialization Cache, use `com.tangosol.net.cache.SerializationCache`.)
- **Overflow Map:** An overflow map doesn't actually provide storage, but it deserves mention in this section because it can tie together two local storage implementations so that when the first one fills up, it will overflow into the second. (For the default implementation of `OverflowMap`, use `com.tangosol.net.cache.OverflowMap`.)

Operations

There are number of operation types performed against the Backing Map:

- Natural access and update operations caused by the application usage. For example, `NamedCache.get()` call naturally causes a `Map.get()` call on a

corresponding Backing Map; the `NamedCache.invoke()` call may cause a sequence of `Map.get()` followed by the `Map.put()`; the `NamedCache.keySet(filter)` call may cause an `Map.entrySet().iterator()` loop, and so on.

- Remove operations caused by the time-based expiry or the size-based eviction. For example, a `NamedCache.get()` or `NamedCache.size()` call from the client tier could cause a `Map.remove()` call due to an entry expiry timeout; or `NamedCache.put()` call causing a number of `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, a number of entries could be inserted or removed from the backing map.

Capacity Planning

Depending on the actual implementation, the Backing Map stores the cache data in one of the following ways:

- on-heap memory
- off-heap memory
- disk (memory-mapped files or in-process DB)
- combination of any of the above

Keeping data in memory naturally provides dramatically smaller access and update latencies and is most commonly used.

More often than not, applications need to ensure that the total amount of data placed into the data grid does not exceed some predetermined amount of memory. It could be done either directly by the application tier logic or automatically using size- or expiry-based eviction. Quite naturally, the total amount of data held in a Coherence cache is equal to the sum of data volume in all corresponding backing maps (one per each cluster node that runs the corresponding partitioned cache service in a storage enabled mode).

Consider following cache configuration excerpts:

```
<backing-map-scheme>
  <local-scheme/>
</backing-map-scheme>
```

The backing map above is an instance of `com.tangosol.net.cache.LocalCache` and does not have any pre-determined size constraints and has to be controlled explicitly. Failure to do so could cause the JVM to go out-of-memory.

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

This backing map above is also a `com.tangosol.net.cache.LocalCache` and has a capacity limit of 100MB. As the total amount of data held by this backing map exceeds that high watermark, some entries will be removed from the backing map, bringing the volume down to the low watermark value (`<low-units>` configuration element, which defaults to 75% of the `<high-units>`). The choice of the removed entries will be based on the LRU (Least Recently Used) eviction policy. Other options are LFU (Least Frequently Used) and Hybrid (a combination of the LRU and LFU). The value of `<high-units>` is limited to 2GB. To overcome that limitation (but maintain backward compatibility) Coherence uses the `<unit-factor>` element. For example, the `<high-units>` value of 8192 with a `<unit-factor>` of 1048576 will result in a high watermark value of 8GB.

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

The backing map above automatically evicts any entries that have not been updated for more than an hour. Note, that such an eviction is a "lazy" one and can happen any time after an hour since the last update happens; the only guarantee Coherence provides is that entries more than one hour old are not returned to a caller.

```
<backing-map-scheme>
  <external-scheme>
    <high-units>100</high-units>
    <unit-calculator>BINARY</unit-calculator>
    <unit-factor>1048576</unit-factor>
    <nio-memory-manager>
      <initial-size>1MB</initial-size>
      <maximum-size>100MB</maximum-size>
    </nio-memory-manager>
  </external-scheme>
</backing-map-scheme>
```

This backing map above is an instance of `com.tangosol.net.cache.SerializationCache` which stores values in the extended (nio) memory and has a capacity limit of 100MB (100*1048576). Configure a backup storage for this cache being off-heap (or file-mapped):

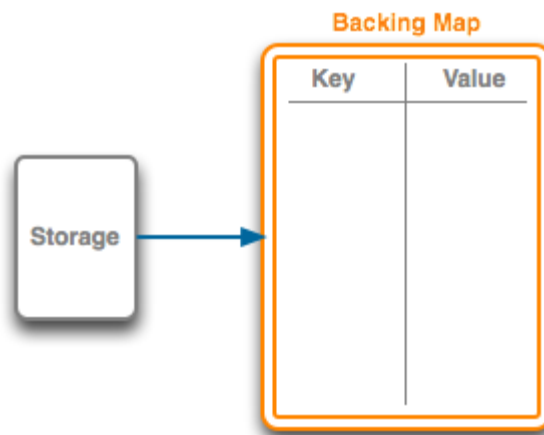
```
<backup-storage>
  <type>off-heap</type>
  <initial-size>1MB</initial-size>
  <maximum-size>100MB</maximum-size>
</backup-storage>
```

Partitioned Backing Maps

Prior to Coherence 3.5, a backing map would contain entries for all partitions owned by the corresponding node. (During partition transfer, it could also hold "in flight" entries that from the clients' perspective are temporarily not owned by anyone).

Figure 12-2 shows a conceptual view of the conventional backing map implementation.

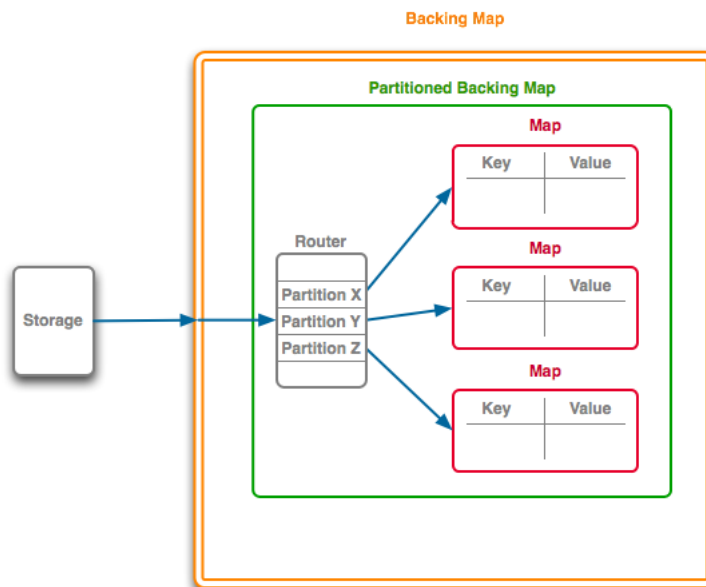
Figure 12–2 Conventional Backing Map Implementation



Coherence 3.5 introduced a concept of partitioned backing map, which is basically a multiplexer of actual Map implementations, each of which would contain only entries that belong to the same partition.

Figure 12–3 shows a conceptual view of the partitioned backing map implementation.

Figure 12–3 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>
    <high-units>8192</high-units>
    <unit-calculator>BINARY</unit-calculator>
    <unit-factor>1048576</unit-factor>
    <nio-memory-manager>
      <initial-size>1MB</initial-size>
      <maximum-size>50MB</maximum-size>
```



```
    </nio-memory-manager>  
  </external-scheme>  
</backing-map-scheme>
```

This backing map is an instance of `com.tangosol.net.partition.PartitionSplittingBackingMap`, with individual partition holding maps being instances of `com.tangosol.net.cache.SerializationCache` that each store values in the extended (nio) memory. The individual nio buffers have a limit of 50MB, while the backing map as whole has a capacity limit of 8GB (8192*1048576). Again, you would need to configure a backup storage for this cache being off-heap or file-mapped.

Read-Through, Write-Through, Write-Behind, and Refresh-Ahead Caching

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" will be used to describe any back-end data source. Effective caches must support both intensive read-only and read/write operations, and in the case of read/write operations, the cache and database must be kept fully synchronized. To accomplish this, Coherence supports **Read-Through, Write-Through, Refresh-Ahead and Write-Behind** caching.

Note: For use with Partitioned (Distributed) and Near cache topologies: 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.

Pluggable Cache Store

A `CacheStore` is an application-specific adapter used to connect a cache to a underlying data source. The `CacheStore` implementation accesses the data source by using a data access mechanism (for example, *Hibernate*, *Toplink Essentials*, *JPA*, application-specific JDBC calls, another application, mainframe, another cache, and so on). The `CacheStore` understands how to build a Java object using data retrieved from the data source, map and write an object to the data source, and erase an object from the data source.

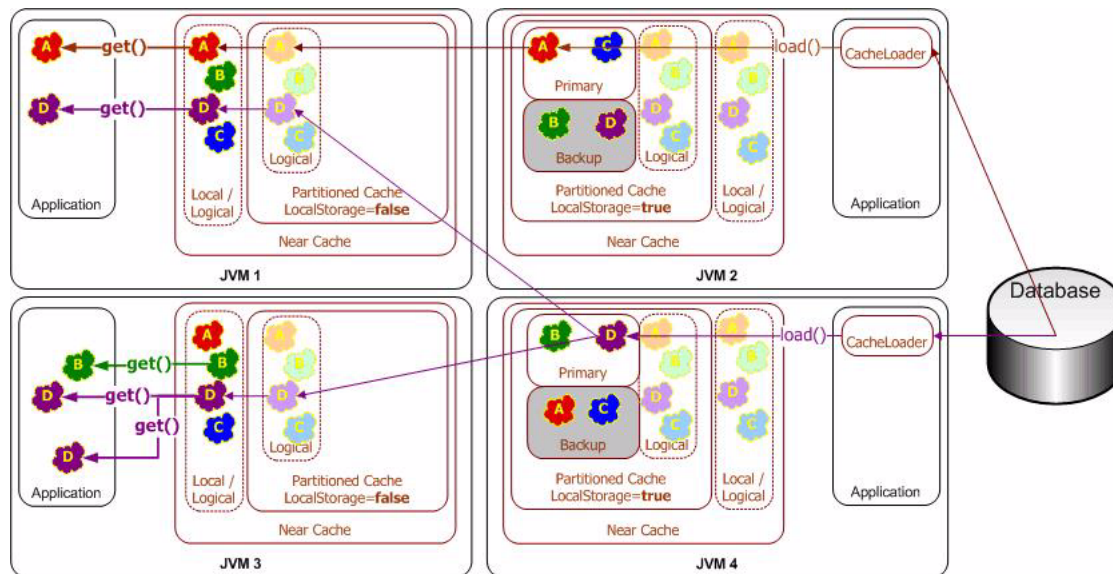
Both the data source connection strategy and the data source-to-application-object mapping information are specific to the data source schema, application class layout, and operating environment. Therefore, this mapping information must be provided by the application developer in the form of a `CacheStore` implementation. See "[Creating a CacheStore Implementation](#)" for more information.

Read-Through Caching

When an application asks the cache for an entry, for example the key `x`, and `x` is not already in the cache, Coherence will automatically delegate to the `CacheStore` and ask it to load `x` from the underlying data source. If `x` exists in the data source, the `CacheStore` will load it, return it to Coherence, then Coherence will place it in the cache for future use and finally will return `x` to the application code that requested it. This is called **Read-Through** caching. Refresh-Ahead Cache functionality may further

improve read performance (by reducing perceived latency). See ["Refresh-Ahead Caching"](#) for more information.

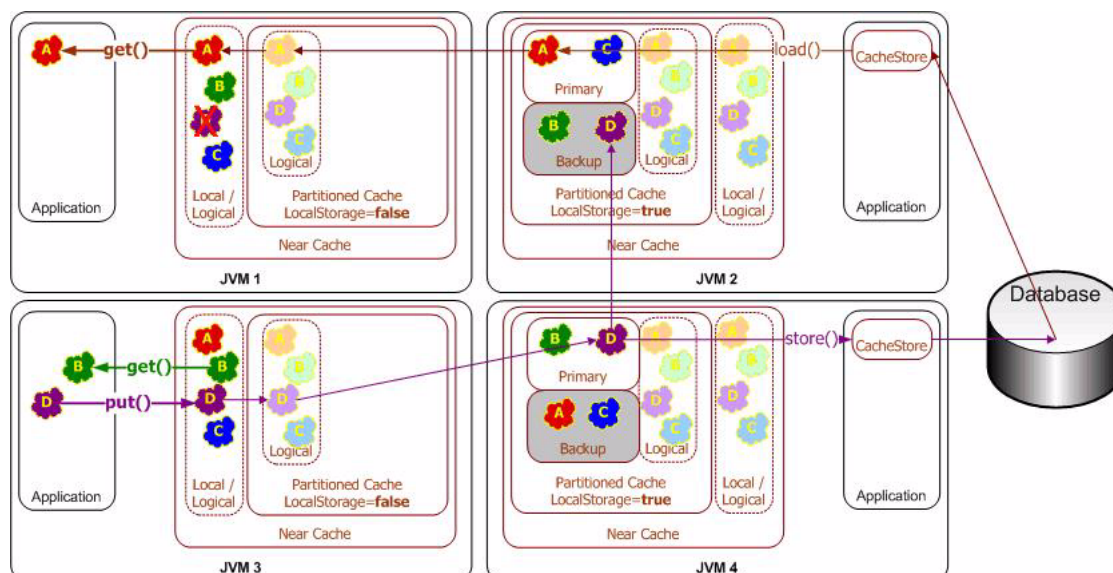
Figure 13–1 Read-Through Caching



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 will not complete (that is, the `put` will not return) until Coherence has gone through the `CacheStore` and successfully stored the data to the underlying data source. This does not improve write performance at all, since you are still dealing with the latency of the write to the data source. Improving the write performance is the purpose for the *Write-Behind Cache* functionality. See ["Write-Behind Caching"](#) for more information.

Figure 13–2 Write-Through Caching

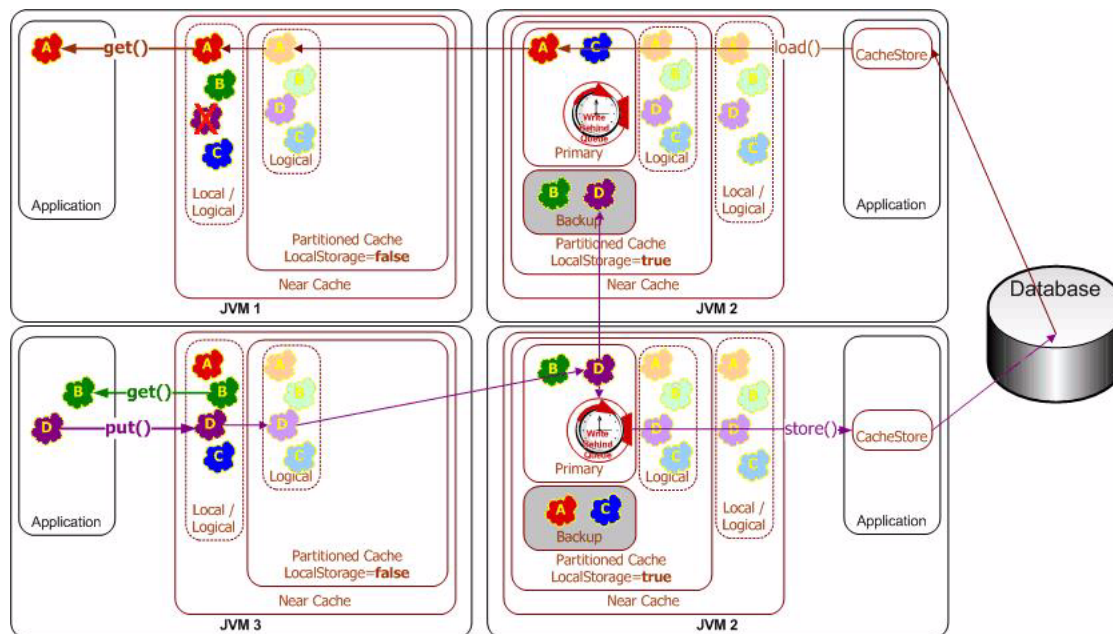


Write-Behind Caching

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 isn't there already; otherwise, it is replaced), and after the specified write-behind delay Coherence will call 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 will never lag 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 will result 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 13–3 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 will (in most cases) complete 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 means 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 will 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.

In earlier releases of Coherence, rebalancing (due to failover/failback) would result in the re-queuing of all cache entries in the affected cache partitions (typically 1/N where N is the number of servers in the cluster). While the nature of write-behind (asynchronous queuing and load-averaging) minimized the direct impact of this, for some workloads it could be problematic. Best practice for affected applications was to use `com.tangosol.net.cache.VersionedBackingMap`. As of Coherence 3.2, backups are notified when a modified entry has been successfully written to the data

source, avoiding the need for this strategy. If possible, applications should deprecate use of the `VersionedBackingMap` if it was used only for its write-queuing behavior.

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 will 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 will perform 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 will perform a *synchronous* read from the cache store to refresh its value. However, if a request is performed for an entry that is more than 30 but less than 60 seconds old, the current value in the cache is returned and Coherence schedules an *asynchronous* reload from the cache store.

Refresh-ahead is especially useful if objects are being accessed by a large number of users. Values remain fresh in the cache and the latency that could result from excessive reloads from the cache store is avoided.

The value of the refresh-ahead factor is specified by the `<refresh-ahead-factor>` subelement of the `<read-write-backing-map-scheme>` element in the `coherence-cache-config.xml` file. Refresh-ahead assumes that you have also set an expiration time (`<expiry-delay>`) for entries in the cache.

The XML code fragment in [Example 13–1](#) configures a refresh-ahead factor of 0.5 and an expiration time of 20 seconds for entries in the local cache. This means that if an entry is accessed within 10 seconds of its expiration time, it will be scheduled for an asynchronous reload from the cache store.

Example 13–1 Cache Configuration Specifying a Refresh-Ahead Factor

```
<cache-config>
...

<distributed-scheme>
  <scheme-name>categories-cache-all-scheme</scheme-name>
  <service-name>DistributedCache</service-name>
  <backing-map-scheme>

  <!--
  Read-write-backing-map caching 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>
...

<!--
    Backing map scheme definition used by all the caches that require
    size limitation and/or expiry eviction policies.
-->
<local-scheme>
    <scheme-name>categories-eviction</scheme-name>
    <expiry-delay>20s</expiry-delay>
</local-scheme>
...
</cache-config>
```

Selecting a Cache Strategy

This section compares and contrasts the benefits of several caching strategies.

- [Read-Through/Write-Through versus Cache-Aside](#)
- [Refresh-Ahead versus Read-Through](#)
- [Write-Behind versus Write-Through](#)

Read-Through/Write-Through versus Cache-Aside

There are two common approaches to the cache-aside pattern in a clustered environment. One involves checking for a cache miss, then querying the database, populating the cache, and continuing application processing. This can result in multiple database visits if different application threads perform this processing at the same time. Alternatively, applications may perform double-checked locking (which works since the check is atomic with respect to the cache entry). This, however, results in a substantial amount of overhead on a cache miss or a database update (a clustered lock, additional read, and clustered unlock - up to 10 additional network hops, or 6-8ms on a typical gigabit Ethernet connection, plus additional processing overhead and an increase in the "lock duration" for a cache entry).

By using inline caching, the entry is locked only for the 2 network hops (while the data is copied to the backup server for fault-tolerance). Additionally, the locks are maintained locally on the partition owner. Furthermore, application code is fully managed on the cache server, meaning that only a controlled subset of nodes will directly access 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 will offer reduced latency and no added overhead. The higher the rate of inaccurate prediction, the greater the impact will be on throughput (as more unnecessary requests will be sent to the database) -

potentially even having a negative impact on latency should the database start to fall behind on request processing.

Write-Behind versus Write-Through

If the requirements for write-behind caching can be satisfied, write-behind caching may deliver considerably higher throughput and reduced latency compared to write-through caching. Additionally write-behind caching lowers the load on the database (fewer writes), and on the cache server (reduced cache value deserialization).

Creating a CacheStore Implementation

CacheStore implementations are pluggable, and depending on the cache's usage of the data source you will need to implement one of two interfaces:

- CacheLoader for read-only caches
- CacheStore which extends CacheLoader to support read/write caches

These interfaces are located in the `com.tangosol.net.cache` package. The CacheLoader interface has two main methods: `load(Object key)` and `loadAll(Collection keys)`, and the CacheStore interface adds the methods `store(Object key, Object value)`, `storeAll(Map mapEntries)`, `erase(Object key)`, and `eraseAll(Collection colKeys)`.

See ["Sample CacheStore"](#) on page 13-8 and ["Sample Controllable CacheStore"](#) on page 13-14 for example CacheStore implementations.

Plugging in a CacheStore Implementation

To plug in a CacheStore module, specify the CacheStore implementation class name within the `distributed-scheme`, `backing-map-scheme`, `cachestore-scheme`, or `read-write-backing-map-scheme`, cache configuration element.

The `read-write-backing-map-scheme` configures a `com.tangosol.net.cache.ReadWriteBackingMap`. This backing map is composed of two key elements: an internal map that actually caches the data (see `internal-cache-scheme`), and a CacheStore module that interacts with the database (see `cachestore-scheme`).

[Example 13-2](#) illustrates a cache configuration that specifies a CacheStore module. The `<init-params>` element contains an ordered list of parameters that will be passed into the CacheStore constructor. The `{cache-name}` configuration macro is used to pass the cache name into the CacheStore implementation, allowing it to be mapped to a database table. For a complete list of available macros, see ["Using Parameter Macros"](#) on page 11-12.

For more detailed information on configuring write-behind and refresh-ahead, see the `read-write-backing-map-scheme`, taking note of the `write-batch-factor`, `refresh-ahead-factor`, `write-requeue-threshold`, and `rollback-cachestore-failures` elements.

Example 13-2 A Cache Configuration with a Cachestore Module

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

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

```
<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>com.company.dto.*</cache-name>
      <scheme-name>distributed-rwbm</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <distributed-scheme>
      <scheme-name>distributed-rwbm</scheme-name>
      <backing-map-scheme>
        <read-write-backing-map-scheme>

          <internal-cache-scheme>
            <local-scheme/>
          </internal-cache-scheme>

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

Note: Thread Count: The use of a CacheStore module will substantially increase the consumption of cache service threads (even the fastest database select is orders of magnitude slower than updating an in-memory structure). Consequently, the cache service thread count will need to be increased (typically in the range 10-100). The most noticeable symptom of an insufficient thread pool is increased latency for cache requests (without corresponding behavior in the backing database).

Sample CacheStore

This section provides a very basic implementation of the `com.tangosol.net.cache.CacheStore` interface. The implementation in [Example 13-3](#) uses a single database connection by using JDBC, and does not use bulk operations. A complete implementation would use a connection pool, and, if write-behind is used, implement `CacheStore.storeAll()` for bulk JDBC inserts and updates. "[Cache of a Database](#)" on page 15-4 provides an example of a database cache configuration.

Tip: Save processing effort by bulk loading the cache. The following example use the `put` method to write values to the cache store. Often, performing bulk loads with the `putAll` method will result in a savings in processing effort and network traffic. For more information on bulk loading, see [Chapter 18, "Pre-Loading the Cache."](#)

Example 13–3 Implementation of the CacheStore Interface

```
package com.tangosol.examples.coherence;

import com.tangosol.net.cache.CacheStore;
import com.tangosol.util.Base;

import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import java.util.Collection;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

/**
 * An example implementation of CacheStore
 * interface.
 *
 * @author erm 2003.05.01
 */
public class DBCacheStore
    extends Base
    implements CacheStore
{
    // ----- constructors -----
    /**
     * Constructs DBCacheStore for a given database table.
     *
     * @param sTableName the db table name
     */
    public DBCacheStore(String sTableName)
    {
        m_sTableName = sTableName;
        configureConnection();
    }

    /**
     * Set up the DB connection.
     */
    protected void configureConnection()
    {
        try
        {
            Class.forName("org.gjt.mm.mysql.Driver");
            m_con = DriverManager.getConnection(DB_URL, DB_USERNAME,
DB_PASSWORD);
            m_con.setAutoCommit(true);
        }
    }
}
```

```
        }
        catch (Exception e)
        {
            throw ensureRuntimeException(e, "Connection failed");
        }
    }

    // ---- accessors -----

    /**
     * Obtain the name of the table this CacheStore is persisting to.
     *
     * @return the name of the table this CacheStore is persisting to
     */
    public String getTableName()
    {
        return m_sTableName;
    }

    /**
     * Obtain the connection being used to connect to the database.
     *
     * @return the connection used to connect to the database
     */
    public Connection getConnection()
    {
        return m_con;
    }

    // ----- CacheStore Interface -----

    /**
     * Return the value associated with the specified key, or null if the
     * key does not have an associated value in the underlying store.
     *
     * @param oKey key whose associated value is to be returned
     *
     * @return the value associated with the specified key, or
     *         <tt>null</tt> if no value is available for that key
     */
    public Object load(Object oKey)
    {
        Object oValue = null;
        Connection con = getConnection();
        String sSQL = "SELECT id, value FROM " + getTableName()
            + " WHERE id = ?";

        try
        {
            PreparedStatement stmt = con.prepareStatement(sSQL);

            stmt.setString(1, String.valueOf(oKey));

            ResultSet rslt = stmt.executeQuery();
            if (rslt.next())
            {
                oValue = rslt.getString(2);
                if (rslt.next())
                {

```

```

        throw new SQLException("Not a unique key: " + oKey);
    }
    }
    stmt.close();
}
catch (SQLException e)
{
    throw ensureRuntimeException(e, "Load failed: key=" + oKey);
}
return oValue;
}

/**
 * Store the specified value under the specific key in the underlying
 * store. This method is intended to support both key/value creation
 * and value update for a specific key.
 *
 * @param oKey    key to store the value under
 * @param oValue  value to be stored
 *
 * @throws UnsupportedOperationException if this implementation or the
 *         underlying store is read-only
 */
public void store(Object oKey, Object oValue)
{
    Connection con    = getConnection();
    String      sTable = getTableName();
    String      sSQL;

    // the following is very inefficient; it is recommended to use DB
    // specific functionality that is, REPLACE for MySQL or MERGE for Oracle
    if (load(oKey) != null)
    {
        // key exists - update
        sSQL = "UPDATE " + sTable + " SET value = ? where id = ?";
    }
    else
    {
        // new key - insert
        sSQL = "INSERT INTO " + sTable + " (value, id) VALUES (?,?)";
    }
    try
    {
        PreparedStatement stmt = con.prepareStatement(sSQL);
        int i = 0;
        stmt.setString(++i, String.valueOf(oValue));
        stmt.setString(++i, String.valueOf(oKey));
        stmt.executeUpdate();
        stmt.close();
    }
    catch (SQLException e)
    {
        throw ensureRuntimeException(e, "Store failed: key=" + oKey);
    }
}

/**
 * Remove the specified key from the underlying store if present.
 *
 * @param oKey key whose mapping is to be removed from the map

```

```
*
* @throws UnsupportedOperationException if this implementation or the
*       underlying store is read-only
*/
public void erase(Object oKey)
{
    Connection con = getConnection();
    String      sSQL = "DELETE FROM " + getTableName() + " WHERE id=?";
    try
    {
        PreparedStatement stmt = con.prepareStatement(sSQL);

        stmt.setString(1, String.valueOf(oKey));
        stmt.executeUpdate();
        stmt.close();
    }
    catch (SQLException e)
    {
        throw ensureRuntimeException(e, "Erase failed: key=" + oKey);
    }
}

/**
 * Remove the specified keys from the underlying store if present.
 *
 * @param colKeys keys whose mappings are being removed from the cache
 *
 * @throws UnsupportedOperationException if this implementation or the
 *       underlying store is read-only
 */
public void eraseAll(Collection colKeys)
{
    throw new UnsupportedOperationException();
}

/**
 * Return the values associated with each the specified keys in the
 * passed collection. If a key does not have an associated value in
 * the underlying store, then the return map will not have an entry
 * for that key.
 *
 * @param colKeys a collection of keys to load
 *
 * @return a Map of keys to associated values for the specified keys
 */
public Map loadAll(Collection colKeys)
{
    throw new UnsupportedOperationException();
}

/**
 * Store the specified values under the specified keys in the underlying
 * store. This method is intended to support both key/value creation
 * and value update for the specified keys.
 *
 * @param mapEntries a Map of any number of keys and values to store
 *
 * @throws UnsupportedOperationException if this implementation or the
 *       underlying store is read-only
 */
```

```

        public void storeAll(Map mapEntries)
        {
            throw new UnsupportedOperationException();
        }

/**
 * Iterate all keys in the underlying store.
 *
 * @return a read-only iterator of the keys in the underlying store
 */
public Iterator keys()
{
    Connection con = getConnection();
    String      sSQL = "SELECT id FROM " + getTableName();
    List        list = new LinkedList();

    try
    {
        PreparedStatement stmt = con.prepareStatement(sSQL);
        ResultSet         rslt = stmt.executeQuery();
        while (rslt.next())
        {
            Object oKey = rslt.getString(1);
            list.add(oKey);
        }
        stmt.close();
    }
    catch (SQLException e)
    {
        throw ensureRuntimeException(e, "Iterator failed");
    }

    return list.iterator();
}

// ----- data members -----

/**
 * The connection.
 */
protected Connection m_con;

/**
 * The db table name.
 */
protected String m_sTableName;

/**
 * Driver class name.
 */
private static final String DB_DRIVER = "org.gjt.mm.mysql.Driver";

/**
 * Connection URL.
 */
private static final String DB_URL =
"jdbc:mysql://localhost:3306/CacheStore";

/**

```

```
    * User name.
    */
    private static final String DB_USERNAME = "root";

    /**
    * Password.
    */
    private static final String DB_PASSWORD = null;
}
```

Sample Controllable CacheStore

This section illustrates the implementation of a controllable cache store. In this scenario, the application can control when updated values in the cache are written to the data store. The most common use case for this scenario is during the initial population of the cache from the data store at startup. At startup, there is no need to write values in the cache back to the data store. Any attempt to do so would be a waste of resources.

The `Main.java` file in [Example 13-4](#) illustrates two different approaches to interacting with a controllable cache store:

- Use a controllable cache (note that it must be on a different service) to enable or disable the cache store. This is illustrated by the `ControllableCacheStore1` class.
- Use the `CacheStoreAware` interface to indicate that objects added to the cache do not need to be stored. This is illustrated by the `ControllableCacheStore2` class.

Both `ControllableCacheStore1` and `ControllableCacheStore2` extend the `com.tangosol.net.cache.AbstractCacheStore` class. This helper class provides unoptimized implementations of the `storeAll` and `eraseAll` operations.

The `CacheStoreAware.java` file is an interface which can be used to indicate that an object added to the cache should not be stored in the database.

See ["Cache of a Database"](#) on page 15-4 for a sample cache configurations.

[Example 13-4](#) provides a listing of the `Main.java` interface.

Example 13-4 *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
{
    /**
    * CacheStore implementation which is controlled by a control cache
    */
    public static class ControllableCacheStore1 extends AbstractCacheStore
    {
        public static final String CONTROL_CACHE = "cachestorecontrol";
    }
}
```



```

String m_sName;

public static void enable(String sName)
{
    CacheFactory.getCache(CONTROL_CACHE).put(sName, Boolean.TRUE);
}

public static void disable(String sName)
{
    CacheFactory.getCache(CONTROL_CACHE).put(sName, Boolean.FALSE);
}

public void store(Object oKey, Object oValue)
{
    Boolean isEnabled = (Boolean) CacheFactory.getCache(CONTROL_
CACHE).get(m_sName);
    if (isEnabled != null && isEnabled.booleanValue())
    {
        log("controllablecachestore1: enabled " + oKey + " = " + oValue);
    }
    else
    {
        log("controllablecachestore1: disabled " + oKey + " = " + oValue);
    }
}

public Object load(Object oKey)
{
    log("controllablecachestore1: load:" + oKey);
    return new MyValue1(oKey);
}

public ControllableCacheStore1(String sName)
{
    m_sName = sName;
}

}

/**
 * CacheStore implementation which is controlled by values
 * implementing the CacheStoreAware interface
 */
public static class ControllableCacheStore2 extends AbstractCacheStore
{

    public void store(Object oKey, Object oValue)
    {
        boolean isEnabled = oValue instanceof CacheStoreAware ?
!((CacheStoreAware) oValue).isSkipStore() : true;
        if (isEnabled)
        {
            log("controllablecachestore2: enabled " + oKey + " = " + oValue);
        }
        else
        {
            log("controllablecachestore2: disabled " + oKey + " = " + oValue);
        }
    }
}

```

```
        public Object load(Object oKey)
        {
            log("controllablecachestore2: load:" + oKey);
            return new MyValue2(oKey);
        }

    }

    public static class MyValue1 implements Serializable
    {
        String m_sValue;

        public String getValue()
        {
            return m_sValue;
        }

        public String toString()
        {
            return "MyValue1[" + getValue() + "]";
        }

        public MyValue1(Object obj)
        {
            m_sValue = "value:" + obj;
        }
    }

    public static class MyValue2 extends MyValue1 implements CacheStoreAware
    {
        boolean m_isSkipStore = false;

        public boolean isSkipStore()
        {
            return m_isSkipStore;
        }

        public void skipStore()
        {
            m_isSkipStore = true;
        }

        public String toString()
        {
            return "MyValue2[" + getValue() + "]";
        }

        public MyValue2(Object obj)
        {
            super(obj);
        }
    }

    public static void main(String[] args)
    {
        try
        {
            // example 1
```

```

NamedCache cache1 = CacheFactory.getCache("cache1");

// disable cachestore
ControllableCacheStore1.disable("cache1");
for(int i = 0; i < 5; i++)
{
    cache1.put(new Integer(i), new MyValue1(new Date()));
}

// enable cachestore
ControllableCacheStore1.enable("cache1");
for(int i = 0; i < 5; i++)
{
    cache1.put(new Integer(i), new MyValue1(new Date()));
}

// example 2

NamedCache cache2 = CacheFactory.getCache("cache2");

// add some values with cachestore disabled
for(int i = 0; i < 5; i++)
{
    MyValue2 value = new MyValue2(new Date());
    value.skipStore();
    cache2.put(new Integer(i), value);
}

// add some values with cachestore enabled
for(int i = 0; i < 5; i++)
{
    cache2.put(new Integer(i), new MyValue2(new Date()));
}

}
catch(Throwable oops)
{
    err(oops);
}
finally
{
    CacheFactory.shutdown();
}
}
}

```

[Example 13-5](#) provides a listing of the `CacheStoreAware.java` interface.

Example 13-5 *CacheStoreAware.java* interface

```

public interface CacheStoreAware
{
    public boolean isSkipStore();
}

```

Implementation Considerations

Please keep the following in mind when implementing a `CacheStore`.

Idempotency

All `CacheStore` operations should be designed to be idempotent (that is, repeatable without unwanted side-effects). For write-through and write-behind caches, this allows Coherence to provide low-cost fault-tolerance for partial updates by re-trying the database portion of a cache update during failover processing. For write-behind caching, idempotency also allows Coherence to combine multiple cache updates into a single `CacheStore` invocation without affecting data integrity.

Applications that have a requirement for write-behind caching but which must avoid write-combining (for example, for auditing reasons), should create a "versioned" cache key (for example, by combining the natural primary key with a sequence id).

Write-Through Limitations

Coherence does not support two-phase `CacheStore` operations across multiple `CacheStore` instances. In other words, if two cache entries are updated, triggering calls to `CacheStore` modules sitting on separate cache servers, it is possible for one database update to succeed and for the other to fail. In this case, it may be preferable to use a cache-aside architecture (updating the cache and database as two separate components of a single transaction) with the application server transaction manager. In many cases it is possible to design the database schema to prevent logical commit failures (but obviously not server failures). Write-behind caching avoids this issue as "puts" are not affected by database behavior (and the underlying issues will have been addressed earlier in the design process). This limitation will be addressed in an upcoming release of Coherence.

Cache Queries

Cache queries only operate on data stored in the cache and will not trigger the `CacheStore` to load any missing (or potentially missing) data. Therefore, applications that query `CacheStore`-backed caches should ensure that all necessary data required for the queries has been pre-loaded. For efficiency, most bulk load operations should be done at application startup by streaming the data set directly from the database into the cache (batching blocks of data into the cache by using `NamedCache.putAll()`). The loader process will need to use a "Controllable Caches" pattern to disable circular updates back to the database. The `CacheStore` may be controlled by using an Invocation service (sending agents across the cluster to modify a local flag in each JVM) or by setting the value in a Replicated cache (a different cache service) and reading it in every `CacheStore` method invocation (minimal overhead compared to the typical database operation). A custom MBean can also be used, a simple task with Coherence's clustered JMX facilities.

Re-entrant Calls

The `CacheStore` implementation must not call back into the hosting cache service. This includes ORM solutions that may internally reference Coherence cache services. Note that calling into another cache service instance is allowed, though care should be taken to avoid deeply nested calls (as each call will "consume" a cache service thread and could result in deadlock if a cache service thread pool is exhausted).

Cache Server Classpath

The classes for cache entries (also known as Value Objects, Data Transfer Objects, and so on) must be in the cache server classpath (as the cache server must serialize-deserialize cache entries to interact with the `CacheStore` module).

CacheStore Collection Operations

The `CacheStore.storeAll` method is most likely to be used if the cache is configured as write-behind and the `<write-batch-factor>` is configured. The `CacheLoader.loadAll` method is also used by Coherence. For similar reasons, its first use will likely require 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).

Serialization Paged Cache

Oracle Coherence provides explicit support for efficient caching of huge amounts of automatically-expiring data using potentially high-latency storage mechanisms such as disk files. The benefits include supporting much larger data sets than can be managed in memory, while retaining an efficient expiry mechanism for timing out the management (and automatically freeing the resources related to the management) of that data. Optimal usage scenarios include the ability to store many large objects, XML documents or content that will be rarely accessed, or whose accesses will tolerate a higher latency if the cached data has been paged to disk. See [Chapter 12, "Implementing Storage and Backing Maps."](#)

Understanding Serialization Paged Cache

This feature is known as a Serialization Paged Cache:

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

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

Configuring Serialization Paged Cache

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

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

Optimizing a Partitioned Cache Service

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

Configuring for High Availability

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

Configuring Load Balancing and Failover

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

Supporting Huge Caches

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

When the cache is used for large amounts of data, the pages will typically be disk-backed. Since the cache eventually expires each page, thus releasing the disk resources, the cache uses a virtual erase optimization by default. This means that data that is explicitly removed or expired from the cache is not actually removed from the underlying Binary Store, but when a page (a Binary Store) is completely emptied, it will be erased in its entirety. This reduces IO by a considerable margin, particularly during expiry processing and during operations such as load-balancing that have to

redistribute large amounts of data within the cluster. The cost of this optimization is that the disk files (if a disk-based Binary Store option is used) will tend to be larger than the data that they are managing would otherwise imply; since disk space is considered to be inexpensive compared to other factors such as response times, the virtual erase optimization is enabled by default, but it can be disabled. Note that the disk space is typically allocated locally to each server, and thus a terabyte cache partitioned over one hundred servers would only use about 20GB of disk space per server (10GB for the primary store and 10GB for the backup store, assuming one level of backup.)

Cache Configurations by Example

This section provides a series of basic cache scheme definitions that can be used or modified as required. See [Chapter 11, "Configuring Caches,"](#) for detailed instructions on how to configure caches. In addition, the samples in this chapter build upon one another and often use a `<scheme-ref>` element to reuse other samples as nested schemes. See ["Using Scheme Inheritance"](#) on page 11-9 for details on using the `<scheme-ref>` element. Lastly, these samples only specify a minimum number of settings, follow the embedded links to a scheme's documentation to see the full set of options.

This section describes configurations for the following caching scenarios:

- [Local Caches \(accessible from a single JVM\)](#)
 - [In-memory Cache](#)
 - [NIO In-memory Cache](#)
 - [Size Limited In-memory Cache](#)
 - [In-memory Cache with Expiring Entries](#)
 - [Cache on Disk](#)
 - [Size Limited Cache on Disk](#)
 - [Persistent Cache on Disk](#)
 - [In-memory Cache with Disk Based Overflow](#)
 - [Cache of a Database](#)
- [Clustered Caches \(accessible from multiple JVMs\)](#)
 - [Replicated Cache](#)
 - [Replicated Cache with Overflow](#)
 - [Partitioned Cache](#)
 - [Partitioned Cache with Overflow](#)
 - [Partitioned Cache of a Database](#)
 - [Partitioned Cache with a Serializer](#)
 - [Local Cache of a Partitioned Cache \(Near cache\)](#)

Local Caches (accessible from a single JVM)

This section defines a series of local cache schemes. In this context "local" means that the cache is only directly accessible by a single JVM. Later in this document local

caches will be used as building blocks for clustered caches. See "[Clustered Caches \(accessible from multiple JVMs\)](#)" on page 15-5.

In-memory Cache

[Example 15-1](#) uses a [local-scheme](#) to define an in-memory cache. The cache will store as much as the JVM heap will allow.

Example 15-1 Configuration for a Local, In-memory Cache

```
<local-scheme>
  <scheme-name>SampleMemoryScheme</scheme-name>
</local-scheme>
```

NIO In-memory Cache

[Example 15-2](#) uses an [external-scheme](#) to define an in-memory cache using an [nio-memory-manager](#). The advantage of an NIO memory based cache is that it allows for large in-memory cache storage while not negatively impacting the JVM's GC times. The size of the cache is limited by the maximum size of the NIO memory region. See the `<maximum-size>` subelement of [nio-memory-manager](#).

Example 15-2 Configuration for a NIO In-memory Cache

```
<external-scheme>
  <scheme-name>SampleNioMemoryScheme</scheme-name>
  <nio-memory-manager/>
</external-scheme>
```

Size Limited In-memory Cache

Adding a `<high-units>` sub element to `<local-scheme>` limits the size of the cache. Here the cache is size limited to one thousand entries. When the limit is exceeded, the scheme's `<eviction-policy>` will determine which elements to evict from the cache.

Example 15-3 Configuration for a Size Limited, In-memory, Local Cache

```
<local-scheme>
  <scheme-name>SampleMemoryLimitedScheme</scheme-name>
  <high-units>1000</high-units>
</local-scheme>
```

In-memory Cache with Expiring Entries

Adding an `<expiry-delay>` subelement to `<local-scheme>` will cause cache entries to automatically expire if they are not updated for a given time interval. When expired the cache will invalidate the entry, and remove it from the cache.

Example 15-4 Configuration for an In-memory Cache with Expiring Entries

```
<local-scheme>
  <scheme-name>SampleMemoryExpirationScheme</scheme-name>
  <expiry-delay>5m</expiry-delay>
</local-scheme>
```

Cache on Disk

Example 15-5 uses an [external-scheme](#) to define an on disk cache. The cache will store as much as the file system will allow.

Note: This example uses the [lh-file-manager](#) for its on disk storage implementation. See [external-scheme](#) for additional external storage options.

Example 15-5 Configuration to Define a Cache on Disk

```
<external-scheme>
  <scheme-name>SampleDiskScheme</scheme-name>
  <lh-file-manager/>
</external-scheme>
```

Size Limited Cache on Disk

Adding a `<high-units>` sub- element to [external-scheme](#) limits the size of the cache. The cache is size limited to one million entries. When the limit is exceeded, LRU eviction is used determine which elements to evict from the cache. Refer to ["paged-external-scheme"](#) on page B-73 for an alternate size limited external caching approach.

Example 15-6 Configuration for a Size Limited Cache on Disk

```
<external-scheme>
  <scheme-name>SampleDiskLimitedScheme</scheme-name>
  <lh-file-manager/>
  <high-units>1000000</high-units>
</external-scheme>
```

Persistent Cache on Disk

Example 15-7 uses an [external-scheme](#) to implement a cache suitable for use as long-term storage for a single JVM.

External caches are generally used for temporary storage of large data sets, and are automatically deleted on JVM shutdown. An external-cache can be used for long term storage (see ["Persistence \(long-term storage\)"](#) on page B-35) in non-clustered caches when using either the [lh-file-manager](#) or [bdb-store-manager](#) storage managers. For clustered persistence see the ["Partitioned Cache of a Database"](#) on page 15-6 sample.

The `{cache-name}` macro is used to specify the name of the file the data will be stored in. See ["Using Parameter Macros"](#) on page 11-12 for more information on this macro.

Example 15-7 Configuration for Persistent cache on disk

```
<external-scheme>
  <scheme-name>SampleDiskPersistentScheme</scheme-name>
  <lh-file-manager>
    <directory>/my/storage/directory</directory>
    <file-name>{cache-name}.store</file-name>
  </lh-file-manager>
</external-scheme>
```

[Example 15-8](#) illustrates using Berkeley DB rather than LH.

Example 15-8 Configuration for Persistent cache on disk with Berkeley DB

```
<external-scheme>
  <scheme-name>SampleDiskPersistentScheme</scheme-name>
  <bdb-store-manager>
    <directory>/my/storage/directory</directory>
    <store-name>{cache-name}.store</store-name>
  </bdb-store-manager>
</external-scheme>
```

In-memory Cache with Disk Based Overflow

[Example 15-9](#) uses an [overflow-scheme](#) to define a size limited in-memory cache, when the in-memory (`<front-scheme>`) size limit is reached, a portion of the cache contents will be moved to the on disk (`<back-scheme>`). The front-scheme's `<eviction-policy>` will determine which elements to move from the front to the back.

Note that this example reuses the examples in ["Size Limited Cache on Disk"](#) and ["Cache on Disk"](#) on page 15-3. to implement the front and back of the cache.

Example 15-9 Configuration for In-memory Cache with Disk Based Overflow

```
<overflow-scheme>
  <scheme-name>SampleOverflowScheme</scheme-name>
  <front-scheme>
    <local-scheme>
      <scheme-ref>SampleMemoryLimitedScheme</scheme-ref>
    </local-scheme>
  </front-scheme>
  <back-scheme>
    <external-scheme>
      <scheme-ref>SampleDiskScheme</scheme-ref>
    </external-scheme>
  </back-scheme>
</overflow-scheme>
```

Cache of a Database

[Example 15-10](#) uses a [read-write-backing-map-scheme](#) to define a cache of a database. This scheme maintains local cache of a portion of the database contents. Cache misses will read-through to the database, and cache writes will be written back to the database.

The [cachestore-scheme](#) element is configured with a custom class implementing either the `com.tangosol.net.cache.CacheLoader` or `com.tangosol.net.cache.CacheStore` interface. This class is responsible for all operations against the database, such as reading and writing cache entries. See ["Sample CacheStore"](#) on page 13-8 implementations for examples of writing a cache store.

The `{cache-name}` macro is used to inform the cache store implementation of the name of the cache it will back. See ["Using Parameter Macros"](#) on page 11-12 for more information on this macro.

Example 15-10 Configuration for the Cache of a Database

```
<read-write-backing-map-scheme>
```

```

<scheme-name>SampleDatabaseScheme</scheme-name>
<internal-cache-scheme>
  <local-scheme>
    <scheme-ref>SampleMemoryScheme</scheme-ref>
  </local-scheme>
</internal-cache-scheme>
<cachestore-scheme>
  <class-scheme>
    <class-name>com.tangosol.examples.coherence.DBCacheStore</class-name>
    <init-params>
      <init-param>
        <param-type>java.lang.String</param-type>
        <param-value>{cache-name}</param-value>
      </init-param>
    </init-params>
  </class-scheme>
</cachestore-scheme>
</read-write-backing-map-scheme>

```

Clustered Caches (accessible from multiple JVMs)

This section defines a series of clustered cache examples. Clustered caches are accessible from multiple JVMs (any cluster node running the same cache service). The internal cache storage (backing-map) on each cluster node is defined using local caches (see ["Local Caches \(accessible from a single JVM\)"](#) on page 15-1). The cache service provides the capability to access local caches from other cluster nodes.

Replicated Cache

[Example 15-11](#) uses the `replicated-scheme` element to define a clustered cache in which a copy of each cache entry will be stored on all cluster nodes.

The sample in ["In-memory Cache"](#) on page 15-2 is used to define the cache storage on each cluster node. The size of the cache is only limited by the cluster node with the smallest JVM heap.

Example 15-11 Configuration for a Replicated Cache

```

<replicated-scheme>
  <scheme-name>SampleReplicatedScheme</scheme-name>
  <backing-map-scheme>
    <local-scheme>
      <scheme-ref>SampleMemoryScheme</scheme-ref>
    </local-scheme>
  </backing-map-scheme>
</replicated-scheme>

```

Replicated Cache with Overflow

The `backing-map-scheme` element could just as easily specify any of the other local cache samples. For instance, if it had used the ["In-memory Cache with Disk Based Overflow"](#) on page 15-4, each cluster node would have a local overflow cache allowing for much greater storage capacity.

Example 15-12 Configuration for a Replicated Cache with Overflow

```

<replicated-scheme>
  <scheme-name>SampleReplicatedOverflowScheme</scheme-name>
  <backing-map-scheme>

```

```
<overflow-scheme>
  <scheme-ref>SampleOverflowScheme</scheme-ref>
</overflow-scheme>
</backing-map-scheme>
</replicated-scheme>
```

Partitioned Cache

[Example 15-13](#) uses the [distributed-scheme](#) to define a clustered cache in which cache storage is partitioned across all cluster nodes.

The "In-memory Cache" on page 15-2 is used to define the cache storage on each cluster node. The total storage capacity of the cache is the sum of all storage enabled cluster nodes running the partitioned cache service. See the `<local-storage>` subelement of "[distributed-scheme](#)" on page B-27.

Example 15-13 Configuration for a Partitioned Cache

```
<distributed-scheme>
  <scheme-name>SamplePartitionedScheme</scheme-name>
  <backing-map-scheme>
    <local-scheme>
      <scheme-ref>SampleMemoryScheme</scheme-ref>
    </local-scheme>
  </backing-map-scheme>
</distributed-scheme>
```

Partitioned Cache with Overflow

The `backing-map-scheme` element could just as easily specify any of the other local cache samples. For instance if it had used the "In-memory Cache with Disk Based Overflow" on page 15-4, each storage-enabled cluster node would have a local overflow cache allowing for much greater storage capacity. Note that the cache's backup storage also uses the same overflow scheme which allows for backup data to be overflowed to disk.

Example 15-14 Configuration for a Partitioned Cache with Overflow

```
<distributed-scheme>
  <scheme-name>SamplePartitionedOverflowScheme</scheme-name>
  <backing-map-scheme>
    <overflow-scheme>
      <scheme-ref>SampleOverflowScheme</scheme-ref>
    </overflow-scheme>
  </backing-map-scheme>
  <backup-storage>
    <type>scheme</type>
    <scheme-name>SampleOverflowScheme</scheme-name>
  </backup-storage>
</distributed-scheme>
```

Partitioned Cache of a Database

Switching the `backing-map-scheme` element to use a [read-write-backing-map-scheme](#) allows the cache to load and store entries against an external source such as a database.

[Example 15-15](#) reuses the "Cache of a Database" on page 15-4 to define the database access.

Example 15–15 Configuration for a Partitioned Cache of a Database

```

<distributed-scheme>
  <scheme-name>SamplePartitionedDatabaseScheme</scheme-name>
  <backing-map-scheme>
    <read-write-backing-map-scheme>
      <scheme-ref>SampleDatabaseScheme</scheme-ref>
    </read-write-backing-map-scheme>
  </backing-map-scheme>
</distributed-scheme>

```

Partitioned Cache with a Serializer

[Example 15–16](#) uses the `serializer` element in `distributed-scheme` to define a serializer that will be used to serialize and deserialize user types. In this case, the partitioned cache will use POF (`ConfigurablePofContext`) as its serialization format. Note that if you use POF and your application uses any custom user type classes, then you must also define a custom POF configuration for them. See [Appendix D, "POF User Type Configuration Elements"](#) for more information on POF elements.

Example 15–16 Configuration for a Partitioned Cache with a Serializer

```

<distributed-scheme>
  <scheme-name>SamplePartitionedPofScheme</scheme-name>
  <service-name>PartitionedPofCache</service-name>
  <serializer>
    <class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>
  </serializer>
  <backing-map-scheme>
    <local-scheme/>
  </backing-map-scheme>
  <autostart>true</autostart>
</distributed-scheme>

```

Local Cache of a Partitioned Cache (Near cache)

[Example 15–17](#) uses the `near-scheme` to define a local in-memory cache of a subset of a partitioned cache. The result is that any cluster node accessing the partitioned cache will maintain a local copy of the elements it frequently accesses. This offers read performance close to the `replicated-scheme`-based caches, while offering the high scalability of a `distributed-scheme`-based cache.

The "Size Limited In-memory Cache" on page 15-2 sample is reused to define the "near" (`<front-scheme>`) cache, while the "Partitioned Cache" on page 15-6 sample is reused to define the `near-scheme`.

Note that the size limited configuration of the front-scheme specifies the limit on how much of the back-scheme cache is locally cached.

Example 15–17 Configuration for a Local Cache of a Partitioned Cache

```

<near-scheme>
  <scheme-name>SampleNearScheme</scheme-name>
  <front-scheme>
    <local-scheme>
      <scheme-ref>SampleLimitedMemoryScheme</scheme-ref>
    </local-scheme>
  </front-scheme>
  <back-scheme>

```

```
<distributed-scheme>  
  <scheme-ref>SamplePartitionedScheme</scheme-ref>  
</distributed-scheme>  
</back-scheme>  
</near-scheme>
```

Part IV

Using the Programming API

Part IV contains the following chapters:

- [Chapter 16, "Serializing Objects"](#)
- [Chapter 17, "Using Portable Object Format"](#)
- [Chapter 18, "Pre-Loading the Cache"](#)
- [Chapter 19, "Using Cache Events"](#)
- [Chapter 20, "Querying Data In a Cache"](#)
- [Chapter 21, "Using Continuous Query Caching"](#)
- [Chapter 22, "Processing Data In a Cache"](#)
- [Chapter 23, "Managing Map Operations with Triggers"](#)
- [Chapter 24, "Using Coherence Query Language"](#)
- [Chapter 25, "Performing Transactions"](#)
- [Chapter 26, "Data Affinity"](#)
- [Chapter 27, "Priority Tasks"](#)
- [Chapter 28, "Specifying a Custom Eviction Policy"](#)
- [Chapter 29, "Constraints on Re-entrant Calls"](#)
- [Chapter 30, "Securing Coherence"](#)

Serializing Objects

Use Coherence caches to cache value objects. These objects may represent data from any source, either internal (such as session data, transient data, and so on) or external (such as a database, mainframe, and so on).

Objects placed in the cache must be serializable. Because serialization is often the most expensive part of clustered data management, Coherence provides the following options for serializing/deserializing data:

- `com.tangosol.io.pof.PofSerializer` – The Portable Object Format (also referred to as POF) is a language agnostic binary format. POF was designed to be incredibly efficient in both space and time and has become the recommended serialization option in Coherence. See [Chapter 17, "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` (For more details, see the API Javadoc for `XmlBean`).

Note: Remember, when serializing an object, Java serialization automatically crawls every visible object (by using object references, including collections like `Map` and `List`). As a result, cached objects **should not** refer to their parent objects directly (holding onto an identifying value like an integer is OK).

Objects that implement their own serialization routines are not affected.

Using Portable Object Format

Using Portable Object Format (POF) has many advantages ranging from performance benefits to language independence. It's recommended that you look closely at POF as your serialization solution when working with Coherence. For information on how to work with POF when building .NET extend clients, see "Building Integration Objects for .NET Clients" in *Oracle Coherence Client Guide*. For information on how to work with POF when building C++ extend clients, see "Building Integration Objects for C++ Clients" in *Oracle Coherence Client Guide*.

The following sections are included in this chapter:

- [Overview](#)
- [Working with POF](#)
- [Using POF Extractors and POF Updaters](#)

Overview

Serialization is the process of encoding an object into a binary format. It is a critical component to working with Coherence as data needs to be moved around the network. The Portable Object Format (also referred to as POF) is a language agnostic binary format. POF was designed to be incredibly efficient in both space and time and has become a cornerstone element in working with Coherence. For more information on the POF binary stream, see [Appendix F, "The PIF-POF Binary Format."](#)

There are several options available with respect to serialization including standard Java serialization, POF, and your own custom serialization routines. Each has their own trade-offs. Standard Java serialization is easy to implement, supports cyclic object graphs and preserves object identity. Unfortunately, it's also comparatively slow, has a verbose binary format, and restricted to only Java objects.

The Portable Object Format on the other hand has the following advantages:

- It's language independent with current support for Java, .NET, and C++.
- It's very efficient, in a simple test class with a `String`, a `long`, and three `ints`, (de)serialization was seven times faster, and the binary produced was one sixth the size compared with standard Java serialization.
- It's versionable, objects can evolve and have forward and backward compatibility.
- It supports the ability to externalize your serialization logic.
- It's indexed which allows for extracting values without deserializing the whole object. See ["Using POF Extractors and POF Updaters"](#) on page 17-6.

Working with POF

POF requires you to implement serialization routines that know how to serialize and deserialize your objects. There are two ways to do this:

- Have your objects implement the `com.tangosol.io.pof.PortableObject` interface.
- Implement a serializer for your objects using the `com.tangosol.io.pof.PofSerializer` interface.

Implementing the `PortableObject` interface

The `PortableObject` interface is a simple interface made up of two methods:

- `public void readExternal(PofReader reader)`
- `public void writeExternal(PofWriter writer)`

As mentioned above, POF elements are indexed. This is accomplished by providing a numerical index for each element that you write or read from the POF stream. It's important to keep in mind that the indexes must be unique to each element written and read from the POF stream, especially when you have derived types involved because the indexes must be unique between the super class and the derived class.

[Example 17-1](#) is a simple example of implementing the `PortableObject` interface:

Example 17-1 *Implementation of the `PortableObject` Interface*

```
public void readExternal(PofReader in)
    throws IOException
{
    m_symbol    = (Symbol) in.readObject(0);
    m_ldtPlaced = in.readLong(1);
    m_fClosed   = in.readBoolean(2);
}

public void writeExternal(PofWriter out)
    throws IOException
{
    out.writeObject(0, m_symbol);
    out.writeLong(1, m_ldtPlaced);
    out.writeBoolean(2, m_fClosed);
}
```

Implementing the `PofSerializer` interface:

The `PofSerializer` interface provide you with a way to externalize your serialization logic from the classes you want to serialize. This is particularly useful when you don't 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 deserializer(PofReader in)`
- `public void serialize(PofWriter out, Object o)`

As with the `PortableObject` interface, all elements written to or read from the POF stream must be uniquely indexed. Below is an example implementation of the `PofSerializer` interface:

Example 17-2 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 we're done reading the object
    in.readRemainder(null);

    return new Trade(symbol, ldtPlaced, fClosed);
}

public void serialize(PofWriter out, Object o)
    throws IOException
{
    Trade trade = (Trade) o;
    out.writeObject(0, trade.getSymbol());
    out.writeLong(1, trade.getTimePlaced());
    out.writeBoolean(2, trade.isClosed());

    // mark that we're done writing the object
    out.writeRemainder(null);
}

```

Assigning POF indexes

Use the following guidelines when assigning POF indexes to an object's attributes:

- Order your reads and writes: start with the lowest index value in the serialization routine and finish with the highest. When deserializing a value, perform reads in the same order as writes.
- Non-contiguous indexes are acceptable but must be read/written sequentially.
- When Subclassing reserve index ranges: index's are cumulative across derived types. As such, each derived type must be aware of the POF index range reserved by its super class.
- Don not re-purpose indexes: to support Evolvable, it's imperative that indexes of attributes are not re-purposed across class revisions.
- Label indexes: indexes that are labeled with a public static final int, are much easier to work with, especially when using POF Extractors and POF Updaters. See ["Using POF Extractors and POF Updaters"](#) on page 17-6. Indexes that are labeled must still be read and written out in the same order as mentioned above.

Using the ConfigurablePofContext Class

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 configuration file. The POF configuration file is an XML file that has a `<user-type-list>` element that contains a list of classes that implement `PortableObject` or have a `PofSerializer`

associated with them. The `<type-id>` for each class must be unique, and must match across all cluster instances (including extend clients). See [Appendix D, "POF User Type Configuration Elements,"](#) for detailed reference of the POF configuration elements.

The following is an example of a POF configuration file:

```
<pof-config>
  <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>
    ...
</pof-config>
```

Note: Coherence reserves the first 1000 type-id's for internal use. As shown in the above example, the `<user-type-list>` includes the `coherence-pof-config.xml` file that is located in the root of the `coherence.jar` file. This is where Coherence specific user types are defined and should be included in all of your POF configuration files.

Configuring Coherence to Use the ConfigurablePofContext Class

Coherence can be configured to use the `ConfigurablePofContext` serializer class in three different ways based on the level of granularity that is required:

- **Per Service** – Each service provides a full `ConfigurablePofContext` serializer class configuration or references a predefined configuration that is included in the operational configuration file.
- **All Services** – All services use a global `ConfigurablePofContext` serializer class configuration. Services that provide their own configuration override the global configuration. The global configuration can also be a full configuration or reference a predefined configuration that is included in the operational configuration file.
- **JVM** – The `ConfigurablePofContext` serializer class is enabled for the whole JVM.

Configure the ConfigurablePofContext Class Per Service

To configure a service to use the `ConfigurablePofContext` class, add a `<serializer>` element to a cache scheme in a cache configuration file. See ["serializer"](#) on page B-95 for a complete reference of the `<serializer>` element.

The following example demonstrates a distributed cache that is configured to use the `ConfigurablePofContext` class and defines a custom POF configuration file:

```

<distributed-scheme>
  <scheme-name>example-distributed</scheme-name>
  <service-name>DistributedCache</service-name>
  <serializer>
    <instance>
      <class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>
      <init-params>
        <init-param>
          <param-value>my-pof-config.xml</param-value>
          <param-type>String</param-type>
        </init-param>
      </init-params>
    </instance>
  </serializer>
  ...
</distributed-scheme>

```

The following example references the default definition in the operational configuration file. Refer to ["serializers"](#) on page A-61 to see the default ConfigurablePofContext serializer definition.

```

<distributed-scheme>
  <scheme-name>example-distributed</scheme-name>
  <service-name>DistributedCache</service-name>
  <serializer>pof</serializer>
  ...
</distributed-scheme>

```

Configure the ConfigurablePofContext Class for All Services

To globally configure the ConfigurablePofContext class for all services, add a `<serializer>` element within the `<defaults>` element in a cache configuration file. Both of the below examples will globally configure a serializer for all cache scheme definitions and do not require any additional configuration within individual cache scheme definitions. See ["defaults"](#) on page B-25 for a complete reference of the `<defaults>` element.

The following example demonstrates a global configuration for the ConfigurablePofContext class and defines a custom POF configuration file:

```

<cache-config>
  <defaults>
    <serializer>
      <instance>
        <class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>
        <init-params>
          <init-param>
            <param-value>my-pof-config.xml</param-value>
            <param-type>String</param-type>
          </init-param>
        </init-params>
      </instance>
    </serializer>
  </defaults>
  ...

```

The following example references the default definition in the operational configuration file. Refer to ["serializers"](#) on page A-61 to see the default ConfigurablePofContext serializer definition.

```
<cache-config>
  <defaults>
    <serializer>pof</serializer>
  </defaults>
  ...
```

Configure the ConfigurablePofContext Class For the JVM

An entire JVM instance can be configured to use POF using the following system properties:

- `tangosol.pof.enabled=true` - This enables POF for the entire JVM instance.
- `tangosol.pof.config=CONFIG_FILE_PATH` - The path to the POF configuration file you want to use. If the file is not in the classpath, then it must be presented as a file resource (for example, `file:///opt/home/coherence/mycustom-pof-config.xml`).

Using POF Extractors and POF Updaters

In Coherence, the `ValueExtractor` and `ValueUpdater` interfaces are used to extract and update values of objects that are stored in the cache. The `PofExtractor` and `PofUpdater` interfaces take advantage of the POF indexed state to extract or update an object without the need to go through the full serialization/deserialization routines.

`PofExtractor` and `PofUpdater` adds flexibility in working with non-primitive types in Coherence. For most cases where you're working with extend clients, it's no longer required to have corresponding Java classes in the grid. Because POF extractors and POF updaters can navigate the binary, the entire key/value does not need 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. There are however circumstances where you must provide a corresponding Java class:

- **Key Association** – When using key association, Coherence always deserializes keys to determine whether they implement key association.
- **Cache Stores** – When using a cache store, Coherence passes the deserialized version of the key and value to the cache store to write to the back end.

Navigating a POF object

Due to the fact that the Portable Object Format (POF) is indexed, it's possible to quickly traverse the binary to a specific element for extraction or updating. It's the responsibility of the `PofNavigator` interface to traverse a POF value object and return the desired POF value object. Out of the box, Coherence provides a `SimplePofPath` class that can navigate a POF value based on integer indices. In the simplest form, all you need to do is to provide the index of the attribute that you want to extract/update.

Consider the following example:

```
public class Contact
    implements PortableObject
{
    ...
    // ----- PortableObject interface -----

    /**
```

```

    * {@inheritDoc}
    */
    public void readExternal(PofReader reader)
        throws IOException
    {
        m_sFirstName      = reader.readString(FIRSTNAME);
        m_sLastName       = reader.readString(LASTNAME);
        m_addrHome        = (Address) reader.readObject(HOME_ADDRESS);
        m_addrWork        = (Address) reader.readObject(WORK_ADDRESS);
        m_mapPhoneNumber = reader.readMap(PHONE_NUMBERS, null);
    }

    /**
    * {@inheritDoc}
    */
    public void writeExternal(PofWriter writer)
        throws IOException
    {
        writer.writeString(FIRSTNAME, m_sFirstName);
        writer.writeString(LASTNAME, m_sLastName);
        writer.writeObject(HOME_ADDRESS, m_addrHome);
        writer.writeObject(WORK_ADDRESS, m_addrWork);
        writer.writeMap(PHONE_NUMBERS, m_mapPhoneNumber);
    }

    ....

    // ----- constants -----

    /**
    * The POF index for the FirstName property
    */
    public static final int FIRSTNAME = 0;

    /**
    * The POF index for the LastName property
    */
    public static final int LASTNAME = 1;

    /**
    * The POF index for the HomeAddress property
    */
    public static final int HOME_ADDRESS = 2;

    /**
    * The POF index for the WorkAddress property
    */
    public static final int WORK_ADDRESS = 3;

    /**
    * The POF index for the PhoneNumbers property
    */
    public static final int PHONE_NUMBERS = 4;

    ...
}

```

Notice that there's a constant for each data member that is being written to and from the POF stream. This is an excellent practice to follow as it simplifies both writing your serialization routines as well as making it easier to work with POF extractors and POF updaters. By labeling each index, it becomes much easier to think about what we're

working with. As mentioned above, in the simplest case, we could pull the work address out of the contact by using the `WORK_ADDRESS` index. The `SimplePofPath` also allows using an Array of ints to traverse the `PofValues`. So for example if we wanted the zip code of the work address we would use `[WORK_ADDRESS, ZIP]`. We'll go through the example in more detail below.

Using PofExtractors

`PofExtractors` are typically used when querying a cache and should greatly improve the performance of your queries. If we were to use the example class above, and wanted to query the cache for all Contacts with the last names Jones, the query would look something like this:

```
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 use case, `PofExtractor` has a convenience constructor that will use a `SimplePofPath` to retrieve a singular index, in our case the `Contact.LASTNAME` index. Now if we wanted to find all Contacts with the area code 01803, the query would look like this:

```
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 PofUpdaters

`PofUpdater` works in the same way as `PofExtractor` except that they will update the value of an object rather than extract it. So if we wanted to change all entries with the last name of Jones to Smith, we would use the `UpdaterProcessor` like this:

```
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 will work on any POF encoded value.

Pre-Loading the Cache

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

Performing Bulk Loading and Processing

[Example 18–5](#), `PagedQuery.java`, demonstrates techniques for efficiently bulk loading and processing items in a Coherence Cache.

Bulk Writing to a Cache

A common scenario when using Coherence is to pre-populate a cache before the application uses it. A simple way to do this is illustrated by the Java code in [Example 18–1](#):

Example 18–1 Pre-Loading a Cache

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

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

This technique works, but each call to `put` may result in network traffic, especially for partitioned and replicated caches. Additionally, each call to `put` will return the object it just replaced in the cache (per the `java.util.Map` interface) which will add more unnecessary overhead. Loading the cache can be made much more efficient by using the `ConcurrentMap.putAll` method instead. This is illustrated in [Example 18–2](#):

Example 18–2 Pre-Loading a Cache Using `ConcurrentMap.putAll`

```

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

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

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

```

Efficient processing of filter results

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

Example 18–3 Using a Filter to Query a Cache

```

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

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

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

```

This example works for small data sets, but it may encounter problems, such as running out of heap space, if the data set is too large. [Example 18–4](#) illustrates a pattern to process query results in batches to avoid this problem:

Example 18–4 Processing Query Results in Batches

```

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

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

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

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

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

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

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

            // Process each entry
            process(entries);

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

public static void process(Map map)
{
    for (Iterator ie = map.entrySet().iterator(); ie.hasNext();)
    {
        Map.Entry e = (Map.Entry) ie.next();
        out("key: "+e.getKey() + ", value: "+e.getValue());
    }
}

```

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

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

A Bulk Loading and Processing Example

[Example 18–5](#) illustrates `PagedQuery.java`, a sample program that demonstrates the concepts described in the previous section.

To run the example, follow these steps:

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

Example 18–5 A Sample Bulk Loading Program

```
package com.tangosol.examples;

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

import java.io.Serializable;

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

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

```

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

    populateCache(cacheContacts);

    executeFilter(cacheContacts);

    CacheFactory.shutdown();
}

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

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

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

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

```

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

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

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

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

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

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

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

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

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

```

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

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

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

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

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

    public String getName()
    {
        return m_sName;
    }

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

    public String getPhone()
    {
        return m_sPhone;
    }

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

    public String toString()
    {

```

```

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

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

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

    private String m_sName;
    private String m_sPhone;
}

```

Example 18–6 illustrates the terminal output from Coherence when you compile and run the example:

Example 18–6 Terminal Output from the Bulk Loading Program

```

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

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

$ javac com/tangosol/examples/PagedQuery.java

$ java com.tangosol.examples.PagedQuery

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

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

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

```



```

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

...repeated many times...

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

```

Performing Distributed Bulk Loading

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

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

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

A Distributed Bulk Loading Example

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

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

Example 18–7 Retrieving Storage-Enabled Members of the Cache

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

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

Example 18–8 Routine to Get a List of Files Assigned to a Cache Member

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

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

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

Example 18–9 Class to Load Each Member of the Cache

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

    Set members = getStorageMembers(cache);

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

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

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

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

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

Using Cache Events

Coherence provides cache events using the JavaBean Event model. It is extremely simple to receive the events that you need, where you need them, regardless of where the changes are actually occurring in the cluster. Developers with any experience with the JavaBean model will have no difficulties working with events, even in a complex cluster.

Listener Interface and Event Object

In the JavaBeans Event model, there is an `EventListener` interface that all listeners must extend. Coherence provides a `MapListener` interface, which allows application logic to receive events when data in a Coherence cache is added, modified or removed.

[Example 19-1](#) illustrates an excerpt from the `com.tangosol.util.MapListener` API.

Example 19-1 Excerpt from the MapListener API

```
public interface MapListener
    extends EventListener
{
    /**
     * Invoked when a map entry has been inserted.
     *
     * @param evt the MapEvent carrying the insert information
     */
    public void entryInserted(MapEvent evt);

    /**
     * Invoked when a map entry has been updated.
     *
     * @param evt the MapEvent carrying the update information
     */
    public void entryUpdated(MapEvent evt);

    /**
     * Invoked when a map entry has been removed.
     *
     * @param evt the MapEvent carrying the delete information
     */
    public void entryDeleted(MapEvent evt);
}
```

An application object that implements the `MapListener` interface can sign up for events from any Coherence cache or class that implements the `ObservableMap`

interface, simply by passing an instance of the application's `MapListener` implementation to one of the `addMapListener()` methods.

The `MapEvent` object that is passed to the `MapListener` carries all of the necessary information about the event that has occurred, including the *source* (`ObservableMap`) that raised the event, the *identity* (key) that the event is related to, what the *action* was against that identity (insert, update or delete), what the old value was and what the new value is:

[Example 19-2](#) illustrates an excerpt from the `com.tangosol.util.MapEvent` API.

Example 19-2 Excerpt from the `MapEvent` API

```
public class MapEvent
    extends EventObject
{
    /**
     * Return an ObservableMap object on which this event has actually
     * occurred.
     *
     * @return an ObservableMap object
     */
    public ObservableMap getMap()

    /**
     * Return this event's id. The event id is one of the ENTRY_*
     * enumerated constants.
     *
     * @return an id
     */
    public int getId()

    /**
     * Return a key associated with this event.
     *
     * @return a key
     */
    public Object getKey()

    /**
     * Return an old value associated with this event.
     * <p>
     * The old value represents a value deleted from or updated in a map.
     * It is always null for "insert" notifications.
     *
     * @return an old value
     */
    public Object getOldValue()

    /**
     * Return a new value associated with this event.
     * <p>
     * The new value represents a new value inserted into or updated in
     * a map. It is always null for "delete" notifications.
     *
     * @return a new value
     */
    public Object getNewValue()

    // ----- Object methods -----
```

```

/**
 * Return a String representation of this MapEvent object.
 *
 * @return a String representation of this MapEvent object
 */
public String toString()

// ----- constants -----

/**
 * This event indicates that an entry has been added to the map.
 */
public static final int ENTRY_INSERTED = 1;

/**
 * This event indicates that an entry has been updated in the map.
 */
public static final int ENTRY_UPDATED = 2;

/**
 * This event indicates that an entry has been removed from the map.
 */
public static final int ENTRY_DELETED = 3;
}

```

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 will be delivered to the application's listeners.

In addition to the Coherence caches (those objects obtained through a Coherence cache factory), several other supporting classes in Coherence also implement the `ObservableMap` interface:

- `ObservableHashMap`
- `LocalCache`
- `OverflowMap`
- `NearCache`
- `ReadWriteBackingMap`
- `AbstractSerializationCache`, `SerializationCache`, and `SerializationPagedCache`
- `WrapperObservableMap`, `WrapperConcurrentMap`, and `WrapperNamedCache`

For a full list of published implementing classes, see the Coherence Javadoc for `ObservableMap`.

Signing Up for All Events

To sign up for events, simply pass an object that implements the `MapListener` interface to one of the `addMapListener` methods on `ObservableMap`. The `addMapListener` methods are illustrated in [Example 19-3](#).

Example 19-3 *Methods on the ObservableMap API*

```
public void addMapListener(MapListener listener);
public void addMapListener(MapListener listener, Object oKey, boolean fLite);
public void addMapListener(MapListener listener, Filter filter, boolean fLite);
```

Let's create an example `MapListener` implementation. [Example 19-4](#) illustrates a sample `MapListener` implementation that prints each event as it receive.

Example 19-4 *Sample MapListener Implementation*

```
/**
 * A MapListener implementation that prints each event as it receives
 * them.
 */
public static class EventPrinter
    extends Base
    implements MapListener
{
    public void entryInserted(MapEvent evt)
    {
        out(evt);
    }

    public void entryUpdated(MapEvent evt)
    {
        out(evt);
    }

    public void entryDeleted(MapEvent evt)
    {
        out(evt);
    }
}
```

Using this implementation, it is extremely simple to print out all events from any given cache (since all caches implement the `ObservableMap` interface):

```
cache.addMapListener(new EventPrinter());
```

Of course, to be able to later remove the listener, it is necessary to hold on to a reference to the listener:

Example 19-5 *Holding a Reference to a Listener*

```
Listener listener = new EventPrinter();
cache.addMapListener(listener);
m_listener = listener; // store the listener in a field
```

Later, to remove the listener:

Example 19–6 Removing a Listener

```
Listener listener = m_listener;
if (listener != null)
{
    cache.removeMapListener(listener);
    m_listener = null; // clean up the listener field
}
```

Each `addMapListener` method on the `ObservableMap` interface has a corresponding `removeMapListener` method. To remove a listener, use the `removeMapListener` method that corresponds to the `addMapListener` method that was used to add the listener.

Using an Inner Class as a MapListener

When creating an inner class to use as a `MapListener`, or when implementing a `MapListener` that only listens to one or two types of events (inserts, updates or deletes), you can use the `AbstractMapListener` base class. For example, the anonymous inner class in [Example 19–7](#) prints out only the insert events for the cache.

Example 19–7 Inner Class that Prints Only Cache Insert Events

```
cache.addMapListener(new AbstractMapListener()
{
    public void entryInserted(MapEvent evt)
    {
        out(evt);
    }
});
```

Another helpful base class for creating a `MapListener` is the `MultiplexingMapListener`, which routes all events to a single method for handling. This class would allow you to simplify the `EventPrinter` example to the code illustrated in [Example 19–8](#). Since only one method must be implemented to capture all events, the `MultiplexingMapListener` can also be very useful when creating an inner class to use as a `MapListener`.

Example 19–8 Routing All Events to a Single Method for Handling

```
public static class EventPrinter
    extends MultiplexingMapListener
{
    public void onMapEvent(MapEvent evt)
    {
        out(evt);
    }
}
```

Configuring a MapListener for a Cache

If the listener should always be on a particular cache, then place it into the cache configuration using the `<listener>` element and Coherence will automatically add the listener when it configures the cache.

Signing up for Events on specific identities

Signing up for events that occur against specific identities (keys) is just as simple. For example, to print all events that occur against the Integer key 5:

```
cache.addMapListener(new EventPrinter(), new Integer(5), false);
```

Thus, the code in [Example 19-9](#) would only trigger an event when the Integer key 5 is inserted or updated:

Example 19-9 Triggering an Event when a Specific Integer Key is Inserted or Updated

```
for (int i = 0; i < 10; ++i)
{
    Integer key = new Integer(i);
    String value = "test value for key " + i;
    cache.put(key, value);
}
```

Filtering Events

Similar to listening to a particular key, it is possible to listen to particular events. In [Example 19-10](#) a listener is added to the cache with a filter that allows the listener to only receive delete events.

Example 19-10 Adding a Listener with Filter for Deleted Events

```
// Filters used with partitioned caches must be
// Serializable, Externalizable or ExternalizableLite
public class DeletedFilter
    implements Filter, Serializable
{
    public boolean evaluate(Object o)
    {
        MapEvent evt = (MapEvent) o;
        return evt.getId() == MapEvent.ENTRY_DELETED;
    }
}

cache.addMapListener(new EventPrinter(), new DeletedFilter(), false);
```

Note: Filtering events versus filtering cached data:

When building a Filter for querying, the object that will be passed to the evaluate method of the Filter will be 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 will be passed to the evaluate method of the Filter will always be of type MapEvent.

For more information on how to use a query filter to listen to cache events, see ["Advanced: Listening to Queries"](#) on page 19-8.

If you then make the following sequence of calls:

```
cache.put("hello", "world");
cache.put("hello", "again");
cache.remove("hello");
```

The result would be:

```
CacheEvent{LocalCache deleted: key=hello, value=again}
```

For more information, see the ["Advanced: Listening to Queries"](#) on page 19-8.

"Lite" Events

By default, Coherence provides both the old and the new value as part of an event. Consider the following example:

Example 19–11 Inserting, Updating, and Removing a Value from the Cache

```
MapListener listener = new MultiplexingMapListener()
{
    public void onMapEvent(MapEvent evt)
    {
        out("event has occurred: " + evt);
        out("(the wire-size of the event would have been "
            + ExternalizableHelper.toBinary(evt).length()
            + " bytes.)");
    }
};
cache.addMapListener(listener);

// insert a 1KB value
cache.put("test", new byte[1024]);

// update with a 2KB value
cache.put("test", new byte[2048]);

// remove the 2KB value
cache.remove("test");
```

The output from running the test, illustrated in [Example 19–12](#), shows that the first event carries the 1KB inserted value, the second event carries both the replaced 1KB value and the new 2KB value, and the third event carries the removed 2KB value.

Example 19–12 Sample Output

```
event has occurred: CacheEvent{LocalCache added: key=test, value=[B@a470b8}
(the wire-size of the event would have been 1283 bytes.)
event has occurred: CacheEvent{LocalCache updated: key=test, old value=[B@a470b8,
new value=[B@1c6f579}
(the wire-size of the event would have been 3340 bytes.)
event has occurred: CacheEvent{LocalCache deleted: key=test, value=[B@1c6f579}
(the wire-size of the event would have been 2307 bytes.)
```

When an application does not require the old and the new value to be included in the event, it can indicate that by requesting only "lite" events. When adding a listener, you can request lite events by using one of the two `addMapListener` methods that takes an additional boolean `fLite` parameter. In [Example 19–11](#), the only change would be:

```
cache.addMapListener(listener, (Filter) null, true);
```

Note: Obviously, a lite event's old value and new value may be null. However, even if you request lite events, the old and the new value *may* be included if there is no additional cost to generate and deliver the event. In other words, requesting that a `MapListener` receive lite events is simply a hint to the system that the `MapListener` does not need to know the old and new values for the event.

Advanced: Listening to Queries

All Coherence caches support querying by any criteria. When an application queries for data from a cache, the result is a point-in-time snapshot, either as a set of identities (`keySet`) or a set of identity/value pairs (`entrySet`). The mechanism for determining the contents of the resulting set is referred to as *filtering*, and it allows an application developer to construct queries of arbitrary complexity using a rich set of out-of-the-box filters (for example, equals, less-than, like, between, and so on), or to provide their own custom filters (for example, `XPath`).

The same filters that are used to query a cache can be used to listen to events from a cache. For example, in a trading system it is possible to query for all open `Order` objects for a particular trader:

Example 19–13 *Listening for Events from a Cache*

```
NamedCache mapTrades = ...
Filter filter = new AndFilter(new EqualsFilter("getTrader", traderid),
                             new EqualsFilter("getStatus", Status.OPEN));
Set setOpenTrades = mapTrades.entrySet(filter);
```

To receive notifications of new trades being opened for that trader, closed by that trader or reassigned to or from another trader, the application can use the same filter:

Example 19–14 *Listening for Events on an Object*

```
// receive events for all trade IDs that this trader is interested in
mapTrades.addMapListener(listener, new MapEventFilter(filter), true);
```

The `MapEventFilter` converts a query filter into an event filter.

The `MapEventFilter` has several very powerful options, allowing an application listener to receive only the events that it is specifically interested in. More importantly for scalability and performance, only the desired events have to be communicated over the network, and they are communicated only to the servers and clients that have expressed interest in those specific events. [Example 19–15](#) illustrates these scenarios.

Example 19–15 *Using MapEventFilter to Filter on Various Events*

```
// receive all events for all trades that this trader is interested in
nMask = MapEventFilter.E_ALL;
mapTrades.addMapListener(listener, new MapEventFilter(nMask, filter), true);

// receive events for all this trader's trades that are closed or
// re-assigned to a different trader
nMask = MapEventFilter.E_UPDATED_LEFT | MapEventFilter.E_DELETED;
mapTrades.addMapListener(listener, new MapEventFilter(nMask, filter), true);

// receive events for all trades as they are assigned to this trader
nMask = MapEventFilter.E_INSERTED | MapEventFilter.E_UPDATED_ENTERED;
mapTrades.addMapListener(listener, new MapEventFilter(nMask, filter), true);
```

```
// receive events only for new trades assigned to this trader
nMask = MapEventFilter.E_INSERTED;
mapTrades.addMapListener(listener, new MapEventFilter(nMask, filter), true);
```

For more information on the various options supported, see the API documentation for `MapEventFilter`.

Filtering Events Versus Filtering Cached Data

When building a `Filter` for querying, the object that will be passed to the `evaluate` method of the `Filter` will be 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 will be passed to the `evaluate` method of the `Filter` will always be of type `MapEvent`.

The `MapEventFilter` converts a `Filter` that is used to do a query into a `Filter` that is used to filter events for a `MapListener`. In other words, the `MapEventFilter` is constructed from a `Filter` that queries a cache, and the resulting `MapEventFilter` is a filter that evaluates `MapEvent` objects by converting them into the objects that a query `Filter` would expect.

Advanced: Synthetic Events

Events usually reflect the changes being made to a cache. For example, one server is modifying one entry in a cache while another server is adding several items to a cache while a third server is removing an item from the same cache, all while fifty threads on each and every server in the cluster is accessing data from the same cache! All the modifying actions will produce events that any server within the cluster can choose to receive. We refer to these actions as *client actions*, and the events as being *dispatched to clients*, even though the "clients" in this case are actually servers. This is a natural concept in a true peer-to-peer architecture, such as a Coherence cluster: Each and every peer is both a client and a server, both consuming services from its peers and providing services to its peers. In a typical Java Enterprise application, a "peer" is an application server instance that is acting as a container for the application, and the "client" is that part of the application that is directly accessing and modifying the caches and listening to events from the caches.

Some events originate from within a cache itself. There are many examples, but the most common cases are:

- When entries automatically expire from a cache;
- When entries are evicted from a cache because the maximum size of the cache has been reached;
- When entries are transparently added to a cache as the result of a Read-Through operation;
- When entries in a cache are transparently updated as the result of a Read-Ahead or Refresh-Ahead operation.

Each of these represents a modification, but the modifications represent natural (and typically automatic) operations from within a cache. These events are referred to as *synthetic* events.

When necessary, an application can differentiate between client-induced and synthetic events simply by asking the event if it is synthetic. This information is carried on a

sub-class of the `MapEvent`, called `CacheEvent`. Using the previous `EventPrinter` example, it is possible to print only the synthetic events:

Example 19–16 Determining Synthetic Events

```
public static class EventPrinter
    extends MultiplexingMapListener
{
    public void onMapEvent(MapEvent evt)
    {
        if (evt instanceof CacheEvent && ((CacheEvent) evt).isSynthetic())
        {
            out(evt);
        }
    }
}
```

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

Advanced: Backing Map Events

While it is possible to listen to events from Coherence caches, each of which presents a local view of distributed, partitioned, replicated, near-cached, continuously-queried, read-through/write-through and/or write-behind data, it is also possible to peek behind the curtains, so to speak.

For some advanced use cases, it may be necessary to "listen to" the "map" behind the "service". Replication, partitioning and other approaches to managing data in a distributed environment are all distribution *services*. The service still has to have something in which to actually *manage* the data, and that *something* is called a "backing map".

Backing maps can be configured. If all the data for a particular cache should be kept in object form on the heap, then use an unlimited and non-expiring `LocalCache` (or a `SafeHashMap` if statistics are not required). If only a small number of items should be kept in memory, use a `LocalCache`. If data are to be read on demand from a database, then use a `ReadWriteBackingMap` (which knows how to read and write through an application's DAO implementation), and in turn give the `ReadWriteBackingMap` a backing map such as a `SafeHashMap` or a `LocalCache` to store its data in.

Some backing maps are observable. The events coming from these backing maps are not usually of direct interest to the application. Instead, Coherence translates them into actions that must be taken (by Coherence) to keep data synchronous and properly backed up, and it also translates them when appropriate into clustered events that are delivered throughout the cluster as requested by application listeners. For example, if a partitioned cache has a `LocalCache` as its backing map, and the local cache expires an entry, that event causes Coherence to expire all of the backup copies of that entry. Furthermore, if any listeners have been registered on the partitioned cache, and if the event matches their event filter(s), then that event will be 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`.)

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

Producing Readable Backing MapListener Events from Distributed Caches

Backing `MapListener` events are returned from replicated caches in readable Java format. However, backing `MapListener` events returned from distributed caches are in internal Coherence format. The Coherence Incubator Common project provides an `AbstractMultiplexingBackingMapListener` class that enables you to obtain readable backing `MapListener` events from distributed caches. See <http://coherence.oracle.com/display/INCUBATOR/Coherence+Common> to download Coherence Common libraries.

To produce readable backing `MapListener` events from distributed caches:

1. Implement the `AbstractMultiplexingBackingMapListener` class.
2. Register the implementation in the `<listener>` section of the `backing-map-scheme` in the `cache-config` file.
3. Start the cache server application file and the client file with the `cacheconfig` Java property:

```
-Dtangosol.coherence.cacheconfig="cache-config.xml"
```

The `AbstractMultiplexingBackingMapListener` class provides an `onBackingMapEvent` method which you can override to specify how you would like the event returned.

The following listing of the `VerboseBackingMapListener` class is a sample implementation of `AbstractMultiplexingBackingMapListener`. The `onBackingMapEvent` method has been over-ridden to send the results to standard output.

Example 19–17 An AbstractMultiplexingBackingMapListener Implementation

```
import com.tangosol.net.BackingMapManagerContext;
import com.tangosol.util.MapEvent;

public class VerboseBackingMapListener extends
AbstractMultiplexingBackingMapListener {

    public VerboseBackingMapListener(BackingMapManagerContext context) {
        super(context);
    }

    @Override
    protected void onBackingMapEvent(MapEvent mapEvent, Cause cause) {

        System.out.printf("Thread: %s Cause: %s Event: %s\n",
Thread.currentThread().getName(), cause, mapEvent);

        try {
            Thread.currentThread().sleep(5000);
        } catch (InterruptedException e) {
            // add Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

```
}
```

[Example 19-18](#) is a listing of a sample `cache-config.xml` file. In the `<listener>` section of the file, the `VerboseBackingMapListener` is identified as being of type `com.tangosol.net.BackingMapManagerContext`.

Example 19-18 Cache Configuration Specifying a Verbose Backing Map Listener

```
<cache-config>

  <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>
  ...
</cache-config>
```


Advanced: Synchronous Event Listeners

Some events are delivered asynchronously, so that application listeners do not disrupt the cache services that are generating the events. In some rare scenarios, asynchronous delivery can cause ambiguity of the ordering of events compared to the results of ongoing operations. To guarantee that the cache API operations and the events are ordered as if the local view of the clustered system were single-threaded, a `MapListener` must implement the `SynchronousListener` marker interface.

One example in Coherence itself that uses synchronous listeners is the Near Cache, which can use events to invalidate locally cached data ("Seppuku").

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

Querying Data In a Cache

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

Query Overview

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

It should be noted that queries apply only to currently cached data (and will 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; in the case of dedicated `CacheServer` instances, this implies (usually) that application classes must be installed in the `CacheServer` classpath.

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

Query Concepts

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

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

Any "void argument" method can be used, including `Object` methods like `toString()` (useful for prototype/debugging). Indexes may be either traditional

"field indexes" (indexing fields of objects) or "functional indexes" (indexing "virtual" object attributes). For example, if a class has field accessors `getFirstName` and `getLastName`, the class may define a function `getFullName` which concatenates those names, and this function may be indexed. See ["Using Query Indexes"](#) on page 20-3 for more information on indexes.

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

Example 20–1 Equality Filter

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

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

Example 20–2 Filter that Constructs a ReflectionExtractor

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

[Example 20–3](#) illustrates a routine to select the entries of a cache that satisfy a particular filter:

Example 20–3 Selecting Cache Entries that Satisfy a Filter

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

[Example 20–4](#) illustrates using a filter to select and sort cache entries:

Example 20–4 Selecting and Sorting Cache Entries that Satisfy a Filter

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

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

Performing Simple Queries

[Example 20–5](#) demonstrates how to create a simple query and uses the `GreaterEqualsFilter` filter.

Example 20–5 Querying the Cache with a Filter

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

for (Iterator iter = cache.entrySet(filter).iterator(); iter.hasNext(); )
{
    Map.Entry entry = (Map.Entry) iter.next();
}
```

```
Integer key = (Integer) entry.getKey();
Person person = (Person) entry.getValue();
System.out.println("key=" + key + " person=" + person);
}
```

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

Note:

- The Partitioned Cache implements this method using the Parallel Query feature, which is only available in Coherence Enterprise Edition or higher. When working with a Partitioned Cache in Coherence Standard Edition, this method will retrieve the data set to the client for processing.
- Although queries can be executed through a near cache, the query will not use the front portion of a near cache. If using a near cache with queries, the best approach is to use the following sequence:

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

Using Query Indexes

Query indexes allow values (or attributes of those values) and corresponding keys to be correlated within a `QueryMap` to increase query performance. Indexes are a feature of Coherence Enterprise Edition or higher.

The following topics are included in this section:

- [Creating an Index](#)
- [Creating User-Defined Indexes](#)

Creating an Index

The `addIndex` method of the `QueryMap` class is used to create indexes. Any attribute able to be queried may be indexed using this method. The method includes three parameters:

```
addIndex(ValueExtractor extractor, boolean fOrdered, Comparator comparator)
```

[Example 20–6](#) demonstrates how to create an index:

Example 20–6 Sample Code to Create an Index

```
NamedCache cache = CacheFactory.getCache("MyCache");
ValueExtractor extractor = new ReflectionExtractor("getAttribute");
cache.addIndex(extractor, true, null);
```

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

The comparator argument can be used to 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) already exists. It is expected that an application will call this method to suggest an index even if the index may already exist, just so that the application is certain that index has been suggested. For example in a distributed environment, each server will likely suggest the same set of indexes when it starts, and there is no downside to the application blindly requesting those indexes regardless of whether another server has already requested the same indexes.

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

A list of applied indexes can be retrieved from the `StorageManagerMBean` by using JMX. For more information, see [Chapter 34, "How to Manage Coherence Using JMX"](#).

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 or not an entry should be indexed.

Implementing the MapIndex Interface

The `MapIndex` interface is used to correlate values stored in an indexed `Map` (or attributes of those values) to the corresponding keys in the indexed `Map`. Applications implement this interface to supply a custom index.

The following example implementation defines an index that only adds entries with non-null values. This would be useful in the case where there is a cache with a large number of entries and only a small subset have meaningful, non-null, values.

```
public class CustomMapIndex implements MapIndex
{
    public void insert(Map.Entry entry)
    {
        if (entry.getValue() != null)
        {
            ...
        }
    }
    ...
}
```

In the above example, the value of the entry is checked for `null` prior to 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 need to 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 or not 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 is not supported. For example:

```
ValueExtractor extractor = new ReflectionExtractor("getLastName");
Filter filter = new NotEqualsFilter("getId", null);
ValueExtractor condExtractor = new ConditionalExtractor(filter, extractor, true);

// add the conditional index which should only contain the last name values for
// the
// entries with non-null Ids
cache.addIndex(condExtractor, true, null);
```

Batching Queries and Memory Usage

In order to preserve memory on the client issuing a query, there are various techniques that can be used to retrieve query results in batches.

Using the `keySet` form of the queries – combined with `getAll()` – reduces memory consumption since the entire entry set is not deserialized on the client at once. It also takes advantage of near caching:

Example 20–7 Using a `keySet` Query Format

```
// keySet(Filter filter)
Set setKeys = cache.keySet(filter);
Set setPageKeys = new HashSet();
int PAGE_SIZE = 100;
for (Iterator iter = setKeys.iterator(); iter.hasNext();)
{
    setPageKeys.add(iter.next());
    if (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. [Example 20–8](#) demonstrates using a `LimitFilter`:

Example 20–8 Using a `Limit Filter`

```
int pageSize = 25;
Filter filter = new GreaterEqualsFilter("getAge", 18);
// get entries 1-25
Filter limitFilter = new LimitFilter(filter, pageSize);
Set entries = cache.entrySet(limitFilter);

// get entries 26-50
```



```
limitFilter.nextPage();
entries = cache.entrySet(limitFilter);
```

When using a distributed/partitioned cache, queries can be targeted to partitions and cache servers using a `PartitionedFilter`. This is the most efficient way of batching query results as each query request will be 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 one of 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 already 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 ...  
}
```

Queries Involving Multi-Value Attributes

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

Example 20–9 Querying on Multi-Value Attributes

```
Set searchTerms = new HashSet();  
searchTerms.add("java");  
searchTerms.add("clustering");  
searchTerms.add("books");  
  
// The cache contains instances of a class "Document" which has a method  
// "getWords" which returns a Collection<String> containing the set of  
// words that appear in the document.  
Filter filter = new ContainsAllFilter("getWords", searchTerms);  
  
Set entrySet = cache.entrySet(filter);  
  
// iterate through the search results  
// ...
```

ChainedExtractor

The `ChainedExtractor` implementation allows chained invocation of zero-argument (accessor) methods. In [Example 20–10](#), the extractor will first use reflection to call `getName()` on each cached `Person` object, and then use reflection to call `length()` on the returned `String`.

Example 20–10 Chaining Invocation Methods

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

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

Using Continuous Query Caching

While it is possible to obtain a point in time query result from a Coherence cache to, and it is possible to receive events that would change the result of that query, Coherence provides a feature that combines a query result with a continuous stream of related events to maintain an up-to-date query result in a real-time fashion. This capability is called *Continuous Query*, because it has the same effect as if the desired query had zero latency *and* the query were being executed several times every millisecond! For more information on point in time query results and events, see [Chapter 20, "Querying Data In a Cache."](#)

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

Uses of Continuous Query Caching

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

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

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

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

The Coherence Continuous Query Cache

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

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

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

Constructing a Continuous Query Cache

There are two items that define a Continuous Query Cache:

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

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

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

See [Appendix B, "Cache Configuration Elements"](#) for more information on specifying cache configuration information.

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

Example 21–1 A Query for a Continuous Query Cache

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

See [Chapter 20, "Querying Data In a Cache"](#) for more information on queries.

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

Example 21–2 Getting Data for the Continuous Query Cache

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

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

Example 21–3 Constructing the Continuous Query Cache

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

Cleaning up the resources associated with a ContinuousQueryCache

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

Caching only keys, or caching both keys and values

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

Example 21–4 A Constructor that Allows the CacheValues Property

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

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

Example 21–5 Setting the CacheValues Property

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

CacheValues Property and Event Listeners

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

Listening to the ContinuousQueryCache

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

Example 21–6 Adding a Listener to a Continuous Query Cache

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

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

Example 21–7 Processing Continuous Query Cache Entries and Adding a Listener

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

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

Example 21–8 Adding a Listener Before Processing Continuous Query Cache Entries

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

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

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

Example 21–9 Providing a Listener When Constructing the Continuous Query Cache

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

Achieving a Stable Materialized View

The `ContinuousQueryCache` implementation faced the same challenge: How to assemble an exact point-in-time snapshot of an underlying cache *while receiving a stream of modification events from that same cache*. The solution has several parts. First, Coherence supports an option for synchronous events, which provides a set of ordering guarantees. See [Chapter 19, "Using Cache Events,"](#) for more information on this option.

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

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

Support for Synchronous and Asynchronous Listeners

By default, listeners to the ContinuousQueryCache will have their events delivered asynchronously. However, the ContinuousQueryCache does respect the option for synchronous events as provided by the SynchronousListener interface. See [Chapter 19, "Using Cache Events,"](#) for more information on this option.

Making the ContinuousQueryCache Read-Only

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

Example 21–10 Making the Continuous Query Cache Read-Only

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

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

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

Processing Data In a Cache

Coherence provides the ideal infrastructure for building Data Grid services, and the client and server-based applications that use a Data Grid. At a basic level, Coherence can manage an immense amount of data across a large number of servers in a grid; it can provide close to zero latency access for that data; it supports parallel queries across that data; and it supports integration with database and EIS systems that act as the system of record for that data. Additionally, Coherence provides several services that are ideal for building effective data grids.

Note: All of the Data Grid capabilities described in the following sections are features of Coherence Enterprise Edition and higher.

Targeted Execution

Coherence provides for the ability to execute an agent against an entry in any map of data managed by the Data Grid:

```
map.invoke(key, agent);
```

In the case of partitioned data, the agent executes on the grid node that owns the data to execute against. This means that the queuing, concurrency management, agent execution, data access by the agent and data modification by the agent all occur on that grid node. (Only the synchronous backup of the resultant data modification, if any, requires additional network traffic.) For many processing purposes, it is much more efficient to move the serialized form of the agent (usually only a few hundred bytes, at most) than to handle distributed concurrency control, coherency and data updates.

For request/response processing, the agent returns a result:

```
Object oResult = map.invoke(key, agent);
```

In other words, Coherence as a Data Grid will determine the location to execute the agent based on the configuration for the data topology, move the agent there, execute the agent (automatically handling concurrency control for the item while executing the agent), back up the modifications if any, and return a result.

Parallel Execution

Coherence additionally provides for the ability to execute an agent against an entire collection of entries. In a partitioned Data Grid, the execution occurs in parallel, meaning that the more nodes that are in the grid, the broader the work is load-balanced across the Data Grid:

```
map.invokeAll(collectionKeys, agent);
```

For request/response processing, the agent returns one result for each key processed:

```
Map mapResults = map.invokeAll(collectionKeys, agent);
```

In other words, Coherence determines the optimal location(s) to execute the agent based on the configuration for the data topology, moves the agent there, executes the agent (automatically handling concurrency control for the item(s) while executing the agent), backing up the modifications if any, and returning the coalesced results.

Query-Based Execution

Coherence supports the ability to query across the entire data grid. For example, in a trading system it is possible to query for all open Order objects for a particular trader:

Example 22–1 Querying Across a Data Grid

```
NamedCache map    = CacheFactory.getCache("trades");
Filter           filter = new AndFilter(new EqualsFilter("getTrader", traderid),
                                         new EqualsFilter("getStatus", Status.OPEN));
Set setOpenTradeIds = mapTrades.keySet(filter);
```

By combining this feature with Parallel Execution in the data grid, Coherence provides for the ability to execute an agent against a query. As in the previous section, the execution occurs in parallel, and instead of returning the identities or entries that match the query, Coherence executes the agents against the entries:

```
map.invokeAll(filter, agent);
```

For request/response processing, the agent returns one result for each key processed:

```
Map mapResults = map.invokeAll(filter, agent);
```

In other words, Coherence combines its Parallel Query and its Parallel Execution together to achieve query-based agent invocation against a Data Grid.

Data-Grid-Wide Execution

Passing an instance of `AlwaysFilter` (or a null) to the `invokeAll` method will cause the passed agent to be executed against all entries in the `InvocableMap`:

```
map.invokeAll((Filter) null, agent);
```

As with the other types of agent invocation, request/response processing is supported:

```
Map mapResults = map.invokeAll((Filter) null, agent);
```

An application can process all the data spread across a particular map in the Data Grid with a single line of code.

Agents for Targeted, Parallel and Query-Based Execution

An agent implements the `EntryProcessor` interface, typically by extending the `AbstractProcessor` class.

Several agents are included with Coherence, including:

- `AbstractProcessor` - an abstract base class for building an `EntryProcessor`

- **ExtractorProcessor** - extracts and returns a specific value (such as a property value) from an object stored in an `InvocableMap`
- **CompositeProcessor** - bundles together a collection of `EntryProcessor` objects that are invoked sequentially against the same `Entry`
- **ConditionalProcessor** - conditionally invokes an `EntryProcessor` if a `Filter` against the `Entry`-to-process evaluates to true
- **PropertyProcessor** - an abstract base class for `EntryProcessor` implementations that depend on a `PropertyManipulator`
- **NumberIncrementor** - pre- or post-increments any property of a primitive integral type, and `Byte`, `Short`, `Integer`, `Long`, `Float`, `Double`, `BigInteger`, `BigDecimal`
- **NumberMultiplier** - multiplies any property of a primitive integral type, and `Byte`, `Short`, `Integer`, `Long`, `Float`, `Double`, `BigInteger`, `BigDecimal`, and returns either the previous or new value

The `EntryProcessor` interface (contained within the `InvocableMap` interface) contains only two methods:

Example 22–2 Methods in the `EntryProcessor` Interface

```
/**
 * An invocable agent that operates against the Entry objects within a
 * Map.
 */
public interface EntryProcessor
    extends Serializable
{
    /**
     * Process a Map Entry.
     *
     * @param entry the Entry to process
     *
     * @return the result of the processing, if any
     */
    public Object process(Entry entry);

    /**
     * Process a Set of InvocableMap Entry objects. This method is
     * semantically equivalent to:
     * <pre>
     * Map mapResults = new ListMap();
     * for (Iterator iter = setEntries.iterator(); iter.hasNext(); )
     * {
     *     Entry entry = (Entry) iter.next();
     *     mapResults.put(entry.getKey(), process(entry));
     * }
     * return mapResults;
     * </pre>
     *
     * @param setEntries a read-only Set of InvocableMap Entry objects to
     * process
     *
     * @return a Map containing the results of the processing, up to one
     * entry for each InvocableMap Entry that was processed, keyed
     * by the keys of the Map that were processed, with a
     * corresponding value being the result of the processing for
     * each key
     */
}
```

```
*/
public Map processAll(Set setEntries);
}
```

(The `AbstractProcessor` implements the `processAll` method as described in the previous example.)

The `InvocableMap.Entry` that is passed to an `EntryProcessor` is an extension of the `Map.Entry` interface that allows an `EntryProcessor` implementation to obtain the necessary information about the entry and to make the necessary modifications in the most efficient manner possible:

Example 22–3 *InvocableMap.Entry API*

```
/**
 * An InvocableMap Entry contains additional information and exposes
 * additional operations that the basic Map Entry does not. It allows
 * non-existent entries to be represented, thus allowing their optional
 * creation. It allows existent entries to be removed from the Map. It
 * supports several optimizations that can ultimately be mapped
 * through to indexes and other data structures of the underlying Map.
 */
public interface Entry
    extends Map.Entry
{
    // ----- Map Entry interface -----

    /**
     * Return the key corresponding to this entry. The resultant key does
     * not necessarily exist within the containing Map, which is to say
     * that InvocableMap.this.containsKey(getKey()) could return
     * false. To test for the presence of this key within the Map, use
     * {@link #isPresent}, and to create the entry for the key, use
     * {@link #setValue}.
     *
     * @return the key corresponding to this entry; may be null if the
     *         underlying Map supports null keys
     */
    public Object getKey();

    /**
     * Return the value corresponding to this entry. If the entry does
     * not exist, then the value will be null. To differentiate between
     * a null value and a non-existent entry, use {@link #isPresent}.
     * 

<b>Note:</b> any modifications to the value retrieved using this
     * method are not guaranteed to persist unless followed by a
     * {@link #setValue} or {@link #update} call.
     *
     * @return the value corresponding to this entry; may be null if the
     *         value is null or if the Entry does not exist in the Map
     */
    public Object getValue();

    /**
     * Store the value corresponding to this entry. If the entry does
     * not exist, then the entry will be created by invoking this method,
     * even with a null value (assuming the Map supports null values).
     *
     * @param oValue the new value for this Entry
     */
}


```

```

*
* @return the previous value of this Entry, or null if the Entry did
*         not exist
*/
public Object setValue(Object oValue);

// ----- InvocableMap Entry interface -----

/**
 * Store the value corresponding to this entry. If the entry does
 * not exist, then the entry will be created by invoking this method,
 * even with a null value (assuming the Map supports null values).
 * <p/>
 * Unlike the other form of {@link #setValue(Object) setValue}, this
 * form does not return the previous value, and consequently may be
 * significantly less expensive (in terms of cost of execution) for
 * certain Map implementations.
 *
 * @param oValue      the new value for this Entry
 * @param fSynthetic  pass true only if the insertion into or
 *                   modification of the Map should be treated as a
 *                   synthetic event
 */
public void setValue(Object oValue, boolean fSynthetic);

/**
 * Extract a value out of the Entry's value. Calling this method is
 * semantically equivalent to
 * <tt>extractor.extract(entry.getValue())</tt>, but this method may
 * be significantly less expensive because the resultant value may be
 * obtained from a forward index, for example.
 *
 * @param extractor  a ValueExtractor to apply to the Entry's value
 *
 * @return the extracted value
 */
public Object extract(ValueExtractor extractor);

/**
 * Update the Entry's value. Calling this method is semantically
 * equivalent to:
 * <pre>
 *   Object oTarget = entry.getValue();
 *   updater.update(oTarget, oValue);
 *   entry.setValue(oTarget, false);
 * </pre>
 * The benefit of using this method is that it may allow the Entry
 * implementation to significantly optimize the operation, such as
 * for purposes of delta updates and backup maintenance.
 *
 * @param updater  a ValueUpdater used to modify the Entry's value
 */
public void update(ValueUpdater updater, Object oValue);

/**
 * Determine if this Entry exists in the Map. If the Entry is not
 * present, it can be created by calling {@link #setValue} or
 * {@link #setValue}. If the Entry is present, it can be destroyed by
 * calling {@link #remove}.
 *

```

```
* @return true iff this Entry is existent in the containing Map
*/
public boolean isPresent();

/**
 * Remove this Entry from the Map if it is present in the Map.
 * <p/>
 * This method supports both the operation corresponding to
 * {@link Map#remove} and synthetic operations such as
 * eviction. If the containing Map does not differentiate between
 * the two, then this method will always be identical to
 * <tt>InvocableMap.this.remove(getKey())</tt>.
 *
 * @param fSynthetic pass true only if the removal from the Map
 *                  should be treated as a synthetic event
 */
public void remove(boolean fSynthetic);
}
```

Data Grid Aggregation

While the agent discussion in the previous section corresponds to *scalar* agents, the `InvocableMap` interface also supports aggregation:

Example 22–4 Aggregation in the `InvocableMap` API

```
/**
 * Perform an aggregating operation against the entries specified by the
 * passed keys.
 *
 * @param collKeys the Collection of keys that specify the entries within
 *                this Map to aggregate across
 * @param agent    the EntryAggregator that is used to aggregate across
 *                the specified entries of this Map
 *
 * @return the result of the aggregation
 */
public Object aggregate(Collection collKeys, EntryAggregator agent);

/**
 * Perform an aggregating operation against the set of entries that are
 * selected by the given Filter.
 * <p/>
 * <b>Note:</b> calling this method on partitioned caches requires a
 * Coherence Enterprise Edition (or higher) license.
 *
 * @param filter the Filter that is used to select entries within this
 *              Map to aggregate across
 * @param agent  the EntryAggregator that is used to aggregate across
 *              the selected entries of this Map
 *
 * @return the result of the aggregation
 */
public Object aggregate(Filter filter, EntryAggregator agent);
```

A simple `EntryAggregator` processes a set of `InvocableMap.Entry` objects to achieve a result:

Example 22–5 EntryAggregator API

```

/**
 * An EntryAggregator represents processing that can be directed to occur
 * against some subset of the entries in an InvocableMap, resulting in a
 * aggregated result. Common examples of aggregation include functions
 * such as min(), max() and avg(). However, the concept of aggregation
 * applies to any process that must evaluate a group of entries to
 * come up with a single answer.
 */
public interface EntryAggregator
    extends Serializable
{
    /**
     * Process a set of InvocableMap Entry objects to produce an
     * aggregated result.
     *
     * @param setEntries a Set of read-only InvocableMap Entry objects to
     *                  aggregate
     *
     * @return the aggregated result from processing the entries
     */
    public Object aggregate(Set setEntries);
}

```

For efficient execution in a Data Grid, an aggregation process must be designed to operate in a parallel manner.

Example 22–6 ParallelAwareAggregator API for running Aggregation in Parallel

```

/**
 * A ParallelAwareAggregator is an advanced extension to EntryAggregator
 * that is explicitly capable of being run in parallel, for example in a
 * distributed environment.
 */
public interface ParallelAwareAggregator
    extends EntryAggregator
{
    /**
     * Get an aggregator that can take the place of this aggregator in
     * situations in which the InvocableMap can aggregate in parallel.
     *
     * @return the aggregator that will be run in parallel
     */
    public EntryAggregator getParallelAggregator();

    /**
     * Aggregate the results of the parallel aggregations.
     *
     * @return the aggregation of the parallel aggregation results
     */
    public Object aggregateResults(Collection collResults);
}

```

Coherence comes with all of the natural aggregation functions, including:

- Count
- DistinctValues
- DoubleAverage

- DoubleMax
- DoubleMin
- DoubleSum
- LongMax
- LongMin
- LongSum

Note: All aggregators that come with Coherence are parallel-aware.

See the `com.tangosol.util.aggregator` package for a list of Coherence aggregators. To implement your own aggregator, see the `AbstractAggregator` abstract base class.

Node-Based Execution

Coherence provides an Invocation Service which allows execution of single-pass agents (called Invocable objects) anywhere within the grid. The agents can be executed on any particular node of the grid, in parallel on any particular set of nodes in the grid, or in parallel on all nodes of the grid.

An invocation service is configured using the `<invocation-scheme>` element in the cache configuration file. Using the name of the service, the application can easily obtain a reference to the service:

```
InvocationService service = CacheFactory.getInvocationService("agents");
```

Agents are simply runnable classes that are part of the application. An example of a simple agent is one designed to request a GC from the JVM:

Example 22-7 Simple Agent to Request Garbage Collection

```
/**
 * Agent that issues a garbage collection.
 */
public class GCAGENT
    extends AbstractInvocable
    {
        public void run()
        {
            System.gc();
        }
    }
```

To execute that agent across the entire cluster, it takes one line of code:

```
service.execute(new GCAGENT(), null, null);
```

Here is an example of an agent that supports a grid-wide request/response model:

Example 22-8 Agent to Support a Grid-Wide Request and Response Model

```
/**
 * Agent that determines how much free memory a grid node has.
 */
public class FreeMemAgent
```



```

        extends AbstractInvocable
    {
    public void run()
    {
        Runtime runtime = Runtime.getRuntime();
        int cbFree = runtime.freeMemory();
        int cbTotal = runtime.totalMemory();
        setResult(new int[] {cbFree, cbTotal});
    }
    }

```

To execute that agent across the entire grid and retrieve all the results from it, **it still takes only one line of code:**

```
Map map = service.query(new FreeMemAgent(), null);
```

While it is easy to do a grid-wide request/response, it takes a bit more code to print out the results:

Example 22–9 Printing the Results from a Grid-Wide Request or Response

```

Iterator iter = map.entrySet().iterator();
while (iter.hasNext())
{
    Map.Entry entry = (Map.Entry) iter.next();
    Member member = (Member) entry.getKey();
    int[] anInfo = (int[]) entry.getValue();
    if (anInfo != null) // nullif member died
        System.out.println("Member " + member + " has "
            + anInfo[0] + " bytes free out of "
            + anInfo[1] + " bytes total");
}

```

The agent operations can be stateful, which means that their invocation state is serialized and transmitted to the grid nodes on which the agent is to be run.

Example 22–10 Stateful Agent Operations

```

/**
 * Agent that carries some state with it.
 */
public class StatefulAgent
    extends AbstractInvocable
    {
    public StatefulAgent(String sKey)
    {
        m_sKey = sKey;
    }

    public void run()
    {
        // the agent has the key that it was constructed with
        String sKey = m_sKey;
        // ...
    }

    private String m_sKey;
    }

```

Work Manager

Coherence provides a grid-enabled implementation of the *CommonJ Work Manager*. Using a Work Manager, an application can submit a collection of work that must be executed. The Work Manager distributes that work in such a way that it is executed in parallel, typically across the grid. In other words, if there are ten work items submitted and ten servers in the grid, then each server will likely process one work item. Further, the distribution of work items across the grid can be tailored, so that certain servers (for example, one that acts as a gateway to a particular mainframe service) will be the first choice to run certain work items, for sake of efficiency and locality of data.

The application can then wait for the work to be completed, and can provide a timeout for how long it is willing to wait. The API for this purpose is quite powerful, allowing an application to wait for the first work item to complete, or for a specified set of the work items to complete. By combining methods from this API, it is possible to do things like "Here are 10 items to execute; for these 7 unimportant items, wait no more than 5 seconds, and for these 3 important items, wait no more than 30 seconds".

Example 22-11 Using a Work Manager

```
Work[] aWork = ...
Collection collBigItems = new ArrayList();
Collection collAllItems = new ArrayList();
for (int i = 0, c = aWork.length; i < c; ++i)
{
    WorkItem item = manager.schedule(aWork[i]);

    if (i < 3)
    {
        // the first three work items are the important ones
        collBigItems.add(item);
    }

    collAllItems.add(item);
}

Collection collDone = manager.waitForAll(collAllItems, 5000L);
if (!collDone.containsAll(collBigItems))
{
    // wait the remainder of 30 seconds for the important work to finish
    manager.waitForAll(collBigItems, 25000L);
}
```

Oracle Coherence Work Manager: Feedback from a Major Financial Institution

Our primary use case for the Work Manager is to allow our application to serve coarse-grained service requests using our blade infrastructure in a standards-based way. We often have what appears to be a simple request, like "give me this family's information." In reality, however, this request expands into a large number of requests to several diverse back-end data sources consisting of web services, RDMBS calls, and so on. This use case expands into two different but related problems that we are looking to the distributed version of the work manager to solve.

- How do we take a coarse-grained request that expands into several fine-grained requests and execute them in parallel to avoid blocking the caller for an unreasonable time? In the previous example, we may have to make upwards of 100 calls to various places to retrieve the information. Since Java EE has no legal threading model, and since the threading we observed when trying a

message-based approach to this was unacceptable, we decided to use the Coherence Work Manager implementation.

- Given that we want to make many external system calls in parallel while still leveraging low-cost blades, we are hoping that fanning the required work across many dual processor (logically 4-processor because of hyperthreading) machines allows us to scale an inherently vertical scalability problem with horizontal scalability at the hardware level. We think this is reasonable because the cost to marshall the request to a remote Work Manager instance is small compared to the cost to execute the service, which usually involves dozens or hundreds of milliseconds.

Managing Map Operations with Triggers

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

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

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

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

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

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

`MapTrigger` functionality is typically added as part of an application start-up process. It can be added programmatically as described in the `MapTrigger` API, or it can be configured using the `class-factory` mechanism in the `coherence-cache-config.xml` configuration file. In this case, a `MapTrigger` will be registered during the very first `CacheFactory.getCache(...)` call for the corresponding cache. [Example 23-1](#) assumes that the `createMapTrigger` method would return a new `MapTriggerListener(new MyCustomTrigger())`;

Example 23–1 Creating a MapTriggerListener in the coherence-cache-config.xml File

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

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

The `MapTriggerListener` is a special purpose `MapListener` implementation that is used to register a `MapTrigger` with a corresponding `NamedCache`. In [Example 23–2](#), `MapTriggerListener` is used to register the `PersonMapTrigger` with the `People` named cache.

Example 23–2 A MapTriggerListener Registering a MapTrigger with a Named Cache

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

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

A Map Trigger Example

The code in [Example 23–3](#) illustrates a map trigger and how it can be called. In the `PersonMapTrigger` class in [Example 23–3](#), the `process` method is implemented to modify an entry before it is placed in the map. In this case, the last name attribute of a `Person` object is converted to upper case characters. The object is then returned to the entry.

Example 23–3 A MapTrigger Class

```
...

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

```

    {
    }

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

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

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

            entry.setValue(person);
        }
    }

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

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

    public int hashCode()
    {
        return getClass().getName().hashCode();
    }
}

```

The MapTrigger in [Example 23–4](#), calls the PersonMapTrigger. The new MapTriggerListener passes the PersonMapTrigger to the People NamedCache.

Example 23–4 Calling a MapTrigger and Passing it to a Named Cache

...

```

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

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

        return new MapTriggerListener(trigger);
    }
}

```

```
    }

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

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

Using Coherence Query Language

This chapter describes how to use Coherence Query Language (CohQL) to interact with Coherence caches. CohQL is a new light-weight syntax (in the tradition of SQL) that is used to perform cache operations on a Coherence cluster. The language can be used either programmatically or from a command-line tool.

The following sections are included in this chapter:

- [Understanding Coherence Query Language Syntax](#)
- [Using the CohQL Command-Line Tool](#)
- [Building Filters in Java Programs](#)
- [Additional Coherence Query Language Examples](#)

Note:

- Although the CohQL syntax may appear similar to SQL, it is important to remember that the syntax is *not* SQL and is actually more contextually related to the Java Persistence Query Language (JPQL) standard.
 - CQL (Continuous Query Language) is a query language related to Complex Event Processing (CEP) and should not be confused with CohQL.
-

Understanding Coherence Query Language Syntax

The following sections describe the functionality provided by CohQL. Each section describes a particular statement, its syntax, and an example. You can find more query examples in "[Additional Coherence Query Language Examples](#)" on page 24-15.

Note: The current release of CohQL does not support subqueries.

The following topics are included in this section:

- [Query Syntax Basics](#)
- [Retrieving Data](#)
- [Managing the Cache Lifecycle](#)
- [Working with Cache Data](#)
- [Working with Indexes](#)

■ Issuing Multiple Query Statements

For reference, [Table 24–1](#) lists the Coherence query statements, clauses, and expressions in alphabetical order.

Table 24–1 Coherence Query Language Statements

For this statement, clause, or expression...	See this section
BACKUP CACHE	"Writing a Serialized Representation of a Cache to a File"
bind variables	"Using Bind Variables"
CREATE CACHE	"Creating a Cache"
CREATE INDEX	"Creating an Index on the Cache"
DELETE	"Deleting Entries in the Cache"
DROP CACHE	"Removing a Cache from the Cluster"
DROP INDEX	"Removing an Index from the Cache"
ENSURE CACHE	"Creating a Cache"
ENSURE INDEX	"Creating an Index on the Cache"
GROUP BY	"Aggregating Query Results"
INSERT	"Inserting Entries in the Cache"
key() pseudo-function	"Using Key and Value Pseudo-Functions"
path-expressions	"Using Path-Expressions"
RESTORE CACHE	"Loading Cache Contents from a File"
SELECT	"Retrieving Data from the Cache"
SOURCE	"Processing Query Statements in Batch Mode"
UPDATE	"Changing Existing Values"
value() pseudo-function	"Using Key and Value Pseudo-Functions"
WHERE	"Filtering Entries in a Result Set"

Query Syntax Basics

This section describes some of the building blocks of the syntax, such as path expressions, bind variables, and pseudo-functions.

Using Path-Expressions

One of the main building blocks of CohQL are path-expressions. Path expressions are used to navigate through a graph of object instances. An identifier in a path expression is used to represent a property in the Java Bean sense. It is backed by a `ReflectionExtractor` that is created by prepending a `get` and capitalizing the first letter. Elements are separated by the "dot" (.) character, that represents object traversal. For example the following path expression is used to navigate an object structure:

```
a.b.c
```

It reflectively invokes these methods:

```
getA().getB().getC()
```

Using Bind Variables

For programmatic uses, the API passes strings to a simple set of query functions. Use bind variables if you want to pass the value of variables and do not want to engage 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 will be "supplied" before the query is run. The syntax for this form is: `?n` where *n* can be any number. Positional bind variables can be used by the `QueryHelper` class in the construction of filters. For example:

```
QueryHelper.createFilter("number = ?1" , new Object[]{new Integer(42)});
```

- the colon (:)—Enter a colon, immediately followed by the identifier to be used as a named place holder for the object to be supplied as a key value pair. The syntax for this is: `:identifier` where *identifier* is an alpha-numeric combination, starting with an alphabetic character. Named bind variables can be used by the `QueryHelper` class in the construction of filters. For example:

```
HashMap env = new HashMap();
env.put("iNum",new Integer(42));
QueryHelper.createFilter("number = :iNum" , env);
```

See ["Building Filters in Java Programs"](#) on page 24-14 for more information on the `QueryHelper` class and constructing filters programmatically.

Using Key and Value Pseudo-Functions

CohQL provides a `key()` pseudo-function because many users store objects with a key property. The `key()` represents the cache's key. The query syntax also provides a `value()` pseudo-function. The `value()` is implicit in chains that do not start with `key()`. The `key()` and `value()` pseudo-functions are typically used in `WHERE` clauses, where they test against the key or value of the cache entry. For examples of using `key()` and `value()`, see ["Key and Value Pseudo-Function Examples"](#) on page 24-17 and ["A Command-Line Example"](#) on page 24-12.

Using Aliases

Although not needed semantically, CohQL supports aliases in order 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 need 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 will throw an error if it encounters an *unquoted* literal argument or filename that contains an offending character.

Retrieving Data

The following sections describe the `SELECT` statement and the `WHERE` clause. These entities are the basic building blocks of most cache queries.

Retrieving Data from the Cache

The `SELECT` statement is the basic building block of a query: it indicates that you want to retrieve data from the cache. The clause can take several forms, including simple and complex path expressions, key expressions, transformation functions, multiple expressions, and aggregate functions. The `SELECT` statement also supports the use of aliases.

The form of the `SELECT` statement is as follows:

```
SELECT (properties* aggregators* | * | alias)
FROM "cache-name" [[AS] alias]
[WHERE conditional-expression] [GROUP [BY] properties+]
```

The asterisk (*) character represents the full object instead of subparts. It is not required to prefix a path with the `cache-name`. The `FROM` part of the `SELECT` statement targets the cache that forms the domain over which the query should draw its results. The `cache-name` is the name of an existing cache.

See ["Simple SELECT * FROM Statements that Highlight Filters"](#) on page 24-16 for additional examples.

Example:

- Select all of the items from the cache `dept`.

```
select * from "dept"
```

Filtering Entries in a Result Set

Use the `WHERE` clause to filter the entries returned in a result set. One of the key features of CohQL is that they can use path expressions to navigate object structure during expression evaluation. Conditional expressions can use a combination of logical operators, comparison expressions, primitive and function operators on fields, and so on.

In the literal syntax of the `WHERE` clause, use single quotes to enclose string literals; they can be escaped within a string by prefixing the quote with another single quote. Numeric expressions are defined according to the conventions of the Java programming language. Boolean values are represented by the literals `TRUE` and `FALSE`. Date literals are not supported.

Note: CohQL does not have access to type information. This means that 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 "%" wild-cards.

The IN operator can be used to 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 be used to 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 allows you to create standalone Coherence filters based on the WHERE clause conditional-expression syntax. See ["Building Filters in Java Programs"](#) on page 24-14.

See ["Simple SELECT * FROM Statements that Highlight Filters"](#) on page 24-16 for additional examples.

Example:

- Select all of the items in the cache dept where the value of the deptno key equals 10.

```
select * from "dept" where deptno = 10
```

Managing the Cache Lifecycle

The following sections describe how to create and remove caches. They also describe how to backup and restore cache contents.

Creating a Cache

Before you begin sending queries, you must first either connect to an existing cache or create a new cache. To do this, you can use either the CREATE CACHE or ENSURE CACHE statement. This statement will first attempt to connect to a cache with the specified *cache-name*. If the cache is not found in the cluster, Coherence attempts to create a new cache with the specified name based on the current cache configuration file. This statement is especially useful on the command line. If you are using this statement in a program, you have the option of specifying service and classloader

information instead of a name (classloaders cannot be accessed from the command line).

Note: Cache names and service names must be enclosed in quotes (either double-quotes (" ") or single-quotes (' ')) in a statement.

The syntax is:

```
[ CREATE | ENSURE ] CACHE "cache-name"  
[ SERVICE "service-name" ]
```

Example:

- Create a cache named dept.

```
create cache "dept"
```

Writing a Serialized Representation of a Cache to a File

Use the BACKUP CACHE statement to write a serialized representation of the given cache to a file represented by the given *filename*. The *filename* is an operating system-dependent path and must be enclosed in single or double quotes. The BACKUP CACHE statement is available only in the command-line tool. The syntax is:

```
BACKUP CACHE "cache-name" [ TO ] [ FILE ] "filename"
```

Note: The backup (and subsequent restore) functionality is designed for use in a development and testing environment and should not be used on a production data set as it makes no provisions to ensure data consistency. It is not supported as a production backup, snapshot, or checkpointing utility.

In particular:

- The backup is slow since it only operates on a single node in the cluster.
 - The backup is not atomic. That is, it will miss changes to elements which occur during the backup and will result in a dirty read of the data.
 - The backup stops if an error occurs and results in an incomplete backup. In such scenarios, an `IOException` is thrown that describes the error.
-

Example:

- Write a serialized representation of the cache dept to the file `textfile`.

```
backup cache "dept" to file "textfile"
```

Loading Cache Contents from a File

Use the RESTORE CACHE statement to read a serialized representation of the given cache from a file represented by the given *filename*. The *filename* is an operating system-dependent path and must be enclosed in single or double quotes. The RESTORE CACHE statement is available only in the command-line tool. The syntax is:

```
RESTORE CACHE "cache-name" [ FROM ] [ FILE ] "filename"
```

Example:

- Restore the cache `dept` from the file `textfile`.

```
restore cache "dept" from file "textfile"
```

Removing a Cache from the Cluster

Use the `DROP CACHE` statement to remove the specified cache completely from the cluster. The cache is removed by a call to the Java `destroy()` method. If any cluster member holds a reference to the dropped cache and tries to perform any operations on it, then the member will receive an `IllegalStateException`. The syntax for the Coherence query `DROP CACHE` statement is:

```
DROP CACHE "cache-name"
```

Example:

- Remove the cache `orders` from the cluster.

```
drop cache "orders"
```

Working with Cache Data

The following sections describe how to work with data in the cache, such as inserting and deleting cache data and filtering result sets.

Aggregating Query Results

An aggregate query is a variation on the `SELECT` query. Use an aggregate query when you want to group results and apply aggregate functions to obtain summary information about the results. A query is considered an aggregate query if it uses an aggregate function or possesses a `GROUP BY` clause. The most typical form of an aggregate query involves the use of one or more grouping expressions followed by aggregate functions in the `SELECT` clause paired with the same lead grouping expressions in a `GROUP BY` clause.

CohQL supports these aggregate functions: `COUNT`, `AVG`, `MIN`, `MAX`, and `SUM`.

See ["Complex Queries that Feature Projection, Aggregation, and Grouping"](#) on page 24-17 for additional examples.

Example:

- Select the total amount and average price for items from the `orders` cache, grouped by supplier.

```
select supplier,sum(amount),avg(price) from "orders" group by supplier
```

Changing Existing Values

Use the `UPDATE` statement to change an existing value in the cache. The syntax is:

```
UPDATE "cache-name" [[AS] alias]
SET update-statement {, update-statement}*
[ WHERE conditional-expression ]
```

Each *update-statement* consists of a path expression, assignment operator (`=`), and an expression. The expression choices for the assignment statement are restricted. The right side of the assignment must resolve to a literal, a bind-variable, a static method, or a new Java-constructor with only literals or bind-variables. The `UPDATE` statement also supports the use of aliases.

See ["UPDATE Examples"](#) on page 24-17 for additional examples.

Example:

- For employees in the `employees` cache whose ranking is above grade 7, update their salaries to 1000 and vacation hours to 200.

```
update "employees" set salary = 1000, vacation = 200 where grade > 7
```

Inserting Entries in the Cache

Use the `INSERT` statement to store the given `VALUE` under the given `KEY`. If the `KEY` clause is not provided, then the newly created object will be sent the message `getKey()`, if possible. Otherwise, the value object will be used as the key.

Note that the `INSERT` statement operates on Maps of Objects. The syntax is:

```
INSERT INTO "cache-name"
[ KEY (literal | new java-constructor | static method) ]
VALUE (literal | new java-constructor | static method)
```

Example:

- Insert the key `writer` with the value `David` into the `employee` cache.

```
insert into "employee" key "writer" value "David"
```

Deleting Entries in the Cache

Use the `DELETE` statement to delete specified entries in the cache. The syntax is:

```
DELETE FROM "cache-name" [[AS] alias]
[WHERE conditional-expression]
```

The `WHERE` clause for the `DELETE` statement functions the same as it would for a `SELECT` statement. All *conditional-expressions* are available to filter the set of entities to be removed. The `DELETE` statement also supports the use of aliases.

Be Careful: If the `WHERE` clause is not present, then all entities in the given cache are removed.

Example:

- Delete the entry from the cache `employee` where `bar.writer` key is not `David`.

```
delete from "employee" where bar.writer IS NOT "David"
```

Working with Indexes

The following sections describe how to create and remove indexes on cache data. Indexes are a powerful tool that allows Coherence's built-in optimizer to more quickly and efficiently analyze queries and return results.

Creating an Index on the Cache

Use the `CREATE INDEX` or the `ENSURE INDEX` statement to create indexes on an identified cache. The syntax is:

```
[ CREATE | ENSURE ] INDEX [ON] "cache-name" (value-extractor-list)
```

The *value-extractor-list* is a comma-separated list that uses path expressions to create `ValueExtractors`. If more than one element exists, then a `MultiExtractor`

is used. If you want to create a `KeyExtractor`, then start the path expression with a `key()` pseudo-function.

Natural ordering for the index is assumed.

Example:

- Create an index on the attribute `lastname` in the `orders` cache.

```
create index "orders" lastname
```

Removing an Index from the Cache

The `DROP INDEX` statement removes the index based on the given `ValueExtractor`. This statement is available only for the command-line tool. The syntax is:

```
DROP INDEX [ON] "cache-name" (value-extractor-list)
```

Example:

- Remove the index on the `lastname` attribute in the `orders` cache.

```
drop index "orders" lastname
```

Issuing Multiple Query Statements

The following section describes how to more efficiently issue multiple query statements to the cache.

Processing Query Statements in Batch Mode

The `SOURCE` statement allows for the "batch" processing of statements. The `SOURCE` statement opens and reads one or more query statements from a file represented by the given *filename*. The *filename* is an operating system-dependent path and must be enclosed in single or double quotes. Each query statement in the file must be separated by a semi-colon (;) 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 line only, a "period" symbol (.) can be used as an abbreviation for '@' but must not 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
```

Using the CohQL Command-Line Tool

The CohQL command-line tool provides a non-programmatic way to interact with caches by allowing statements to be issued from the command line. The tool can be run using the `com.tangosol.coherence.dslquery.QueryPlus` class or, for convenience, a startup script is available to run the tool and is located in the `COHERENCE_HOME/bin/` directory. The script is available for both Windows (`query.cmd`) and Unix (`query.sh`).

The script starts a cluster node in console mode; that is, storage is not enabled on the node. This is the suggested setting for production environments and assumes that the node will join a cluster that contains storage-enabled cache servers. However, a storage-enabled node can be created for testing by changing the `storage_enabled` setting in the script to `true`.

Note: As configured, the startup script uses the default operational configuration file (`tangosol-coherence.xml`) and the default cache configuration file (`coherence-cache-config.xml`) that are located in the `coherence.jar` when creating/joining a cluster and configuring caches. For more information on configuring Coherence, see [Chapter 3, "Understanding Configuration."](#)

The script also provides the option for setting the `COHERENCE_HOME` environment variable as well as the `JLINE_HOME` environment variable. If `COHERENCE_HOME` is not already set on the computer, set it in the script to the location where Coherence was installed. The `JLINE_HOME` environment variable should be set to the location of the JLine JAR, which is used 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.

Out-of-Box, the script expects the `jline-0.9.94.jar` library to be located in the `COHERENCE_HOME/bin/` directory. If it is not found, a message displays indicating that the JAR was not loaded. Copy the JAR to the `COHERENCE_HOME/bin/` directory, or modify the `JLINE_HOME` variable to point to the location of the JAR.

The JLine JAR can be downloaded from the following location.

<http://jline.sourceforge.net/>

Starting the Command-line Tool

The following procedure demonstrates how to start the CohQL command-line tool using the startup script and assumes that the `storage_enabled` setting in the script is set to `false` (the default):

1. Start a cache server cluster node or ensure that an existing cache server cluster node is started.

To start a cache server cluster node, open a command prompt or shell and execute the cache server startup script that is located in the `/bin` directory:

`cache-server.cmd` on the Windows platform or `cache-server.sh` for UNIX platforms. The cache server starts and output is emitted that provides information about this cluster member.

2. Open a command prompt or shell and execute the CohQL command-line startup script that is located in the `/bin` directory: `query.cmd` on the Windows platform or `query.sh` for UNIX platforms. Information about the Java environment displays. The command-line tool prompt (`CohQL>`) is returned.

Note: When joining an existing cache server node, modify the startup script to use the same cluster settings as the existing cache server node, including the same cache configuration.

3. Enter `help` at the prompt to view the complete command-line help. Enter `commands` to list the help without detailed descriptions.

Typically, your first statement will be to create a new cache or connect to an existing cache. See ["Creating a Cache"](#) on page 24-5.

["A Command-Line Example"](#) on page 24-12 illustrates a series of query statements that exercise the command-line tool.

Using Command-Line Tool Arguments

The CohQL command-line tool includes a set of arguments that are read and executed before the `CohQL>` prompt returns. This is useful when using the script as part of a larger script—for example, as part of a build process or to pipe input/output. Enter `help` at the `CohQL>` prompt to view help for the arguments within the command-line tool.

Table 24–2 Coherence Query Language Command-Line Tool Arguments

Argument	Description
<code>-t</code>	enable trace mode to print debug information.
<code>-c</code>	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.
<code>-s</code>	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 (<code><myInput</code>) and standard output (<code>>myOutput</code>).
<code>-e</code>	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.
<code>-l statement</code>	Execute the given statement. Statements must be enclosed in single or double quotes. Any number of <code>-l</code> arguments can be used.
<code>-f filename</code>	Process the statements in the given file. The statements in the file must be separated by a semi-colon (;). The file is an operating system-dependent path and must be enclosed in single or double quotes. Any number of <code>-f</code> arguments can be used.

Examples

Return all entries in the `contact` cache and print the entries to the standard out then exit the command-line tool.

```
query.sh -c -l "select * from contact"
```

Return all entries in the `dist-example` cache and print the entries (suppressing extra verbiage) to the file named `myOutput` then exit the command-line tool.

```
query.cmd -s -c -l "select * from 'dist-example'" >myOutput
```

Process all the segments in the file named `myStatements` then exit the command-line tool.

```
query.sh -c -f myStatements
```

Read the commands from the `myInput` file and print the output (suppressing extra verbiage) to the file named `myOutput`.

```
query.sh -s <myInput >myOutput
```

A Command-Line Example

[Example 24-1](#) illustrates a simple example that exercises the command-line tool on Windows. This example is intended to test statements against a local cache, so the `storage_enabled` setting in the startup script is set to `true`. The example illustrates creating and dropping a cache, storing and retrieving entries, and restoring the cache from a backup file. It also highlights the use of the `key()` and `value()` pseudo-functions.

When you start `query.cmd` at the command prompt, information about the Java environment, the Coherence version and edition, and Coherence cache server is displayed. You then receive a prompt (`CohQL>`) where you can enter your query statements.

Annotations that describe the commands and their output have been added to the example in bold-italic font. Here is an example:

< This is an annotation. >

Example 24-1 A Command-Line Query Exercise

```
C:/coherence/bin/query.cmd
** Starting storage enabled console **
java version "1.6.0_14"
Java(TM) SE Runtime Environment (build 1.6.0_14-b08)
Java HotSpot(TM) Server VM (build 14.0-b16, mixed mode)

2010-01-27 16:54:07.501/0.265 Oracle Coherence 3.6.0.0 Internal <Info> (thread=main, member=n/a):
Loaded operational configuration from
"jar:file:/C:/coherence360/coherence/lib/coherence.jar!/tangosol-coherence.xml"
2010-01-27 16:54:07.501/0.265 Oracle Coherence 3.6.0.0 Internal <Info> (thread=main, member=n/a):
Loaded operational overrides from
"jar:file:/C:/coherence360/coherence/lib/coherence.jar!/tangosol-coherence-override-dev.xml"
2010-01-27 16:54:07.501/0.265 Oracle Coherence 3.6.0.0 Internal <D5> (thread=main, member=n/a):
Optional configuration override "/tangosol-coherence-override.xml" is not specified
2010-01-27 16:54:07.517/0.281 Oracle Coherence 3.6.0.0 Internal <D5> (thread=main, member=n/a):
Optional configuration override "/custom-mbeans.xml" is not specified

Oracle Coherence Version 3.6.0.0 Internal Build 0
Grid Edition: Development mode
Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.

2010-01-27 16:54:09.173/1.937 Oracle Coherence GE 3.6.0.0 Internal <D5> (thread=Cluster,
member=n/a): Service Cluster joined the cluster with senior service member n/a
2010-01-27 16:54:12.423/5.187 Oracle Coherence GE 3.6.0.0 Internal <Info> (thread=Cluster,
member=n/a): Created a new cluster "cluster:0xC4DB" with Member(Id=1, Timestamp=2010-01-27
16:54:08.032, Address=130.35.99.213:8088, MachineId=49877,
Location=site:us.oracle.com,machine:tpfaeffl-lap7,process:4316, Role=TangosolCoherenceQueryPlus,
Edition=Grid Edition, Mode=Development, CpuCount=2, SocketCount=1)
UID=0x822363D500000126726BBBA0C2D51F98
2010-01-27 16:54:12.501/5.265 Oracle Coherence GE 3.6.0.0 Internal <D5>
(thread=Invocation:Management, member=1): Service Management joined the cluster with senior service
member 1
```

```

    < Create a cache named "employees". >
CohQL> create cache "employees"

2010-01-27 16:54:26.892/19.656 Oracle Coherence GE 3.6.0.0 Internal <Info> (thread=main, member=1):
Loaded cache configuration from
"jar:file:/C:/coherence360/coherence/lib/coherence.jar!/coherence-cache-config.xml"
2010-01-27 16:54:27.079/19.843 Oracle Coherence GE 3.6.0.0 Internal <D5> (thread=DistributedCache,
member=1): Service DistributedCache joined the cluster with senior service member 1
2010-01-27 16:54:27.095/19.859 Oracle Coherence GE 3.6.0.0 Internal <D5> (thread=DistributedCache,
member=1): Service DistributedCache: sending Config
Sync to all
Result

    < Insert an entry (key-value pair) into the cache. >
CohQL> insert into "employees" key "David" value "ID-5070"

    < Change the value of the key. >
CohQL> update employees set value() = "ID-5080" where key() like "David"
Result
David, true

    < Retrieve the values in the cache. >
CohQL> select * from "employees"
Result
ID-5080

    < Retrieve the value of a key that does not exist. An empty result set is returned >
CohQL> select key(), value() from "employees" where key() is "Richard"
Result

    < Delete an existing key in the cache. An empty result set is returned. >
CohQL> delete from employees where key() = "David"
Result

    < Delete the contents of the employees cache. An empty result set is returned. >
CohQL> delete from "employees"
Result

    < Destroy the employees cache. >
CohQL> drop cache "employees"

    < Re-create the employees cache. >
CohQL> create cache "employees"
Result

    < Insert more entries into the cache. >
CohQL> insert into "employees" key "David" value "ID-5080"

CohQL> insert into "employees" key "Julie" value "ID-5081"

CohQL> insert into "employees" key "Mike" value "ID-5082"

CohQL> insert into "employees" key "Mary" value "ID-5083"

    < Retrieve the keys and value in the employees cache. >
CohQL> select key(), value() from "employees"
Result
Julie, ID-5081
Mike, ID-5082

```

Mary, ID-5083
David, ID-5080

< Save a serialized representation of the cache in a file. >

CohQL> backup cache "employees" to "emp.bkup"

< Delete a key from the cache. >

CohQL> delete from "employees" where key() = "David"

Result

< Retrieve the cache contents again, notice that the deleted key and value are not present. >

CohQL> select key(), value() from "employees"

Result

Julie, ID-5081

Mike, ID-5082

Mary, ID-5083

< Delete the contents of the cache. >

CohQL> delete from "employees"

Result

< Retrieve the contents of the cache. An empty result set is returned. >

CohQL> select * from "employees"

Result

< Restore the cache contents from the backup file. >

CohQL> restore cache "employees" from file "emp.bkup"

< Retrieve the cache contents. Note that all of the entries have been restored and returned. >

CohQL> select key(), value() from "employees"

Result

Julie, ID-5081

Mike, ID-5082

Mary, ID-5083

David, ID-5080

< Destroy the employees cache. >

CohQL> drop cache "employees"

< Exit the command-line tool. >

CohQL> bye

Building Filters in Java Programs

The `FilterBuilder` API is a string-oriented way to filter a result set from within a Java program, without having to remember details of the Coherence API. To do this, 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 (described in ["Filtering Entries in a Result Set"](#) on page 24-4), but without the literal `WHERE`. The forms that take an `Object` array or `Map` are for passing objects that are referenced by bind variables. Each form constructs a filter from the provided Coherence query string.

- `public static Filter createFilter(String s)`—where `s` is a `String` in the Coherence query representing a `Filter`.

- `public static Filter createFilter(String s, Object[] aBindings)`—where `s` is a String in the Coherence query representing a Filter and `aBindings` is an array of Objects to use for bind variables.
- `public static Filter createFilter(String s, Map bindings)`—where `s` is a String in the Coherence query representing a Filter and `bindings` is a Map of Objects to use for bind variables.
- `public static Filter createFilter(String s, Object[] aBindings, Map bindings)`—where `s` is a String in the Coherence query representing a Filter, `aBindings` is an array of Objects to use for bind variables, and `bindings` is a Map of Objects to use for bind variables.

These factory methods will throw an `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 allows you to create value extractors when building filters. The extractor is used to both extract values (for example, for sorting or filtering) from an object, and to provide an identity for that extraction. The following example demonstrates using `createExtractor` when creating an index:

```
cache.addIndex(QueryHelper.createExtractor("key().lastName"), /*fOrdered*/ true,
/*comparator*/ null);
```

Additional Coherence Query Language Examples

This section provides additional examples and shows their equivalent Coherence API calls with instantiated Objects (Filters, ValueExtractors, Aggregators, and so on). The simple `select *` examples that highlight Filters can be used to understand the translation for `FilterBuilder` API if you focus only on the Filter part. The full set of examples can be used to understand the translation for the `QueryBuilder` API and the command-line tool.

The examples use an abbreviated form of the path syntax where the cache name to qualify an identifier is dropped.

The Java language form of the examples also use `ReducerAggregator` instead of `EntryProcessors` for projection. Note also that the use of `KeyExtractor` should no longer be needed given changes to `ReflectionExtractor` in Coherence 3.5.

Simple SELECT * FROM Statements that Highlight Filters

- Select the items from the cache orders where 40 is greater than the value of the price key.

```
select * from "orders" where 40 > price
```
- Select the items from the cache orders where the value of the price key exactly equals 100, and the value of insurance key is less than 10 or the value of the shipping key is greater than or equal to 20.

```
select * from "orders" where price is 100 and insurance < 10 or shipping >= 20
```
- Select the items from the cache orders where the value of the price key exactly equals 100, and *either* the value of insurance key is less than 10 or the value of the shipping key is greater than or equal to 20.

```
select * from "orders" where price is 100 and (insurance < 10 or shipping >= 20)
```
- Select the items from the cache orders where either the value of the price key equals 100, or the bar key equals 20.

```
select * from "orders" where price = 100 or shipping = 20
```
- Select the items from the cache orders where the value of the insurance key is not null.

```
select * from "orders" where insurance is not null
```
- Select the items from the cache employees where the emp_id key has a value between 1 and 1000 or the bar.emp key is not "Smith".

```
select * from "employees" where emp_id between 1 and 1000 or bar.emp is not "Smith"
```
- Select items from the cache orders where the value of item key is similar to the value "coat".

```
select * from "orders" where item like "coat%"
```
- Select items from the cache employees where the value of emp_id is in the set 5, 10, 15, or 20.

```
select * from "employees" where emp_id in (5,10,15,20)
```
- Select items from the cache employees where emp_id contains the list 5, 10, 15, and 20.

```
select * from "employees" where emp_id contains (5,10,15,20)
```
- Select items from the cache employees where emp_id contains the all of the items 5, 10, 15, and 20.

```
select * from "employees" where emp_id contains all (5,10,15,20)
```
- Select items from the cache employees where emp_id contains any of the items 5, 10, 15, or 20.

```
select * from "employees" where emp_id contains any (5,10,15,20)
```
- Select items from cache employees where the value of foo key is less than 10 and occurs in the set 10, 20.


```
select * from "employees" where emp_id < 10 in (10,20)
```

Complex Queries that Feature Projection, Aggregation, and Grouping

- Select the home state and age of employees in the cache `ContactInfoCache`, and group by state and age.


```
select homeAddress.state, age, count() from "ContactInfoCache" group by
homeAddress.state, age
```
- Select the spurious `frobit` key from the `orders` cache. Note, an empty result set will be returned.


```
select frobit,supplier,sum(amount),avg(price) from "orders" group by supplier
```
- For the items in the `orders` cache that are greater than \$1000, select the items, their prices and colors.


```
select item_name,price,color from "orders" where price > 1000
```
- Select the total amount for items from the `orders` cache.


```
select sum(amount) from "orders"
```
- Select the total amount for items from the `orders` cache where the color attribute is red or green.


```
select sum(amount) from "orders" where color is "red" or color is "green"
```
- Select the total amount and average price for items from the `orders` cache


```
select sum(amount),avg(price) from "orders"
```
- Select one copy of the `lastname` and `city` from possible duplicate rows from the `employees` cache, where the state is California.


```
select distinct lastName,city from "employees" where state = "CA"
```

UPDATE Examples

- For employees in the `employees` cache whose ranking is above grade 7, increase their salaries by 10% and add 50 hours of vacation time.


```
update "employees" set salary = salary*1.10, vacation = vacation + 50 where
grade > 7
```

Key and Value Pseudo-Function Examples

This section provides examples of how to use the `key()` and `value()` pseudo-functions. For additional examples, see ["A Command-Line Example"](#) on page 24-12.

- Select the employees from the `ContactInfoCache` whose home address is in Massachusetts, but work out of state.


```
select key().firstName, key().lastName from "ContactInfoCache"
homeAddress.state is 'MA' and workAddress.state != "MA"
```
- Select the employees from the `ContactInfoCache` cache whose age is greater than 42.


```
select key().firstName, key().lastName, age from "ContactInfoCache" where age >
42
```

Performing Transactions

This chapter provides instructions for using Coherence's transaction and data concurrency features. Users should be familiar with transaction principles before reading this chapter. In addition, the Coherence Resource Adapter requires knowledge of J2EE Connector Architecture (J2CA), Java Transaction API (JTA) and Java EE deployment.

The following topics are included in this chapter:

- [Overview of Transactions](#)
- [Using Explicit Locking for Data Concurrency](#)
- [Using Entry Processors for Data Concurrency](#)
- [Using the Transaction Framework API](#)
- [Using the Coherence Resource Adapter](#)

Overview of Transactions

Transactions ensure correct outcomes in systems that undergo state changes by allowing a programmer to scope a number of state changes into a single 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 or not 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 25–1](#) summarizes the various transactions option that Coherence offers.

Table 25–1 Coherence Transaction Options

Option Name	Description
Explicit locking	The <code>ConcurrentMap</code> interface (which is extended by the <code>NamedCache</code> interface) supports explicit locking operations. The locking API guarantees data concurrency but does not offer atomic guarantees. For detailed information on this option, see "Using Explicit Locking for Data Concurrency" on page 25-2.
Entry Processors	Coherence also supports a lock-free programming model through the <code>EntryProcessor</code> API. For many transaction types, this minimizes contention and latency and improves system throughput, without compromising the fault-tolerance of data operations. This option offers high-level concurrency control but does not offer atomic guarantees. For detailed information on this option, see "Using Entry Processors for Data Concurrency" on page 25-3.
Transaction Framework API	Coherence Transaction Framework API is a connection-based API that provides atomic transaction guarantees across partitions and caches even in the event of a client failure. The framework supports the use of <code>NamedCache</code> operations, queries, aggregation, and entry processors within the context of a transaction. For detailed information on this option, see "Using the Transaction Framework API" on page 25-5.
Coherence Resource Adapter	The Coherence resource adapter leverages the Coherence Transaction Framework API and allows Coherence to participate as a resource in XA transactions that are managed by a JavaEE container's transaction manager. This transaction option offers atomic guarantees. For detailed information on this option, see "Using the Coherence Resource Adapter" on page 25-21.

Using Explicit Locking for Data Concurrency

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

Example 25–1 Applying Locking Operations on a Cache

```
...
NamedCache cache = CacheFactory.getCache("dist-cache");
Object key = "example_key";
cache.lock(key, -1);
try
{
    Object value = cache.get(key);
    // application logic
    cache.put(key, value);
}
finally
{
    // Always unlock in a "finally" block
    // to ensure that uncaught exceptions
    // don't leave data locked
}
```

```

        cache.unlock(key);
    }
    ...

```

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

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

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

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

Using Entry Processors for Data Concurrency

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

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

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

`InvocableMap` provides three methods of starting `EntryProcessors`:

- Invoke an `EntryProcessor` on a specific key. Note that the key need not exist in the cache to invoke an `EntryProcessor` on it.
- Invoke an `EntryProcessor` on a collection of keys.
- Invoke an `EntryProcessor` on a `Filter`. In this case, the `Filter` will be executed against the cache entries. Each entry that matches the `Filter` criteria will have the `EntryProcessor` executed against it. For more information on `Filters`, see [Chapter 20, "Querying Data In a Cache"](#).

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

Note: `EntryProcessor` classes must be available in the classpath for each cluster node.

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

Example 25–2 Concurrency Control without Using `EntryProcessors`

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

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

// begin processing

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

// end processing
```

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

Example 25–3 Concurrency Control Using `EntryProcessors`

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

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

// begin processing

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

```

// end processing

...

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

```

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

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

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

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 in the event of client failure. The framework utilizes its own concurrency strategy and storage implementation as well as its own recovery manager for failed transactions.

Note: The TransactionMap API has been deprecated and is superseded by the Transaction Framework API. The two APIs are mutually exclusive.

Known Limitations

The Transaction Framework API has the following limitations:

- Database Integration – For existing Coherence users, the most noticeable limitation is the lack of support for database integration as compared to the existing

Partitioned NamedCache implementation. This functionality is currently being developed and is anticipated in a future release.

- Server-Side Functionality – Transactional caches do not support eviction or expiry, though they will support garbage collection of older object versions. Backing map listeners, triggers, and CacheStore modules are not supported.
- Explicit Locking and Pessimistic Transactions – Pessimistic/explicit locking (ConcurrentMap interface) are not supported.
- Filters – Filters, such as PartitionedFilter, LimitFilter and KeyAssociationFilter, are not supported.
- Synchronous Listener – The SynchronousListener interface is not supported.
- Near Cache – Wrapping a near cache around a transactional cache is not supported.
- Key Partitioning Strategy – You cannot specify a custom KeyPartitioningStrategy for a transactional cache; although, KeyAssociation or a custom KeyAssociator will work.

The following topics are included in this section:

- [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](#)

The Transaction Framework API is also the underlying transaction framework for the Coherence JCA resource adapter. For details on using the resource adapter, see ["Using the Coherence Resource Adapter"](#) on page 25-21.

Defining Transactional Caches

Transactional caches are specialized distributed caches that provide transactional guarantees. Transactional caches are required whenever performing a transaction using the Transaction Framework API. Transactional caches are not interoperable with non-transactional caches.

At run-time, transactional caches are automatically used together with a set of internal transactional caches that provide transactional storage and recovery. Transactional caches also allow default transaction behavior (including the default behavior of the internal transactional caches) to be overridden at run-time.

Transactional caches are defined within a cache configuration file using a `<transactional-scheme>` element. A transaction scheme includes many of the same elements and attributes that are available to a distributed cache scheme. For detailed information about the `<transactional-scheme>` element and all its subelements, see ["transactional-scheme"](#) on page B-105.

Note: The use of transaction schemes within near cache schemes is currently not supported.

The following example demonstrates defining a transactional cache scheme in a cache configuration file. The cache is named `MyTxCache` and maps to a `<transactional-scheme>` that is named `example-transactional`. The cache name can also use the `tx-*` convention which allows multiple cache instances to use a single mapping to a transactional cache scheme.

Note:

- The `<service-name>` element, as shown in the example below, is optional. If no `<service-name>` element is included in the transactional cache scheme, `TransactionalCache` is used as the default service name. In this case, applications must connect to a transactional service using the default service name. See ["Creating Transactional Connections"](#) on page 25-10.
-

Example 25-4 Example Transactional Cache Definition

```
<cache-mapping>
  <cache-name>MyTxCache</cache-name>
  <scheme-name>example-transactional</scheme-name>
</cache-mapping>
...
</cache-mapping>

<cache-schemes>
<!-- Transactional caching scheme. -->
  <transactional-scheme>
    <scheme-name>example-transactional</scheme-name>
    <service-name>TransactionalCache</service-name>
    <autostart>true</autostart>
    <thread-count>10</thread-count>
    <request-timeout>30000</request-timeout>
  </transactional-scheme>
  ...
</cache-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`:

```
<cache-mapping>
  <cache-name>tx-*</cache-name>
  <scheme-name>example-transactional</scheme-name>
</cache-mapping>
...
</cache-mapping>

<cache-schemes>
  <transactional-scheme>
    <scheme-name>example-transactional</scheme-name>
    <scheme-ref>base-transactional</scheme-ref>
```

```
<thread-count>10</thread-count>
</transactional-scheme>

<transactional-scheme>
  <scheme-name>base-transactional</scheme-name>
  <service-name>TransactionalCache</service-name>
  <request-timeout>30000</request-timeout>
  <autostart>true</autostart>
</transactional-scheme>
...
</caching-schemes>
```

Performing Cache Operations within a Transaction

Applications perform cache operations within a transaction in one of three ways:

- [Using the NamedCache API](#) – Applications use the `NamedCache` API to implicitly perform cache operations within a transaction.
- [Using the Connection API](#) – Applications use the `Connection` API to explicitly perform cache operations within a transaction.
- [Using the Coherence Resource Adapter](#) – Java EE applications use the Coherence Resource Adapter to connect to a Coherence data cluster and perform cache operations as part of a distributed (global) transaction.

Using the NamedCache API

The `NamedCache` API can be used to 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. This means each operation is immediately committed when it successfully completes; multiple operations cannot be scoped into a single transaction. Applications that require more control over transactional behavior must use the `Connection` API. See ["Using Transactional Connections"](#) on page 25-11 for a detailed description of a transaction's default behaviors.

The `NamedCache` API approach is ideally suited for ensuring atomicity guarantees when performing single operations such as `putAll`. The following example demonstrates a simple client that creates a `NamedCache` instance and uses the `CacheFactory.getCache()` method to get a transactional cache. The example uses the transactional cache that was defined in [Example 25-4](#). The client performs a `putAll` operation that is only committed if all the `put` operations succeed. The transaction is automatically rolled back if any `put` operation fails.

```
...
String key = "k";
String key2 = "k2";
String key3 = "k3";
String key4 = "k4";

CacheFactory.ensureCluster();
NamedCache cache = CacheFactory.getCache("MyTxCache");

Map map = new HashMap();
map.put(key, "value");
map.put(key2, "value2");
map.put(key3, "value3");
```

```
map.put(key4, "value4");

//operations performed on the cache are atomic
cache.putAll(map);

CacheFactory.shutdown();
...
```

Using the Connection API

The `Connection` API is used to perform cache operations within a transaction and provides the ability to explicitly control transaction behavior. For example, applications can enable or disable auto-commit mode or change transaction isolation levels.

The examples in this section demonstrate how to use the `Connection` interface, `DefaultConnectionFactory` class, and the `OptimisticNamedCache` interface which are located in the `com.tangosol.coherence.transaction` package. The examples use the transactional cache that was defined in [Example 25-4](#). The `Connection` API is discussed in detail following the examples.

[Example 25-5](#) demonstrates an auto-commit transaction; where, two insert operations are each executed as separate transactions.

Example 25-5 Performing an Auto-Commit Transaction

```
...
Connection con = new DefaultConnectionFactory().
    createConnection("TransactionalCache");

OptimisticNamedCache cache = con.getNamedCache("MyTxCache");

cache.insert(key, value);
cache.insert(key2, value2);

con.close();
...
```

[Example 25-6](#) demonstrates a non auto-commit transaction; where, two insert operations are performed within a single transaction. Applications that use non auto-commit transactions must manually demarcate transaction boundaries.

Example 25-6 Performing a Non Auto-Commit Transaction

```
...
Connection con = new DefaultConnectionFactory().
    createConnection("TransactionalCache");

con.setAutoCommit(false);

try
{
    OptimisticNamedCache cache = con.getNamedCache("MyTxCache");

    cache.insert(key, value);
    cache.insert(key2, value2);
    con.commit();
}
catch (Exception e)
```

```
        {
            con.rollback();
            throw e;
        }

    finally
    {
        con.close();
    }
    ...
```

[Example 25–7](#) demonstrates performing a transaction that spans multiple caches. Each transactional cache must be defined in a cache configuration file.

Example 25–7 Transaction Across Multiple Caches

```
...
Connection con = new DefaultConnectionFactory().
    createConnection("TransactionalCache");

con.setAutoCommit(false);
OptimisticNamedCache cache = con.getNamedCache("MyTxCache");
OptimisticNamedCache cache1 = con.getNamedCache("MyTxCache1");

cache.insert(key, value);
cache1.insert(key2, value2);

con.commit();

con.close();
...
```

Note: Transactions can span multiple partitions and/or 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 25–5](#) demonstrates creating a `Connection` instance using the factory's no argument constructor:

```
Connection con = new DefaultConnectionFactory().
    createConnection("TransactionalCache");
```

In this example, the first cache configuration file found on the classpath (or specified using the `-Dtangosol.coherence.cacheconfig` system property) is used by this `Connection` instance. Optionally, a URI can be passed as an argument to the factory class that specifies the location and name of a cache configuration file. For example, the following code demonstrates constructing a connection factory that uses a cache configuration file named `cache-config.xml` that is located in a `config` directory found on the classpath.

```
Connection con = new DefaultConnectionFactory("config/cache-config.xml").
    createConnection("TransactionalCache");
```

The `DefaultConnectionFactory` class provides methods for creating connections:

- `createConnection()` – The no-argument method creates a connection that is a member of the default transactional service, which is named `TransactionalCache`. Use the no-argument method when the `<transactional-scheme>` element being used does not include a specific `<service-name>` element. For details on defining transactional cache schemes and specifying the service name, see ["Defining Transactional Caches"](#) on page 25-6.
- `createConnection(ServiceName)` – This method creates a connection that is a member of a transactional service. The service name is a `String` that indicates the transactional service to which this connection belongs. The `ServiceName` maps to a `<service-name>` element that is defined within a `<transactional-scheme>` element in the cache configuration file. If no service name is used, the default name (`TransactionalCache`) is used as the service name. For details on defining transactional cache schemes and specifying the service name, see ["Defining Transactional Caches"](#) on page 25-6.
- `createConnection(ServiceName, loader)` – This method also creates a connection that is a member of a transactional service. In addition, it specifies the class loader for which the configuration should be used. In the above example, the connection is created by only specifying a service name; in which case, the default class loader is used.

Using Transactional Connections

The `com.tangosol.coherence.transaction.Connection` interface represents a logical connection to a Coherence service. An active connection is always associated with a transaction. A new transaction implicitly starts when a connection is created and also when a transaction is committed or rolled back.

Transactions that are derived from a connection have several default behaviors that are listed below. The default behaviors balance ease-of-use with performance.

- A transaction is automatically committed or rolled back for each cache operation. See ["Using Auto-Commit Mode"](#) below.
- A transaction uses the read committed isolation level. See ["Setting Isolation Levels"](#) below.
- A transaction immediately performs operations on the cache. See ["Using Eager Mode"](#) below.
- A transaction has a default timeout of 300 seconds. See ["Setting Transaction Timeout"](#) below.

A connection's default behaviors can be changed using the `Connection` instance's methods as required.

Using Auto-Commit Mode

Auto-commit mode allows an application to choose whether each cache operation should be associated with a separate transaction or whether multiple cache operations should be executed as a single transaction. Each cache operation is executed in a distinct transaction when auto-commit is enabled; the framework automatically commits or rolls back the transaction after an operation completes and then the connection is associated with a new transaction and the next operation is performed. By default, auto-commit is enabled when a `Connection` instance is created.

The following code from [Example 25-5](#) demonstrates `insert` operations that are each performed as a separate transaction:

```
OptimisticNamedCache cache = con.getNamedCache("MyTxCache");

cache.insert(key, value);
cache.insert(key2, value2);
```

Multiple operations are performed as part of a single transaction by disabling auto-commit mode. If auto-commit mode is disabled, an application must manually demarcate transaction boundaries. The following code from [Example 25-6](#) demonstrates insert operations that are performed within a single transaction:

```
con.setAutoCommit(false);

OptimisticNamedCache cache = con.getNamedCache("MyTxCache");

cache.insert(key, value);
cache.insert(key2, value2);

con.commit();
```

An application cannot use the `commit()` or `rollback()` method when auto-commit mode is enabled. Moreover, if auto-commit mode is enabled while in an active transaction, any work is automatically rolled back.

Setting Isolation Levels

Isolation levels help control data concurrency and consistency. The Transaction Framework uses implicit write-locks and does not implement read-locks. Any attempt to write to a locked entry results in an `UnableToAcquireLockException`; the request will not block. When a transaction is set to eager mode, the exception is thrown immediately. In non-eager mode, exceptions may not be thrown until the statement is flushed, which is typically at the next read or when the transaction commits. See ["Using Eager Mode"](#) on page 25-13.

The Coherence Transaction Framework API supports the following isolation levels:

- `READ_COMMITTED` – This is the default isolation level if no level is specified. This isolation level guarantees that only committed data is visible and doesn't 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. This means that a read is guaranteed to return a version that is equal to or greater than any version that was previously encountered while using the connection. Due to the monotonic 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. This means that a read is guaranteed to return a version that is equal to or greater than any version that was previously encountered while using the connection. Due to the monotonic 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. This means that if a transaction writes and commits a value, then immediately reads the same value in the next transaction with one of the consistent read isolation levels, the updated value may not be immediately visible. If reading the most recent value is critical, then the `READ_COMMITTED` isolation level should be used.

Isolation levels are set on a `Connection` instance and must be set prior to 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 as well as 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 25-5](#) demonstrates getting a transactional cache called `MyTxCache` and performs operations on the cache. For this example, a transactional cache that is named `MyTxCache` must be located in the cache configuration file at run-time. For details on defining a transactional cache, see ["Defining Transactional Caches"](#) on page 25-6.

```
OptimisticNamedCache cache = con.getNamedCache("MyTxCache");  
  
cache.insert(key, value);  
cache.insert(key2, value2);
```

Configuring POF When Performing Transactions

Transactional caches support Portable Object Format (POF) serialization within transactions. POF is enabled within a transactional cache scheme using the `<serializer>` element. The following example demonstrates enabling POF serialization in a transactional cache scheme.

```
<transactional-scheme>  
  <scheme-name>example-transactional</scheme-name>  
  <service-name>TransactionalCache</service-name>  
  <autostart>true</autostart>  
  <serializer>  
    <class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>  
  </serializer>  
</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 runtime, 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>  
  <user-type-list>  
    <include>txn-pof-config.xml</include>  
  ...
```

```
</pof-config>
```

See ["Combining Multiple POF Configuration Files"](#) on page 3-9 for more information on using the `<include>` element to combine POF configuration files.

Configuring Transactional Storage Capacity

The Transaction Framework API stores transactional data in internal distributed caches that utilize 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. Once 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 will slightly exceed the high-water mark before the storage eviction policy is notified.
 - It is possible that storage for a transactional cache will exceed 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 need to be increased so that it is larger than the size of the current data set. Otherwise, the eviction policy will be notified on every write once 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 will ever be read. When using consistent reads, the high-water mark should be high enough to provide for a sufficient number of historical versions. The formulas shown below can be used 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>
...
</transactional-scheme>
```

The following formulas provide a rough estimate of the memory usage for a row in a transactional cache.

For insert operations:

- Primary – key(serialized) + key (on-heap size) + value(serialized) + 1095 bytes constant overhead
- Backup – key(serialized) + value(serialized) + 530 bytes constant overhead

For updated operations:

- Primary – value(serialized) + 685 bytes constant overhead
- Backup – value(serialized) + 420 bytes constant overhead

Performing Transactions from Java Extend Clients

The Transaction Framework API provides Java extend clients with the ability to perform cache operations within a transaction. In this case, the transaction API is used within an entry processor that is located on the cluster. At run time, the entry processor is executed on behalf of the Java client.

The instructions in this section do not include detailed instructions on how to setup and use Coherence*Extend. For those new to Coherence*Extend, see "Setting Up Coherence*Extend" in *Oracle Coherence Client Guide*. For details on performing transactions from C++ or .NET clients, see "Performing Transactions for C++ Clients" and "Performing Transactions for .NET Clients" in the *Oracle Coherence Client Guide*.

The following topics are included in this section and are required to perform transactions from Java extend clients:

- [Create an Entry Processor for Transactions](#)
- [Configure the Cluster-Side Transaction Caches](#)
- [Configure the Client-Side Remote Cache](#)
- [Use the Transactional Entry Processor from a Java Client](#)

Create an Entry Processor for Transactions

Transactions are performed using the transaction API within an entry processor that resides on the cluster. The entry processor is executed on behalf of a Java extend client.

[Example 25–8](#) demonstrates an entry processor that performs a simple update operation within a transaction. At run time, the entry processor must be located on both the client and cluster.

Example 25–8 Entry Processor for Extend Client Transaction

```
public class MyTxProcessor extends AbstractProcessor
{
    public Object process(InvocableMap.Entry entry)
    {
        // obtain a connection and transaction cache
        ConnectionFactory connFactory = new DefaultConnectionFactory();
        Connection conn = connFactory.createConnection("TransactionalCache");
        OptimisticNamedCache cache = conn.getNamedCache("MyTxCache");

        conn.setAutoCommit(false);

        // get a value for an existing entry
        String sValue = (String) cache.get("existingEntry");
```

```

// create predicate filter
Filter predicate = new EqualsFilter(IdentityExtractor.INSTANCE, sValue);

try
{
    // update the previously obtained value
    cache.update("existingEntry", "newValue", predicate);
}
catch (PredicateFailedException e)
{
    // value was updated after it was read
    conn.rollback();
    return false;
}
catch (UnableToAcquireLockException e)
{
    // row is being updated by another transaction
    conn.rollback();
    return false;
}
try
{
    conn.commit();
}
catch (RollbackException e)
{
    // transaction was rolled back
    return false;
}
return true;
}
}

```

Configure the Cluster-Side Transaction Caches

Transactions require a transactional cache to be defined in the cluster-side cache configuration file. For details on defining a transactional cache, see ["Defining Transactional Caches"](#) on page 25-6.

The following example defines a transactional cache that is named `MyTxCache`, which is the cache name that was used by the entry processor in [Example 25-8](#). The example also includes a proxy scheme and a distributed cache scheme that are required to execute the entry processor from a remote client. The proxy is configured to accept client TCP/IP connections on localhost at port 9099.

```

<cache-config>
  <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>
<task-timeout>0</task-timeout>
<thread-count>7</thread-count>
<autostart>true</autostart>
<high-units>15M</high-units>
</transactional-scheme>
<distributed-scheme>
  <scheme-name>example-distributed</scheme-name>
  <service-name>DistributedCache</service-name>
  <backing-map-scheme>
    <local-scheme/>
  </backing-map-scheme>
  <autostart>true</autostart>
</distributed-scheme>
<proxy-scheme>
  <service-name>ExtendTcpProxyService</service-name>
  <thread-count>5</thread-count>
  <acceptor-config>
    <tcp-acceptor>
      <local-address>
        <address>localhost</address>
        <port>9099</port>
      </local-address>
    </tcp-acceptor>
  </acceptor-config>
  <autostart>true</autostart>
</proxy-scheme>
</caching-schemes>
</cache-config>
```

Configure the Client-Side Remote Cache

Remote clients require a remote cache in order 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.

```
<cache-config>
  <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>
        </tcp-initiator>
      </initiator-config>
    </remote-cache-scheme>
  </caching-schemes>
</cache-config>
```

```

        <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 runtime, 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 25–8](#).

```

NamedCache cache = CacheFactory.getCache("dist-example");
Object oReturn = cache.invoke("AnyKey", new MyTxProcessor());

System.out.println("Result of extend tx execution: " + oReturn);

```

Viewing Transaction Management Information

The transaction framework leverages the existing Coherence JMX management framework. For detailed information on enabling and using the Coherence JMX management framework, see [Chapter 34, "How to Manage Coherence Using JMX."](#)

This section describes two MBeans that can be used to view transaction information: `CacheMBean` and `TransactionManagerMBean`.

CacheMBeans for Transactional Caches

The `CacheMBean` managed resource provides attributes and operations for all caches, including transactional caches. Many of the MBeans attributes are not applicable to transactional cache; invoking such attributes simply returns a -1 value. A cluster node may have zero or more instances of cache managed beans for transactional caches. The object name uses the form:

```

type=Cache, service=service name, name=cache name,
nodeId=cluster node's id

```

[Table 25–2](#) describes the `CacheMBean` attributes that are supported for transactional caches.

Table 25–2 Transactional Cache Supported Attributes

Attribute	Type	Description
AverageGetMillis	Double	The average number of milliseconds per <code>get()</code> invocation
AveragePutMillis	Double	The average number of milliseconds per <code>put()</code> 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 will prune itself automatically once it reaches its maximum unit level. This is often referred to as the high water mark of the cache.

Table 25–2 (Cont.) Transactional Cache Supported Attributes

Attribute	Type	Description
Size	Integer	The number of entries in the current data set
TotalGets	Long	The total number of <code>get ()</code> operations since the cache statistics were last reset.
TotalGetsMillis	Long	The total number of milliseconds spent on <code>get ()</code> operations since the cache statistics were last reset.
TotalPuts	Long	The total number of <code>put ()</code> operations since the cache statistics were last reset.
TotalPutsMillis	Long	The total number of milliseconds spent on <code>put ()</code> operations since the cache statistics were last reset.

For transactional caches, the `resetStatistics` operation is supported and resets all transaction manager statistics.

TransactionManagerBean

The `TransactionManagerMBean` managed resource is specific to the transactional framework. It provides global transaction manager statistics 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 will only be incremented on the MBean on the node that coordinated the commit of that transaction.

Table 25–3 describes `TransactionManager` attributes.

Table 25–3 TransactionManagerMBean Attributes

Attribute	Type	Description
TotalActive	Long	The total number of currently active transactions. An active transaction is counted as any transaction that contains at least one modified entry and has yet to be committed or rolled back. Note that the count is maintained at the coordinator node for the transaction, even though multiple nodes may have participated in the transaction.
TotalCommitted	Long	The total number of transactions that have been committed by the Transaction Manager since the last time the statistics were reset. Note that the count is maintained at the coordinator node for the transaction being committed, even though multiple nodes may have participated in the transaction.

Table 25–3 (Cont.) TransactionManagerMBean Attributes

Attribute	Type	Description
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 will only apply to transactional connections obtained after the value is set. This attribute is currently not supported.

The TransactionManagerMBean includes a single operation called `resetStatistics`, which resets all transaction manager statistics.

Using the Coherence Resource Adapter

Coherence includes a J2EE Connector Architecture (J2CA) 1.5 compliant resource adaptor that is used to get connections to a Coherence cache. The resource adapter leverages the connection API of the Coherence Transaction Framework and therefore provides default transaction guarantees. In addition, the resource adapter provides full XA support which allows Coherence to participate in global transactions. A global transaction is unit of work that is managed by one or more resource managers and is controlled and coordinated by an external transaction manager, such as the transaction manager that is included with WebLogic server or OC4J.

The resource adapter is packaged as a standard Resource Adaptor Archive (RAR) and is named `coherence-transaction.rar`. The resource adapter is located in `COHERENCE_HOME/lib` and can be deployed to any Java EE container compatible with J2CA 1.5. The resource adapter includes proprietary resource adapter deployment descriptors for WebLogic (`weblogic-ra.xml`) and OC4J (`oc4j-ra.xml`) and can be deployed to these platforms without modification. Check your application server vendor's documentation for details on defining a proprietary resource adapter descriptor that can be included within the RAR.

Note: Coherence continues to include the `coherence-tx.rar` resource adapter for backwards compatibility. However, it is strongly recommended that applications use the `coherence-transaction.rar` resource adapter which provides full XA support. Those accustomed to using the Coherence `CacheAdapter` class can continue to do so with either resource adapter. See ["Using the Coherence Cache Adapter for Transactions"](#) on page 25-27.

The following topics are included in this section:

- [Performing Cache Operations within a Transaction](#)
- [Packaging the Application](#)
- [Using the Coherence Cache Adapter for Transactions](#)

Performing Cache Operations within a Transaction

Java EE application components (Servlets, JSPs, and EJBs) use the Coherence resource adapter to perform cache operations within a transaction. The resource adapters supports both local transactions and global transactions. Local transactions are used to perform cache operations within a transaction that is only scoped to a Coherence cache and cannot participate in a global transaction. Global transactions are used to perform cache operations that automatically commit or roll back based on the outcome of multiple resources that are enlisted in the transaction.

Like all JavaEE application components, the Java Naming and Directory Interface (JNDI) API is used to lookup the resource adapter's connection factory. The connection factory is then used to get logical connections to a Coherence cache.

The following examples demonstrate how to use the Coherence resource adapter to perform cache operations within a global transaction. [Example 25–9](#) is an example of using Container Managed Transactions (CMT); where, the container ensures that all methods execute within the scope of a global transaction. [Example 25–10](#) is an example of user-controlled transactions; where, the application component uses the Java Transaction API (JTA) to manually demarcate transaction boundaries.

Transactions require a transactional cache scheme to be defined within a cache configuration file. These examples use the transactional cache that was defined in [Example 25–4](#).

Example 25–9 Performing a Transaction When Using CMT

```
Context initCtx = new InitialContext();
ConnectionFactory cf = (ConnectionFactory)
    initCtx.lookup("java:comp/env/eis/CoherenceTxCF");

Connection con = cf.createConnection("TransactionalCache");

try
{
    OptimisticNamedCache cache = con.getNamedCache("MyTxCache");

    cache.delete("key1", null);
    cache.insert("key1", "value1");
}
finally
{
    con.close();
}
```

Example 25–10 Performing a User-Controlled Transaction

```
Context initCtx = new InitialContext();
ConnectionFactory cf = (ConnectionFactory)
    initCtx.lookup("java:comp/env/eis/CoherenceTxCF");

UserTransaction ut = (UserTransaction) new
    InitialContext().lookup("java:comp/UserTransaction");
ut.begin();
```



```

Connection con = cf.createConnection("TransactionalCache");

try
{
    OptimisticNamedCache cache = con.getNamedCache("MyTxCache");

    cache.delete("key1", null);
    cache.insert("key1", "value1");
    ut.commit();
}

catch (Exception e)
{
    ut.rollback();
    throw e;
}

finally
{
    con.close();
}

```

Creating a Coherence Connection

Applications use the `com.tangosol.coherence.ConnectionFactory` interface to create connections to a Coherence cluster. An instance of this interface is obtained using a JNDI lookup. The following code sample from [Example 25-10](#) performs a JNDI lookup for a connection factory that is bound to the `java:comp/env/eis/CoherenceTxCF` namespace:

```

Context initCtx = new InitialContext();
ConnectionFactory cf = (ConnectionFactory)
    initCtx.lookup("java:comp/env/eis/CoherenceTxCF");

```

The `ConnectionFactory` is then used to create a `com.tangosol.coherence.transaction.Connection` instance. The `Connection` instance represents a logical connection to a Coherence service:

```

Connection con = cf.createConnection("TransactionalCache");

```

The `createConnection(ServiceName)` method creates a connection that is a member of a transactional service. The service name is a `String` that indicates which transactional service this connection belongs to and must map to a service name that is defined in a `<transactional-scheme>` within a cache configuration file. For details on defining transactional cache schemes and specifying the service name, see ["Defining Transactional Caches"](#) on page 25-6.

A `Connection` instance always has an associated transaction which is scoped within the connection. A new transaction is started when a transaction is completed. The following default behaviors are associated with a connection. For more information on the `Connection` interface and changing the default settings, see ["Using Transactional Connections"](#) on page 25-11.

- Connections are in auto-commit mode by default which means that each statement is executed in a distinct transaction and when the statement completes the transaction is committed and the connection is associated with a new transaction.

Note: When the connection is used in the context of a global transaction, auto-commit mode is disabled and cannot be enabled. This means all cache operations are performed in a single transaction and either commit or roll back as a unit. In addition, the `Connection` interface's `commit` and `rollback` methods cannot be used if the connection is enlisted in a global transaction.

- The connection's isolation level is set to `READ_COMMITTED`. This means that 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 in the context of 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 25–10](#) demonstrates getting a named cache called `MyTxCache` and performing operations on the cache. The cache must be defined in the cache configuration file.

```
try
{
    OptimisticNamedCache cache = con.getNamedCache("MyTxCache");

    cache.delete("key1", null);
    cache.insert("key1", "value1");
}
```

Note: `OptimisticNamedCache` does not extend any operations from the `ConcurrentMap` interface since it uses its own locking strategy.

Demarcating Transaction Boundaries

Application components that perform user-controlled transactions use a JNDI lookup to get a JTA `UserTransaction` interface instance. The interface provides methods for demarcating the transaction. The following code sample from [Example 25–10](#) demonstrates getting a `UserTransaction` instance and demarcating the transaction boundaries:

```

UserTransaction ut = (UserTransaction) new
    InitialContext().lookup("java:comp/UserTransaction");

ut.begin();
Connection con = cf.createConnection("TransactionalCache");

try
{
    OptimisticNamedCache cache = con.getNamedCache("MyTxCache");

    cache.delete("key1", null);
    cache.insert("key1", "value1");
    ut.commit();
}

```

The above code demonstrates a typical scenario where the connection and the named cache exist within the transaction boundaries. However, the resource adapter also supports scenarios where connections are used across transaction boundaries and are obtained prior to the start of a global transaction. For example:

```

Connection con = cf.createConnection("TransactionalCache");

try
{
    OptimisticNamedCache cache = con.getNamedCache("MyTxCache");

    cache.delete("key1", null);

    UserTransaction ut = (UserTransaction) new
        InitialContext().lookup("java:comp/UserTransaction");

    ut.begin();
    cache.insert("key1", "value1");
    ut.commit();
}

```

Packaging the Application

This section provides instructions for packaging JavaEE applications that use the Coherence resource adapter so that they can be deployed to an application server. The following topics are included in this section:

- [Configure the Connection Factory Resource Reference](#)
- [Configure the Resource Adapter Module Reference](#)
- [Include the Required Libraries](#)

Configure the Connection Factory Resource Reference

Application components must provide a resource reference for the resource adapter's connection factory. For EJBs, the resource references are defined in the `ejb-jar.xml` deployment descriptor. For Servlets and JSPs, the resource references are defined in the `web.xml` deployment descriptor. The following sample demonstrates defining a resource reference for the resource adapter's connection factory and is applicable to the code in [Example 25-10](#):

```

<resource-ref>
  <res-ref-name>eis/CoherenceTxCF</res-ref-name>
  <res-type>
    com.tangosol.coherence.transaction.ConnectionFactory
  </res-type>

```

```
<res-auth>Container</res-auth>
</resource-ref>
```

In addition to the standard Java EE application component deployment descriptors, many application servers require a proprietary deployment descriptor as well. For example, WebLogic server resource references are defined in the `weblogic.xml` or `weblogic-ejb-jar.xml` files respectively:

```
<reference-descriptor>
  <resource-description>
    <res-ref-name>eis/CoherenceTxCF</res-ref-name>
    <jndi-name>tangosol.coherenceTx</jndi-name>
  </resource-description>
</reference-descriptor>
```

Consult your application server vendor's documentation for detailed information on using their proprietary application component deployment descriptors as well as information on alternate methods for defining resource reference through the use of 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. For different Coherence deployment options, see [Chapter 31, "Deploying Coherence."](#)

Using the Coherence Cache Adapter for Transactions

The `CoherenceCacheAdapter` 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 some of 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 25-11](#) demonstrates performing cache operations within a transaction when using the `CacheAdapter` class with the new `coherence-transaction.rar` resource adapter. For this example a transactional cache named `MyTxCache` must be configured in the cache configuration file. The cache must map to a transactional cache scheme with the service name `TransactionalCache`. See ["Defining Transactional Caches"](#) on page 25-6 for more information on defining a transactional cache scheme.

Example 25-11 Using the CacheAdapter Class When Using coherence-transaction.rar

```
Context initCtx = new InitialContext();

CacheAdapter adapter = new CacheAdapter(initCtx,
    "java:comp/env/eis/CoherenceTxCCICF", 0, 0, 0);

adapter.connect("TransactionalCache", "scott", "tiger");

try
{
    UserTransaction ut = (UserTransaction) new
        InitialContext().lookup("java:comp/UserTransaction");

    ut.begin();
    OptimisticNamedCache cache =
        (OptimisticNamedCache) adapter.getNamedCache("MyTxCache",
            getClass().getClassLoader());
    cache.delete("key", null);
    cache.insert("key", "value");
    ut.commit();
}
finally
{
    adapter.close();
}
```

```
}
```

Example 25–12 demonstrates performing cache operations within a transaction when using the `CacheAdapter` class with the `coherence-tx.rar` resource adapter.

Example 25–12 Using the `CacheAdapter` Class When Using `coherence-tx.rar`

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

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

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

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

}
}

Data Affinity

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

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

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

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

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

Specifying Affinity

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

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

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

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

Specifying Data Affinity with a KeyAssociation

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

Example 26–1 *Creating a Key Association*

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

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

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

    // {...}
}
```

Specifying Data Affinity with a KeyAssociator

Applications may also provide a custom `KeyAssociator`:

Example 26–2 *A Custom KeyAssociator*

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

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

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

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

Example 26–3 Configuring a Key Associator

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

Example of Using Affinity

[Example 26–4](#) illustrates how to use affinity to create a more efficient query (`NamedCache.entrySet(Filter)`) and cache access (`NamedCache.getAll(Collection)`).

Example 26–4 Using Affinity for a More Efficient Query

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

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

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

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

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

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

Priority Tasks

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

Priority Tasks — Timeouts

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

Configuring Execution Timeouts

The `<request-timeout>`, `<task-timeout>`, and the `<task-hung-threshold>` elements are used to configuring execution timeouts for a service's worker threads. These timeout settings are configured in a cache configuration file and can also be set using command line parameters. See [Chapter 46, "Using the Service Guardian,"](#) for information on setting timeouts for service threads.

[Table 27-1](#) describes the execution timeout elements.

Table 27–1 Execution Timeout Elements

Element Name	Description
<code><request-timeout></code>	<p>Specifies the default timeout value for requests that can time-out (for example, implement the <code>PriorityTask</code> interface), but don't explicitly specify the request timeout value. The request time is measured on the client side as the time elapsed from the moment a request is sent for execution to the corresponding server node(s) and includes the following:</p> <ol style="list-style-type: none"> 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.
<code><task-timeout></code>	<p>Specifies the default timeout value for tasks that can be timed-out (for example, implement the <code>PriorityTask</code> interface), but don't explicitly specify the task execution timeout value. The task execution time is measured on the server side and does not include the time spent waiting in a service backlog queue before being started. This attribute is applied only if the thread pool is used (the <code>thread-count</code> value is positive). If zero is specified, the default service-guardian <code><timeout-milliseconds></code> value is used.</p>
<code><task-hung-threshold></code>	<p>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 <code>thread-count</code> value is positive).</p>

The following example sets a distributed cache's thread count to 7 with a task time out of 5000 milliseconds and a task hung threshold of 10000 milliseconds:

Example 27–1 Sample Task Time and Task Hung Configuration

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

Setting the client request timeout to 15 milliseconds

Example 27–2 Sample Client Request Timeout Configuration

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

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

Command Line Options

The command line options can be used to set the service type default (such as distributed cache, invocation, proxy, and so on) for the node. [Table 27–2](#) describes the options.

Table 27–2 *Command Line Options for Setting Service Type*

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

Priority Task Execution — Custom Objects

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

The API defines the following ways to schedule tasks for execution

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

APIs for Creating Priority Task Objects

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

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

After extending each of these classes the developer will need to implement several methods. The return values for `getRequestTimeoutMillis`, `getExecutionTimeoutMillis`, and `getSchedulingPriority` should be stored on a class-by-class basis in your application configuration parameters. These methods are described in [Table 27–3](#).

Table 27–3 *Methods to Support Task Timeout*

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

Errors Thrown by Task Timeouts

When a task timeout occurs the node will get a `RequestTimeoutException`.

[Example 27–3](#) illustrates an exception that may be thrown.

Example 27–3 Exception Thrown by a TaskTimeout

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


Specifying a Custom Eviction Policy

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

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

[Example 28–1](#) shows the implementation of a simple custom eviction policy:

Example 28–1 Implementing a Custom Eviction Policy

```
package com.tangosol.examples.eviction;

import com.tangosol.net.cache.AbstractEvictionPolicy;
import com.tangosol.net.cache.ConfigurableCacheMap;
import com.tangosol.net.cache.LocalCache;
import com.tangosol.net.BackingMapManagerContext;
import com.tangosol.util.ConverterCollections;
import java.util.Iterator;
import java.util.Map;

/**
 * Custom eviction policy that evicts items randomly (or more specifically,
 * based on the natural order provided by the map's iterator.)
 * This example may be used in cases where fast eviction is required
 * with as little processing as possible.
 */
public class SimpleEvictionPolicy
    extends AbstractEvictionPolicy
{
    /**
     * Default constructor; should be 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 (i.e. the key/value) need
            // to be examined, invoke convertEntry(entry) in case
            // the entry needs to 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);

            // In this example, we're simply iterating and removing items
            // from the cache until we are 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 will deserialize 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 28–2](#) illustrates a Coherence cache configuration file (coherence-cache-config.xml) with an eviction policy:

Example 28–2 Custom Eviction Policy in a coherence-cache-config.xml File

```

<?xml version="1.0"?>

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

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>*</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>
    </near-scheme>
  </caching-schemes>
</cache-config>

```

```

        </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 will be required for deserializing entries
                            -->
                            <init-param>
                                <param-type>
                                    com.tangosol.net.BackingMapManagerContext</param-type>
                                <param-value>{manager-context}</param-value>
                            </init-param>
                        </init-params>
                    </class-scheme>
                </eviction-policy>
                <high-units>20m</high-units>
                <unit-calculator>binary</unit-calculator>
            </local-scheme>
        </backing-map-scheme>
        <autostart>true</autostart>
    </distributed-scheme>
</caching-schemes>
</cache-config>

```

Constraints on Re-entrant Calls

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

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

Re-entrancy, Services, and Service Threads

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

Parent-Child Object Relationships

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

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

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

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

Avoiding Deadlock

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

So:

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

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

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

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

Re-entrancy and Listeners

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

Securing Coherence

This chapter describes the following security features:

- [Using the Access Controller](#)
- [Working in Applications with Installed Security Manager](#)
- [Using SSL in Coherence](#)

Using the Access Controller

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

The following topics are included in this section:

- [Overview of the Access Controller](#)
- [Proof of Identity](#)
- [Proof of Trustworthiness](#)
- [Enabling the Default Access Controller Implementation](#)

Overview of the Access Controller

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

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

The Access Controller serves three purposes:

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

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

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

Coherence provides a default Access Controller implementation that is based on the Key Management infrastructure that is shipped as a standard part of Sun's JDK. See "[Enabling the Default Access Controller Implementation](#)" on page 30-4.

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

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

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

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

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

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

Proof of Identity

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

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

...

Subject subject = Security.login(sName, acPassword);
```

```

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

```

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

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

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

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

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

Proof of Trustworthiness

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

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

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

Enabling the Default Access Controller Implementation

Coherence ships with a default Access Controller implementation that uses a standard Java keystore. The implementation class is the `com.tangosol.net.security.DefaultController` class and is configured in the Coherence operational deployment descriptor.

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

The default access controller implementation is not enabled by default. To enable the default implementation, override the `<enabled>` element within the `<security-config>` node in an operational override file and set it to `true`. For example:

```
<coherence>
  <security-config>
    <enabled>true</enabled>
  </security-config>
</coherence>
```

The default access controller implementation can also be enabled by setting the `tangosol.coherence.security` system property to `true`.

Note: When Coherence security is enabled, every call to the `CacheFactory.getCache()` or `ConfigurableCacheFactory.ensureCache()` API causes a security check. This can negatively impact an application's performance if these calls are made very frequently. The best practice is for the application to hold on to the cache reference and re-use it so that the security check is only performed on the initial call. When using this approach, it is the application's responsibility to ensure that those references are only used in an authorized way.

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

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

The `<access-controller>` element defines the Access Controller implementation that takes two parameters to instantiate.

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

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

Two more steps have to be performed to make the default Access Controller implementation usable in your application:

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

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

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

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

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

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

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

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

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

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

Working in Applications with Installed Security Manager

Complete the following steps when working in applications with the installed security manager.

1. Enter the minimum set of privileges in the policy file.

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

The policy file format is fully described in Java SE Security Guide. See

<http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-spec.doc3.html#20128>

For example:

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

2. Sign the binaries using the JDK `jarsigner` tool, for example:

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

and then additionally protected in the policy file:

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

3. Use operating system mechanisms to protect all relevant files such as policy format, coherence binaries, and permissions from malicious modifications.

Using SSL in Coherence

Coherence provides a Secure Socket Layer (SSL) implementation that can be used to secure TCMP communication between cluster nodes as well as the TCP communication between Coherence*Extend clients and proxies. Coherence supports the Transport Layer Security (TLS) 1.0 protocol which is the next version of the SSL 3.0 protocol; however, the term SSL is used in this documentation since it is the more widely recognized term.

The following topics are included in this section:

- [Overview of SSL](#)
- [Configuring SSL for TCMP](#)

Note: This section only contains instructions for using SSL for TCMP between cluster nodes. For detailed instructions on SSL with Coherence*Extend, see "Using SSL to Secure Client Communication" in *Oracle Coherence Client Guide*.

Overview of SSL

This section provides a brief overview of common SSL concepts that are used in this documentation and is not intended to be a complete guide to SSL. Those new to SSL should refer to the formal specification maintained at <http://www.ietf.org> as well as the Java SE Security resources located at <http://java.sun.com/javase/technologies/security/index.jsp>. Those familiar with SSL can skip this section.

SSL is a security protocol that is used to secure communication between entities (typically, clients and servers) over a network. SSL works by authenticating clients and servers using digital certificates and by encrypting/decrypting communication using unique keys that are associated with authenticated clients and servers.

Establishing Identity and Trust

An entity's identity is established using a digital certificate together with a public and private encryption key. The digital certificate contains general information about the entity and also contains the public encryption key embedded within it. A digital certificate is verified by a Certificate Authority (CA) and signed using the CA's digital certificate. The CA's digital certificate establishes trust that the entity is authentic.

Encrypting and Decrypting Data

An entity's digital certificate contains a public encryption key that is paired with the entity's private encryption key. Certificates are passed between entities during an initial connection. Data is then encrypted using the public key. Data that is encrypted using an entity's public key can only be decrypted by the entity's private key. This ensures that only the entity that owns the public encryption key can decrypt the data.

Using One-Way Authentication Versus Two-Way Authentication

SSL communication between clients and servers can be set up using either one-way or two-way authentication. With one-way authentication, a server is required to identify itself to a client by sending its digital certificate for authentication. The client is not required to send the server a digital certificate and remains anonymous to the server. Two-Way authentication requires both the client and the server to send their respective digital certificates to each other for mutual authentication. Two-way authentication provides stronger security by assuring that the identity on both sides of the communication are known.

Generating SSL Artifacts

The Java `keytool` utility that is located in the `JDK_HOME/bin` directory can be used to generate and manage SSL artifacts. This includes: creating a key store; generating a unique public/private key pair; creating a self-signed digital certificate that includes the public key, associating the certificate with the private key; and storing these artifacts in the key store.

The following example creates a key store named `server.jks` that is located in the `/test` directory. A public and private key pair are generated for the entity identified by the `-dname` value (`"cn=administrator, ou=Coherence, o=Oracle, c=US"`). A self-signed certificate is created that includes the public key and identity information. The certificate is valid for 180 days and is associated with the private key in a key store entry referred to by the alias (`admin`). Passwords must be entered for both the key store and private key.

```
keytool -genkeypair -dname "cn=administrator, ou=Coherence, o=Oracle, c=US"
-alias admin -keypass password -keystore /test/server -storepass password
-validity 180
```

The certificate generated by this command is adequate for development purposes. However, certificates are typically verified by a trusted CA (such as VeriSign). To have the certificate verified, use the `keytool` utility to generate a Certificate Signing Request (CSR) file:

```
keytool -certreq -file admin.csr
```

This CSR file must be sent to the CA, which will return a signed certificate. Use the `keytool` utility to import the returned certificate which will replace the self-signed certificate in the key store:

```
keytool -importcert -trustcacerts -file signed_admin.cer
```

Lastly, the `keytool` utility is used to create a second key store that acts as a trust store. The trust store contains digital certificates of trusted CAs. Certificates that have been verified by a CA are only considered trusted if the CA's certificate is also found in the trust store. For example, in a typical one-way authentication scenario, a client must have a trust store that contains a digital certificate of the CA that signed the server's certificate. For development purposes, a self-signed certificate can be used for both identity and trust; moreover, a single keystore can be used as both the identity store and the trust store.

Configuring SSL for TCMP

A Coherence cluster can be configured to use SSL with TCMP. Coherence supports both one-way and two-way authentication. Two-Way authentication is typically used more often than one-way authentication, which has fewer use cases in a cluster environment. In addition, it is important to realize that TCMP is a peer-to-peer protocol that generally runs in trusted environments where many cluster nodes are expected to remain connected with each other. The implications of SSL on administration and performance should be carefully considered.

Defining a SSL Socket Provider

SSL for TCMP is configured in an operational override file by overriding the `<socket-provider>` element within the `<unicast-listener>` element. The preferred approach is to use the `<socket-provider>` element to reference a SSL socket provider configuration that is defined within a `<socket-providers>` node. The `<socket-provider>` element can also be used to define a full configuration within the `<unicast-listener>` element. Both approaches are demonstrated below. See ["socket-provider"](#) on page A-75 for a detailed reference of the `<socket-provider>` element.

Note: A cluster must be configured to use well known addresses in order to use SSL with TCMP.

[Example 30-1](#) demonstrates a SSL two-way authentication setup and requires both an identity store and trust store to be located on each node in the cluster. The example uses the default values for the `<protocol>` and `<algorithm>` element (TLS and SunX509, respectively). These are shown for completeness but may be left out when using the default values. The example uses the preferred approach where the SSL socket provider is defined within the `<socket-providers>` node and referred to from within the `<unicast-listener>` element.

Example 30-1 Sample SSL Configuration for TCMP Communication

```
<coherence>
  <cluster-config>
    <unicast-listener>
      <well-known-addresses>
        <socket-address id="1">
          <address>198.168.1.5</address>
          <port>8088</port>
        </socket-address>
      </well-known-addresses>
      <socket-provider>mySSLConfig</socket-provider>
    </unicast-listener>

    <socket-providers>
      <socket-provider id="mySSLConfig">
        <ssl>
          <protocol>TLS</protocol>
          <identity-manager>
            <algorithm>SunX509</algorithm>
            <key-store>
              <url>file:server.jks</url>
              <password>password</password>
              <type>JKS</type>
            </key-store>
            <password>password</password>
          </identity-manager>
          <trust-manager>
            <algorithm>SunX509</algorithm>
            <key-store>
              <url>file:trust.jks</url>
              <password>password</password>
              <type>JKS</type>
            </key-store>
          </trust-manager>
        </ssl>
      </socket-provider>
    </socket-providers>
  </cluster-config>
</coherence>
```

As an alternative, the SSL socket provider can also be directly defined in the `<unicast-listener>` element as shown below:

```
<coherence>
  <cluster-config>
    <unicast-listener>
      <well-known-addresses>
        <socket-address id="1">
          <address>198.168.1.5</address>
          <port>8088</port>
        </socket-address>
```

```
</well-known-addresses>
<socket-provider>
  <ssl>
    <protocol>TLS</protocol>
    <identity-manager>
      <algorithm>SunX509</algorithm>
      <key-store>
        <url>file:server.jks</url>
        <password>password</password>
        <type>JKS</type>
      </key-store>
      <password>password</password>
    </identity-manager>
    <trust-manager>
      <algorithm>SunX509</algorithm>
      <key-store>
        <url>file:trust.jks</url>
        <password>password</password>
        <type>JKS</type>
      </key-store>
    </trust-manager>
  </ssl>
</socket-provider>
</unicast-listener>
</cluster-config>
</coherence>
```

Using the Pre-Defined SSL Socket Provider

Out-of-box, a pre-defined SSL socket provider is included that allows for configuration of two-way SSL connections that is based on peer trust where every trusted peer resides within a single JKS key store. The proprietary peer trust algorithm (PeerX509) works by assuming trust (and only trust) of the certificates that are in the key store. The peer algorithm can increase the performance of SSL by leveraging the fact that TCMP is a peer-to-peer protocol.

The pre-defined SSL socket provider is located within the `<socket-providers>` element in the operational deployment descriptor:

```
...
<socket-providers>
  ...
  <socket-provider id="ssl">
    <ssl>
      <identity-manager>
        <key-store>
          <url system-property="tangosol.coherence.security.keystore">
            file:keystore.jks
          </url>
          <password system-property="tangosol.coherence.security.password"/>
        </key-store>
        <password system-property="tangosol.coherence.security.password"/>
      </identity-manager>
      <trust-manager>
        <algorithm>PeerX509</algorithm>
        <key-store>
          <url system-property="tangosol.coherence.security.keystore">
            file:keystore.jks
          </url>
          <password system-property="tangosol.coherence.security.password"/>
        </key-store>
      </trust-manager>
    </ssl>
  </socket-provider>
</socket-providers>
```

```
        </key-store>
    </trust-manager>
</ssl>
</socket-provider>
</socket-providers>
...
```

As configured, the pre-defined SSL socket provider requires a Java key store named `keystore.jks` that is found on the classpath. This name can be overridden using the `tangosol.coherence.security.keystore` system property to specify a different key store. In addition, the `tangosol.coherence.security.password` system property can be used to specify the required password for the key store and certificate. As an alternative, an operational override file may be used to modify the pre-defined SSL socket provider definition as required.

Note: Ensure that certificates for all nodes in the cluster have been imported into the key store.

To use the pre-defined SSL socket provider, override the `<socket-provider>` element in the `<unicast-listener>` configuration and reference the SSL socket provider using its `id` attribute. The following example configures a unicast listener to use the pre-defined SSL socket provider.

```
<coherence>
  <cluster-config>
    <unicast-listener>
      <well-known-addresses>
        <socket-address id="1">
          <address>198.168.1.5</address>
          <port>8088</port>
        </socket-address>
      </well-known-addresses>
      <socket-provider>ssl</socket-provider>
    </unicast-listener>
  </cluster-config>
</coherence>
```


Part V

Deploying Coherence Applications

Part V contains the following chapters:

- [Chapter 31, "Deploying Coherence"](#)
- [Chapter 32, "Platform-Specific Deployment Considerations"](#)
- [Chapter 33, "Production Checklist"](#)

Deploying Coherence

This chapter provides instructions for deploying Coherence into a runtime environment.

The following sections are included in this chapter:

- [Deploying Coherence with a Standalone Application](#)
- [Deploying Coherence to an Application Server](#)

Deploying Coherence with a Standalone Application

Standalone applications that leverage Coherence must include the `COHERENCE_HOME/lib/coherence.jar` library on the application's classpath. This scenario assumes the application is using the default Coherence configuration included within the archive. If a configuration override file is used, then the directory that includes the override file must also be included in the classpath. For more information on modifying the default configuration, see [Chapter 3, "Understanding Configuration."](#)

The following example starts an application called `MyApp`. The classpath includes the `coherence.jar` library as well as the location (`COHERENCE_HOME`) that contains a `tangosol-coherence-override.xml` configuration file.

```
java -jar -cp COHERENCE_HOME;COHERENCE_HOME\lib\coherence.jar com.MyApp
```

Deploying Coherence to an Application Server

Java EE applications that leverage Coherence have two options for deploying Coherence: as an application server library or as part of a Java EE module. Each option results in a different deployment scenario because Coherence cluster nodes are class loader scoped. This means each module can have access to its own Coherence node or all modules can share a single Coherence node. Each option has its own benefits and assumptions and generally balances resource utilization with how isolated the cluster node is from other modules.

Note: This section does not include instructions for deploying Coherence*Web. See the *Oracle Coherence*Web User's Guide* for instructions on deploying Coherence*Web and clustering HTTP session data.

Deploying Coherence as an Application Server Library

Coherence can be deployed as an application server library. In this deployment scenario, an application server's startup classpath is modified to include the

`COHERENCE_HOME/lib/coherence.jar` library. In addition, any objects that are being placed into the cache must also be available in the server's classpath. Consult your application server vendor's documentation for instructions on adding libraries to the server's classpath.

This scenario results in a single Coherence node that is shared by all applications that are deployed in the server's containers. This scenario minimizes resource utilization because only one copy of the Coherence classes is loaded into the JVM. This scenario is ideal if all applications being deployed in a server are known and development teams carefully coordinate and enforce cache conventions and naming standards.

Note: This scenario should be considered very carefully and never used in application servers where application interaction is unknown or unpredictable. In this deployment scenario, all applications are part of the same cluster and the likelihood of collisions between namespaces for caches, services and other configuration settings is high and may lead to unexpected results. All applications may be affected by any one application's use of the Coherence node.

Deploying Coherence in a Java EE Module

Coherence can be deployed within an EAR file or a WAR file. This style of deployment is generally preferred because modification to the application server runtime is not required and because cluster nodes are isolated to either the EAR or WAR.

Deploying Coherence Within an EAR

Coherence can be deployed as part of an EAR. This deployment scenario results in a single Coherence cluster node that is shared by all Web applications in the EAR. Resource utilization is moderate because only one copy of the Coherence classes will be loaded per EAR. However, all Web applications may be affected by any one module's use of the Coherence node. This option is ideal when deploying a single EAR to an application server.

To deploy Coherence within an enterprise application:

1. Copy the `coherence.jar` library to a location within the enterprise application directory structure.
2. Using a text editor, open the `META-INF/application.xml` deployment descriptor.
3. Add a `<java>` element that contains the path (relative to the top level of the application directory) and name of the coherence library. For example:

```
<application>
  <display-name>MyApp</display-name>
  <module>
    <java>coherence.jar</java>
  </module>
  ...
</application>
```

4. Make sure any objects that are to be placed in the cache are added to the application in the same manner as described above.
5. Save and close the descriptor.
6. package and deploy the application.

Deploying Coherence Within a WAR

Coherence can be deployed as part of a Web application. This deployment scenario results in each Web application having its own Coherence node in the cluster, which is isolated from all other Web applications. This scenario utilizes the most amount of resources because there will be as many copies of the Coherence classes loaded as there are deployed Web applications that include Coherence. This scenario is ideal when deploying only a few Web applications to an application server.

To deploy Coherence within a Web application:

1. Copy the `coherence.jar` library to the Web Application's `WEB-INF/lib` directory.
2. Make sure any objects that are to be placed in the cache are located in either the `WEB-INF/lib` or `WEB-INF/classes` directory.
3. Package and deploy the application.

Platform-Specific Deployment Considerations

The following sections in this appendix provide information on deploying Coherence to various platforms.

- [Deploying to AIX](#)
- [Deploying to Oracle JRockit JVMs](#)
- [Deploying to Cisco Switches](#)
- [Deploying to Foundry Switches](#)
- [Deploying to IBM BladeCenters](#)
- [Deploying to IBM JVMs](#)
- [Deploying to Linux](#)
- [Deploying to OS X](#)
- [Deploying to Solaris](#)
- [Deploying to Sun JVMs](#)
- [Deploying to Virtual Machines](#)
- [Deploying to Windows](#)
- [Deploying to z OS](#)

Deploying to AIX

When deploying Coherence on AIX please be aware of the following:

Socket Buffers sizes and JVMs

There is an issue with IBM's 1.5 JVMs for AIX which may prevent them from allocating socket buffers larger than 64K (Oracle 2MB). This issue has been addressed in IBM's 1.5 SR3 SDK. See ["Operating System Tuning"](#) on page 45-1.

Multicast and IPv6

AIX 5.2 and above default to running multicast over IPv6 rather than IPv4. If you run in a mixed IPv6/IPv4 environment you will need to configure your JVMs to explicitly use IPv4. This can be done by setting the `java.net.preferIPv4Stack` system property to true on the Java command line. See the IBM 32-bit SDK for AIX User Guide for details.

Unique Multicast Addresses and Ports

On AIX it is suggested that each Coherence cluster use a unique multicast address and port, as some versions of AIX will not take both into account when delivering packets. See "[multicast-listener](#)" on page A-42 for details on configuring the address.

Deploying to Oracle JRockit JVMs

When deploying Coherence on JRockit JVMs please be aware of the following:

JRockit and the Native Posix Thread Library (NPTL)

When running JRockit on Linux, Oracle recommends using 2.6 kernels, and ensuring that the NPTL is enabled. Please see Oracle's documentation regarding this issue.

OutOfMemoryError

JVMs that experience an `OutOfMemoryError` can be left in an indeterministic state which can have adverse effects on a cluster. We recommend configuring JVMs to exit upon encountering an `OutOfMemoryError` instead of allowing the JVM to attempt recovery. Here is the parameter to configure this setting on JRockit JVMs:

```
-XXexitOnOutOfMemory
```

Additionally, it is recommended to configure the JVM to generate a heap dump if an `OutOfMemoryError` is thrown in order to assist the investigation into the root cause for the error. Use the following flags to enable this feature on JRockit:

```
-Djrockit.oomdiagnostics=true -Djrockit.oomdiagnostics.filename=<path to file>
```

Deploying to Cisco Switches

When deploying Coherence with Cisco switches please be aware of the following:

Buffer Space and Packet Pauses

Under heavy UDP packet load some Cisco switches may run out of buffer space and exhibit frequent multi-second communication pauses. These communication pauses can be identified by a series of Coherence log messages referencing communication delays with multiple nodes which cannot be attributed to local or remote GCs.

```
Experienced a 4172 ms communication delay (probable remote GC) with Member(Id=7,
Timestamp=2008-09-15 12:15:47.511, Address=xxx.xxx.x.xx:8089, MachineId=13838);
320 packets rescheduled, PauseRate=0.31, Threshold=512
```

The Cisco 6500 series support configuration the amount of buffer space available to each Ethernet port or ASIC. In high load applications it may be necessary to increase the default buffer space. This can be accomplished by executing:

```
fabric buffer-reserve high
```

See Cisco's documentation for additional details on this setting.

Multicast Connectivity on Large Networks

Cisco's default switch configuration does not support proper routing of multicast packets between switches due to the use of IGMP snooping. See the Cisco's documentation regarding the issue and solutions.

Multicast Outages

Some Cisco switches have shown difficulty in maintaining multicast group membership resulting in existing multicast group members being silently removed from the multicast group. This will cause a partial communication disconnect for the associated Coherence node(s) and they will be forced to leave and rejoin the cluster. This type of outage can most often be identified by the following Coherence log messages indicating that a partial communication problem has been detected.

A potential network configuration problem has been detected. A packet has failed to be delivered (or acknowledged) after 60 seconds, although other packets were acknowledged by the same cluster member (Member(Id=3, Timestamp=Sat Sept 13 12:02:54 EST 2008, Address=xxx.xxx.x.xxx, Port=8088, MachineId=48991)) to this member (Member(Id=1, Timestamp=Sat Sept 13 11:51:11 EST 2008, Address=xxx.xxx.x.xxx, Port=8088, MachineId=49002)) as recently as 5 seconds ago.

To confirm the issue you may run the using the same multicast address and port as the running cluster. If the issue affects a multicast test node its logs will show that at some point it will suddenly stop receiving multicast test messages. See [Chapter 43, "Performing a Multicast Connectivity Test"](#).

The following test logs show the issue:

Example 32-1 Log for a Multicast Outage

Test Node 192.168.1.100:

```
Sun Sept 14 16:44:22 GMT 2008: Received 83 bytes from a Coherence cluster node at
182.168.1.100: ???
Sun Sept 14 16:44:23 GMT 2008: Received test packet 76 from ip=/192.168.1.101,
group=/224.3.2.0:32367, ttl=4.
Sun Sept 14 16:44:23 GMT 2008: Received 83 bytes from a Coherence cluster node at
182.168.1.100: ???
Sun Sept 14 16:44:23 GMT 2008: Sent packet 85. Sun Sept 14 16:44:23 GMT 2008:
Received test packet 85 from self.
Sun Sept 14 16:44:24 GMT 2008: Received 83 bytes from a Coherence cluster node at
182.168.1.100: ???
Sun Sept 14 16:44:25 GMT 2008: Received test packet 77 from ip=/192.168.1.101,
group=/224.3.2.0:32367, ttl=4.
Sun Sept 14 16:44:25 GMT 2008: Received 83 bytes from a Coherence cluster node at
182.168.1.100: ???
Sun Sept 14 16:44:25 GMT 2008: Sent packet 86.
Sun Sept 14 16:44:25 GMT 2008: Received test packet 86 from self. Sun Sept 14
16:44:26 GMT 2008: Received 83 bytes from a Coherence cluster node at
182.168.1.100: ???
Sun Sept 14 16:44:27 GMT 2008: Received test packet 78 from ip=/192.168.1.101,
group=/224.3.2.0:32367, ttl=4.
Sun Sept 14 16:44:27 GMT 2008: Received 83 bytes from a Coherence cluster node at
182.168.1.100: ???
Sun Sept 14 16:44:27 GMT 2008: Sent packet 87.
Sun Sept 14 16:44:27 GMT 2008: Received test packet 87 from self.
Sun Sept 14 16:44:28 GMT 2008: Received 83 bytes from a Coherence cluster node at
182.168.1.100: ???
Sun Sept 14 16:44:29 GMT 2008: Received 83 bytes from a Coherence cluster node at
182.168.1.100: ???
Sun Sept 14 16:44:29 GMT 2008: Sent packet 88.
Sun Sept 14 16:44:29 GMT 2008: Received test packet 88 from self.
Sun Sept 14 16:44:30 GMT 2008: Received 83 bytes from a Coherence cluster node at
182.168.1.100: ???
Sun Sept 14 16:44:31 GMT 2008: Received 83 bytes from a Coherence cluster node at
182.168.1.100: ???
```

```
Sun Sept 14 16:44:31 GMT 2008: Sent packet 89.
Sun Sept 14 16:44:31 GMT 2008: Received test packet 89 from self.
Sun Sept 14 16:44:32 GMT 2008: Received 83 bytes from a Coherence cluster node at
192.168.1.100: ???
Sun Sept 14 16:44:33 GMT 2008: Received 83 bytes from a Coherence cluster node at
192.168.1.100: ???
```

Test Node 192.168.1.101:

```
Sun Sept 14 16:44:22 GMT 2008: Sent packet 76.
Sun Sept 14 16:44:22 GMT 2008: Received test packet 76 from self.
Sun Sept 14 16:44:22 GMT 2008: Received 83 bytes from a Coherence cluster node at
192.168.1.100: ???
Sun Sept 14 16:44:22 GMT 2008: Received test packet 85 from ip=/192.168.1.100,
group=/224.3.2.0:32367, ttl=4.
Sun Sept 14 16:44:23 GMT 2008: Received 83 bytes from a Coherence cluster node at
192.168.1.100: ??? Sun Sept 14 16:44:24 GMT 2008: Sent packet 77.
Sun Sept 14 16:44:24 GMT 2008: Received test packet 77 from self.
Sun Sept 14 16:44:24 GMT 2008: Received 83 bytes from a Coherence cluster node at
192.168.1.100: ???
Sun Sept 14 16:44:24 GMT 2008: Received test packet 86 from ip=/192.168.1.100,
group=/224.3.2.0:32367, ttl=4.
Sun Sept 14 16:44:25 GMT 2008: Received 83 bytes from a Coherence cluster node at
192.168.1.100: ???
Sun Sept 14 16:44:26 GMT 2008: Sent packet 78. Sun Sept 14 16:44:26 GMT 2008:
Received test packet 78 from self.
Sun Sept 14 16:44:26 GMT 2008: Received 83 bytes from a Coherence cluster node at
192.168.1.100: ???
Sun Sept 14 16:44:26 GMT 2008: Received test packet 87 from ip=/192.168.1.100,
group=/224.3.2.0:32367, ttl=4.
Sun Sept 14 16:44:27 GMT 2008: Received 83 bytes from a Coherence cluster node at
192.168.1.100: ???
Sun Sept 14 16:44:28 GMT 2008: Sent packet 79.
Sun Sept 14 16:44:28 GMT 2008: Received test packet 79 from self.
Sun Sept 14 16:44:28 GMT 2008: Received 83 bytes from a Coherence cluster node at
192.168.1.100: ???
Sun Sept 14 16:44:28 GMT 2008: Received test packet 88 from ip=/192.168.1.100,
group=/224.3.2.0:32367, ttl=4.
Sun Sept 14 16:44:29 GMT 2008: Received 83 bytes from a Coherence cluster node at
192.168.1.100: ???
Sun Sept 14 16:44:30 GMT 2008: Sent packet 80.
Sun Sept 14 16:44:30 GMT 2008: Received test packet 80 from self.
Sun Sept 14 16:44:30 GMT 2008: Received 83 bytes from a Coherence cluster node at
192.168.1.100: ???
Sun Sept 14 16:44:30 GMT 2008: Received test packet 89 from ip=/192.168.1.100,
group=/224.3.2.0:32367, ttl=4.
Sun Sept 14 16:44:31 GMT 2008: Received 83 bytes from a Coherence cluster node at
192.168.1.100: ???
Sun Sept 14 16:44:32 GMT 2008: Sent packet 81. Sun Sept 14 16:44:32 GMT 2008:
Received test packet 81 from self.
Sun Sept 14 16:44:32 GMT 2008: Received 83 bytes from a Coherence cluster node at
192.168.1.100: ???
Sun Sept 14 16:44:32 GMT 2008: Received test packet 90 from ip=/192.168.1.100,
group=/224.3.2.0:32367, ttl=4.
Sun Sept 14 16:44:33 GMT 2008: Received 83 bytes from a Coherence cluster node at
192.168.1.100: ???
Sun Sept 14 16:44:34 GMT 2008: Sent packet 82.
```

Note that at 16:44:27 the first test node stops receiving multicast packets from other machines. The operating system continues to properly forward multicast traffic from

other processes on the same machine, but the test packets (79 and higher) from the second test node are not received. Also note that both the test packets and the cluster's multicast traffic generated by the first node do continue to be delivered to the second node. This indicates that the first node was silently removed from the multicast group.

If you encounter this multicast issue it is suggested that you contact Cisco technical support, or you may consider changing your configuration to unicast-only by using the Coherence well-known-addresses feature. See ["well-known-addresses"](#) on page A-86.

Deploying to Foundry Switches

When deploying Coherence with Foundry switches please be aware of the following:

Multicast Connectivity

Foundry switches have shown to exhibit difficulty in handing multicast traffic. When deploying on with Foundry switches it is recommend that you use the to ensure that all machines which will be part of the Coherence cluster can communicate over multicast. See [Chapter 43, "Performing a Multicast Connectivity Test"](#).

If you encounter issues with multicast you may consider changing your configuration to unicast-only by using the well-known-addresses feature. See ["well-known-addresses"](#) on page A-86.

Deploying to IBM BladeCenters

When deploying Coherence on IBM BladeCenters please be aware of the following:

MAC Address Uniformity and Load Balancing

A typical deployment on a BladeCenter may include blades with two NICs where one is used for administration purposes and the other for cluster traffic. By default the MAC addresses assigned to the blades of a BladeCenter are uniform enough that the first NIC will generally have an even MAC address and the second will have an odd MAC address. If the BladeCenter's uplink to a central switch also has an even number of channels then layer 2 (MAC based) load balancing may prevent one set of NICs from making full use of the available uplink bandwidth as they will all be bound to either even or odd channels. This issue arises due to the assumption in the switch that MAC addresses are essentially random, which in BladeCenter's is untrue. Remedies to this situation include:

- Use layer 3 (IP based) load balancing, assuming that the IP addresses do not follow the same even/odd pattern.
 - This setting would need to be applied across all switches carrying cluster traffic.
- Randomize the MAC address assignments by swapping them between the first and second NIC on alternating machines.
 - Linux enables you to change a NIC's MAC address using the `ifconfig` command.
 - For other operating systems custom tools may be available to perform the same task.

Deploying to IBM JVMs

When deploying Coherence on IBM JVMs please be aware of the following:

UDP Socket Buffer Sizes

There is an issue with IBM's 1.5 JVMs which may prevent them from allocating socket buffers larger than 64K (Note that buffers of 2MB are recommended for Coherence). This issue has been addressed in IBM's 1.5 SR3 SDK. For performance reasons it is suggested that the patch be applied.

OutOfMemoryError

JVMs that experience an `OutOfMemoryError` can be left in an indeterministic state which can have adverse effects on a cluster. We recommend configuring JVMs to exit upon encountering an `OutOfMemoryError` instead of allowing the JVM to attempt recovery. Here is the parameter to configure this setting on IBM JVMs (1.5 and above):

Unix:

```
-Xdump:tool:events=throw,filter=java/lang/OutOfMemoryError,exec="kill -9 %pid"
```

Windows:

```
-Xdump:tool:events=throw,filter=java/lang/OutOfMemoryError,exec="taskkill /F /PID %pid"
```

Heap Sizing

IBM does not recommend fixed size heaps for JVMs. In many cases, it is recommended to use the default for `-Xms` (in other words, omit this setting and only set `-Xmx`). See this link for more details:

<http://www.ibm.com/developerworks/java/jdk/diagnosis/>

It is recommended to configure the JVM to generate a heap dump if an `OutOfMemoryError` is thrown in order to assist the investigation into the root cause for the error. IBM JVMs will generate a heap dump on `OutOfMemoryError` by default; no further configuration is required.

Deploying to Linux

When deploying Coherence on Linux please be aware of the following:

Native POSIX Thread Library (NPTL)

Early versions of the NPTL are prone to deadlock, especially when combined with 2.4 Linux Kernels. The kernel version and NPTL version can be obtained by executing the following commands:

```
uname -a
getconf GNU_LIBPTHREAD_VERSION
```

If running on a 2.4 kernel, it is recommended that you avoid using any version of the NPTL, and revert to using LinuxThreads library. This can be done by setting the LD_ASSUME_KERNEL environment variable before launching Java.

```
export LD_ASSUME_KERNEL=2.4.19
getconf GNU_LIBPTHREAD_VERSION
```

If running on a 2.6 kernel, it is recommended that you use a 1.0 or higher version of NPTL. If upgrading the NPTL version is not possible then it is then recommended that you switch to LinuxThreads library.

NPTL related issues are known to occur with Red Hat Linux 9 and Red Hat Enterprise Linux 3, and are also likely to effect any 2.4 based Linux distribution with a backported version of the NPTL. See

<http://java.sun.com/developer/technicalArticles/JavaTechandLinux/RedHat> for more details on this issue.

TSC High Resolution Timesource

Linux has several high resolution timesources to choose from, the fastest TSC (Time Stamp Counter) unfortunately is not always reliable. Linux chooses TSC by default, and during boot checks for inconsistencies, if found it switches to a slower safe timesource. The slower time sources can be 10 to 30 times more expensive to query than the TSC timesource, and may have a measurable impact on Coherence performance. Note that Coherence and the underlying JVM are not aware of the timesource which the operating system is using. It is suggested that you check your system logs (/var/log/dmesg) to verify that the following is not present.

```
kernel: Losing too many ticks!
kernel: TSC cannot be used as a timesource.
kernel: Possible reasons for this are:
kernel:   You're running with Speedstep,
kernel:   You don't have DMA enabled for your hard disk (see hdparm),
kernel:   Incorrect TSC synchronization on an SMP system (see dmesg).
kernel: Falling back to a sane timesource now.
```

As the log messages suggest, this can be caused by a variable rate CPU (SpeedStep), having DMA disabled, or incorrect TSC synchronization on multi CPU machines. If present it is suggested that you work with your system administrator to identify the cause and allow the TSC timesource to be used.

Deploying to OS X

When deploying Coherence on OS X please be aware of the following:

Multicast and IPv6

OS X defaults to running multicast over IPv6 rather than IPv4. If you run in a mixed IPv6/IPv4 environment you will need to configure your JVMs to explicitly use IPv4. This can be done by setting the `java.net.preferIPv4Stack` system property to true on the Java command line.

Unique Multicast Addresses and Ports

On OS X it is suggested that each Coherence cluster use a unique multicast address and port, as some versions of OS X will not take both into account when delivering packets. See the `multicast-listener` for details on configuring the address.

Socket Buffer Sizing

Generally Coherence prefers 2MB or higher buffers, but in the case of OS X this may result in unexpectedly high kernel CPU time, which in turn reduces throughput. For OS X the suggested buffers size is 768KB, though your own tuning may find a better sweet spot. See ["packet-buffer"](#) on page A-48 for details on specifying the amount of buffer space Coherence will request.

Deploying to Solaris

When deploying Coherence on Solaris please be aware of the following:

Solaris 10 (x86 and SPARC)

Note: If running on Solaris 10, please review Sun issues 102712 and 102741 which relate to packet corruption and multicast disconnections. These will most often manifest as either `EOFExceptions`, "Large gap" warnings while reading packet data, or frequent packet timeouts. It is highly recommend that the patches for both issues be applied when using Coherence on Solaris 10 systems.

Sun Alert 102712:

Possible Data Integrity Issues on Solaris 10 Systems Using the e1000g Driver for the Intel Gigabit Network Interface Card (NIC)

Sun Alert 102741:

IGMP(1) Packets do not Contain IP Router Alert Option When Sent From Solaris 10 Systems With Patch 118822-21 (SPARC) or 118844-21 (x86/x64) or Later Installed

Solaris 10 Networking

If running on Solaris 10, please review Sun issues 102712 and 102741 which relate to packet corruption and multicast disconnections. These will most often manifest as either `EOFExceptions`, "Large gap" warnings while reading packet data, or frequent packet timeouts. It is highly recommend that the patches for both issues be applied when using Coherence on Solaris 10 systems.

Deploying to Sun JVMs

When deploying Coherence on Sun JVMs please be aware of the following:

Heap Sizes

Coherence recommends keeping heap sizes at 1GB in size per JVM. Multiple cache servers can be used allow a single machine to achieve higher capacities. With Sun's 1.5 JVMs, heap sizes beyond 1GB are reasonable, though GC tuning is still advisable to minimize long GC pauses. See Sun's GC Tuning Guide for tuning details. It is also advisable to run with fixed sized heaps as this generally lowers GC times. See ["JVM Tuning"](#) on page 45-7 for additional information.

AtomicLong

When available Coherence will make use of the highly concurrent AtomicLong class, which allows concurrent atomic updates to long values without requiring synchronization.

It is suggested to run in server mode to ensure that the stable and highly concurrent version can be used. To run the JVM in server mode include the -server option on the Java command line.

OutOfMemoryError

JVMs that experience an OutOfMemoryError can be left in an indeterministic state which can have adverse effects on a cluster. We recommend configuring JVMs to exit upon encountering an OutOfMemoryError instead of allowing the JVM to attempt recovery. Here is the parameter to configure this setting on Sun JVMs:

Unix:

```
-XX:OnOutOfMemoryError="kill -9 %p"
```

Windows:

```
-XX:OnOutOfMemoryError="taskkill /F /PID %p"
```

Note: As of December 2008, this flag is available on version 1.6, but not on 1.5.

Additionally, it is recommended to configure the JVM to generate a heap dump if an OutOfMemoryError is thrown in order to assist the investigation into the root cause for the error. Use the following flag to enable this feature on the Sun JVM:

```
-XX:+HeapDumpOnOutOfMemoryError
```

Deploying to Virtual Machines

When deploying Coherence to virtual machines, please be aware of the following:

Supported Deployment

Coherence is supported within virtual machine environments and there should be no functional differences between running it there or in a non-virtualized operating system.

There is currently an issue with the VMWare Query Performance Counter feature that causes Coherence to hang when starting with JRockit on a Windows 2003 or Windows XP VM image. To work around this issue, remove the `/usepmtimer` switch from the `boot.ini` file and restart the virtual machine. The issue is detailed in the following VMWare knowledge base article:

http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1011714

Multicast Connectivity

Using virtualization adds another layer to your network topology, and like all other layers it must be properly configured to support multicast networking. See "[multicast-listener](#)" on page A-42.

Performance

It is less likely that a process running in a virtualized OS will be able to fully use gigabit Ethernet. This is not specific to Coherence, and will be visible on most network intensive virtualized applications.

See the following VMWare article covering their network performance as compared to non-virtualized operating systems.

http://www.vmware.com/pdf/esx_network_planning.pdf

Fault Tolerance

From a Coherence fault tolerance perspective there is more configuration which needs to occur to ensure that cache entry backups reside on physically separate hardware. Manual machine identity must be configured so that Coherence can ensure that backups are not inadvertently stored on the same physical machine as the primary. This can be configured by using the `machine-id` element within the operational configuration file. See the configuration for "[unicast-listener](#)" on page A-83 for details.

Deploying to Windows

When deploying Coherence on Windows please be aware of the following:

Performance Tuning

Out of the box Windows is not optimized for background processes and heavy network loads. This may be addressed by running the `optimize.reg` script included in the Coherence installation's `bin` directory. See "[Operating System Tuning](#)" on page 45-1 for details on the optimizations which will be performed.

Personal Firewalls

If running a firewall on a machine you may have difficulties in forming a cluster consisting of multiple computers. This can be resolved by either:

- Disabling the firewall, though this is generally not recommended.
- Granting full network access to the Java executable which will run Coherence.
- Opening up individual address and ports for Coherence.

By default Coherence uses TCP and UDP ports starting at 8088, subsequent nodes on the same machine will use increasing port numbers. Coherence may also communicate over multicast; the default address and port differs between releases and is based on the version number of the release. See "[unicast-listener](#)" on page A-83 and "[multicast-listener](#)" on page A-42 for details on address and port configuration.

Disconnected Network Interface

On Microsoft Windows, if the Network Interface Card (NIC) is unplugged from the network the OS will invalidate the associated IP address. The effect of this is that any socket which is bound to that IP address will enter an error state. This will result the Coherence nodes running exiting the cluster and residing in an error state until the NIC is reattached to the network. In cases where it is desirable to allow multiple collocated JVMs to remain clustered during a physical outage Windows must be configured to not invalidate the IP address.

To adjust this parameter:

1. Run Registry Editor (regedit)
2. Locate the following registry key

HKLM\System\CurrentControlSet\Services\Tcpip\Parameters

3. Add or reset the following new DWORD value

Name: DisableDHCPMediaSense
Value: 1 (boolean)

4. Reboot

While the name of the keyword includes DHCP, the setting effects both static and dynamic IP addresses. See Microsoft Windows TCP/IP Implementation Details for additional information:

<http://technet.microsoft.com/en-us/library/bb726981.aspx#EDAA>

Deploying to z OS

When deploying Coherence on z/OS please be aware of the following:

EBCDIC

When deploying Coherence into environments where the default character set is EBCDIC rather than ASCII, please make sure that Coherence the configuration files which are loaded from JAR files or off of the classpath are in ASCII format. Configuration files loaded directly from the file system should be stored in the systems native format of EBCDIC.

Multicast

Under some circumstances, Coherence cluster nodes that run within the same logical partition (LPAR) on z/OS on IBM zSeries cannot communicate with each other. (This problem does not occur on the zSeries when running on Linux.)

The root cause is that z/OS may bind the MulticastSocket that Coherence uses to an automatically-assigned port, but Coherence requires the use of a specific port in order for cluster discovery to operate correctly. (Coherence does explicitly initialize the `java.net.MulticastSocket` to use the necessary port, but that information appears to be ignored on z/OS when there already is an instance of Coherence running within that same LPAR.)

The solution is to run only one instance of Coherence within a z/OS LPAR; if multiple instances are required, each instance of Coherence should be run in a separate z/OS LPAR. Alternatively well known addresses may be used. See ["well-known-addresses"](#) on page A-86 for more information.

Production Checklist

Note: Deploying Coherence in a production environment is very different from using Coherence in a development environment.

Development environments do not reflect the challenges of a production environment.

Coherence tends to be so simple to use in development that developers do not take the necessary planning steps and precautions when moving an application using Coherence into production. This article is intended to accomplish the following:

- Create a healthy appreciation for the complexities of deploying production software, particularly large-scale infrastructure software and enterprise applications;
- Enumerate areas that require planning when deploying Coherence;
- Define why production awareness should exist for each of those areas;
- Suggest or require specific approaches and solutions for each of those areas; and
- Provide a check-list to minimize risk when deploying to production.

Deployment recommendations are available for:

- [Network](#)
- [Hardware](#)
- [Operating System](#)
- [JVM](#)
- [Java Security Manager](#)
- [Application Instrumentation](#)
- [Coherence Editions and Modes](#)
- [Coherence Operational Configuration](#)
- [Coherence Cache Configuration](#)
- [Large Cluster Configuration](#)
- [Death Detection](#)

Network

During development, a Coherence-enabled application on a developer's local machine can accidentally form a cluster with the application running on other developers' machines.

Developers often use and test Coherence locally on their workstations. There are several ways in which they may accomplish this, including:

- Setting the multicast TTL to zero,
- Using a "loopback", or
- By each developer using a different multi-cast address and port from all other developers.

If one of these approaches is not used, then multiple developers on the same network will find that Coherence has clustered across different developers' locally running instances of the application; in fact, this happens relatively often and causes confusion when it is not understood by the developers.

Setting the TTL to zero on the command line is very simple: Add the following to the JVM startup parameters:

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

Starting with Coherence version 3.2, setting the TTL to zero for all developers is also very simple. Edit the `tangosol-coherence-override-dev.xml` in the `coherence.jar` file, changing the TTL setting as follows:

```
<time-to-live system-property="tangosol.coherence.ttl">0</time-to-live>
```

On some UNIX operating systems, including some versions of Linux and Mac OS X, setting the TTL to zero may not be enough to isolate a cluster to a single machine. To be safe, assign a different cluster name for each developer, for example using the developer's email address as the cluster name. If the cluster communication does go across the network to other developer machines, then the different cluster name will cause an error on the node that is attempting to start up.

To ensure that the clusters are completely isolated, select a different multicast IP address and port for each developer. In some organizations, a simple approach is to use the developer's phone extension number as part of the multicast address and as the port number (or some part of it). For information on configuring the multicast IP address and port, see ["multicast-listener"](#) on page A-42.

During development, clustered functionality is often not being tested.

After the POC or prototype stage is complete, and until load testing begins, it is not out of the ordinary for the application to be developed and tested by engineers in a non-clustered form. This is dangerous, as testing primarily in the non-clustered configuration can hide problems with the application architecture and implementation that will show up later in staging, or even production.

Make sure that the application is being tested in a clustered configuration as development proceeds. There are several ways for clustered testing to be a natural part of the development process; for example:

- Developers can test with a locally clustered configuration (at least two instances running on their own machine). This works well with the `TTL=0` setting, since clustering on a single machine works with the `TTL=0` setting.

- Unit and regression tests can be introduced that run in a test environment that is clustered. This may help automate certain types of clustered testing that an individual developer would not always remember (or have the time) to do.

What is the type and speed of the production network?

Most production networks are based on gigabit Ethernet, with a few still built on slower 100Mb Ethernet or faster ten-gigabit Ethernet. It is important to understand the topology of the production network, and what the full set of devices that will connect all of the servers that will be running Coherence. For example, if there are ten different switches being used to connect the servers, are they all the same type (make and model) of switch? Are they all the same speed? Do the servers support the network speeds that are available?

In general, all servers should share a reliable, fully switched network. This generally implies sharing a single switch (ideally, two parallel switches and two network cards per server for availability). There are two primary reasons for this. The first is that using more than one switch almost always results in a reduction in effective network capacity. The second is that multi-switch environments are more likely to have network "partitioning" events where a partial network failure will result in two or more disconnected sets of servers. While partitioning events are rare, Coherence cache servers ideally should share a common switch.

To demonstrate the impact of multiple switches on bandwidth, consider several servers plugged into a single switch. As additional servers are added, each server receives dedicated bandwidth from the switch backplane. For example, on a fully switched gigabit backplane, each server receives a gigabit of inbound bandwidth and a gigabit of outbound bandwidth for a total of 2Gbps "full duplex" bandwidth. Four servers would have an aggregate of 8Gbps bandwidth. Eight servers would have an aggregate of 16Gbps. And so on up to the limit of the switch (in practice, usually in the range of 160-192Gbps for a gigabit switch). However, consider the case of two switches connected by a 4Gbps (8Gbps full duplex) link. In this case, as servers are added to each switch, they will have full "mesh" bandwidth up to a limit of four servers on each switch (that is, all four servers on one switch can communicate at full speed with the four servers on the other switch). However, adding additional servers will potentially create a bottleneck on the inter-switch link. For example, if five servers on one switch send data to five servers on the other switch at 1Gbps per server, then the combined 5Gbps will be restricted by the 4Gbps link. Note that the actual limit may be much higher depending on the traffic-per-server and also the portion of traffic that actually needs to move across the link. Also note that other factors such as network protocol overhead and uneven traffic patterns may make the usable limit much lower from an application perspective.

Avoid mixing and matching network speeds: Make sure that all servers can **and do** connect to the network at the same speed, and that all of the switches and routers between those servers run at that same speed or faster.

Oracle strongly suggests GigE or faster: Gigabit Ethernet is supported by most servers built since 2004, and Gigabit switches are economical, available and widely deployed.

Before deploying an application, you must run the Datagram Test to test the actual network speed and determine its capability for pushing large amounts of data. Furthermore, the Datagram test must be run with an increasing ratio of publishers to consumers, since a network that appears fine with a single publisher and a single consumer may completely fall apart as the number of publishers increases, such as occurs with the default configuration of Cisco 6500 series switches. See ["Deploying to Cisco Switches"](#) on page 32-2 for more information.

Will the production deployment use multicast?

The term "multicast" refers to the ability to send a packet of information from one server and to have that packet delivered in parallel by the network to many servers. Coherence supports both multicast and multicast-free clustering. Oracle *suggests* the use of multicast when possible because it is an efficient option for many servers to communicate. However, there are several common reasons why multicast cannot be used:

- Some organizations disallow the use of multicast.
- Multicast cannot operate over certain types of network equipment; for example, many WAN routers disallow or do not support multicast traffic.
- Multicast is occasionally unavailable for technical reasons; for example, some switches do not support multicast traffic.

First determine if multicast will be used. In other words, determine if the desired deployment configuration is to use multicast.

Before deploying an application that will use multicast, you must run the Multicast Test to verify that multicast is working and to determine the correct (the minimum) TTL value for the production environment. See [Chapter 43, "Performing a Multicast Connectivity Test"](#) for more information.

Applications that cannot use multicast for deployment must use the WKA configuration. See ["well-known-addresses"](#) on page A-86 and "Network Protocols" for more information.

Are your network devices configured optimally?

If the above datagram and/or multicast tests have failed or returned poor results, it is possible that there are configuration problems with the network devices in use. Even if the tests passed without incident and the results were perfect, it is still possible that there are lurking issues with the configuration of the network devices.

Review the suggestions in ["Network Tuning"](#) on page 45-4.

How will the cluster handle a sustained network outage?

The Coherence cluster protocol is capable of detecting and handling a wide variety of connectivity failures. The clustered services are able to identify the connectivity issue, and force the offending cluster node to leave and re-join the cluster. In this way the cluster ensures a consistent shared state among its members.

See ["Death Detection"](#) on page 33-14 for more details. See also:

- ["Deploying to Cisco Switches"](#) on page 32-2
- ["Deploying to Foundry Switches"](#) on page 32-5

Hardware

During development, developers can form unrealistic performance expectations.

Most developers have relatively fast workstations. Combined with test cases that are typically non-clustered and tend to represent single-user access (that is, only the developer), the application may seem extraordinarily responsive.

Include as a requirement that realistic load tests be built that can be run with simulated concurrent user load.

Test routinely in a clustered configuration with simulated concurrent user load.

During development, developer productivity can be adversely affected by inadequate hardware resources, and certain types of quality can also be affected negatively.

Coherence is compatible with all common workstation hardware. Most developers use PC or Apple hardware, including notebooks, desktops and workstations.

Developer systems should have a significant amount of RAM to run a modern IDE, debugger, application server, database and at least two cluster instances. Memory utilization varies widely, but to ensure productivity, the suggested minimum memory configuration for developer systems is 2GB. Desktop systems and workstations can often be configured with 4GB for minimal additional cost.

Developer systems should have two CPU cores or more. Although this will have the likely side-effect of making developers happier, the actual purpose is to increase the quality of code related to multi-threading, since many bugs related to concurrent execution of multiple threads will only show up on multi-CPU systems (systems that contain multiple processor sockets and/or CPU cores).

What are the supported and suggested server hardware platforms for deploying Coherence on?

The short answer is that Oracle works to support the hardware that the customer has standardized on or otherwise selected for production deployment.

- Oracle has customers running on virtually all major server hardware platforms. The majority of customers use "commodity x86" servers, with a significant number deploying Sun Sparc (including Niagara) and IBM Power servers.
- Oracle continually tests Coherence on "commodity x86" servers, both Intel and AMD.
- Intel, Apple and IBM provide hardware, tuning assistance and testing support to Oracle.
- Oracle conducts internal Coherence certification on all IBM server platforms at least once a year.
- Oracle and Azul test Coherence regularly on Azul appliances, including the 48-core "Vega 2" chip.

If the server hardware purchase is still in the future, the following are suggested for Coherence (as of December 2006):

The most cost-effective server hardware platform is "commodity x86", either Intel or AMD, with one to two processor sockets and two to four CPU cores per processor socket. If selecting an AMD Opteron system, it is strongly recommended that it be a two processor socket system, since memory capacity is usually halved in a single socket system. Intel "Woodcrest" and "Clovertown" Xeons are **strongly** recommended over the previous Intel Xeon CPUs due to significantly improved 64-bit support, much lower power consumption, much lower heat emission and far better performance. These new Xeons are currently the fastest commodity x86 CPUs, and can support a large memory capacity per server regardless of the processor socket count by using fully buffered memory called "FB-DIMMs".

It is strongly recommended that servers be configured with a minimum of 4GB of RAM. For applications that plan to store massive amounts of data in memory - tens or hundreds of gigabytes, or more - it is recommended to evaluate the cost-effectiveness of 16GB or even 32GB of RAM per server. As of December, 2006, commodity x86 server RAM is readily available in a density of 2GB per DIMM, with higher densities available from only a few vendors and carrying a large price premium; this means that a server with 8 memory slots will only support 16GB in a cost-effective manner. Also

note that a server with a very large amount of RAM will likely need to run more Coherence nodes (JVMs) per server to use that much memory, so having a larger number of CPU cores will help. Applications that are "data heavy" will require a higher ratio of RAM to CPU, while applications that are "processing heavy" will require a lower ratio. For example, it may be sufficient to have two dual-core Xeon CPUs in a 32GB server running 15 Coherence "Cache Server" nodes performing mostly identity-based operations (cache accesses and updates), but if an application makes frequent use of Coherence features such as indexing, parallel queries, entry processors and parallel aggregation, then it will be more effective to have two quad-core Xeon CPUs in a 16GB server - a 4:1 increase in the CPU:RAM ratio.

A minimum of 1000Mbps for networking (for example, Gigabit Ethernet or better) is **strongly** recommended. NICs should be on a high bandwidth bus such as PCI-X or PCIe, and not on standard PCI. In the case of PCI-X having the NIC on an isolated or otherwise lightly loaded 133MHz bus may significantly improve performance.

How many servers are optimal?

Coherence is primarily a scale-out technology. While Coherence can effectively scale-up on large servers by using multiple JVMs per server, the natural mode of operation is to span several small servers (for example, 2-socket or 4-socket commodity servers). Specifically, failover and failback are more efficient in larger configurations. And the impact of a server failure is lessened. As a rule of thumb, a cluster should contain at least four physical servers. In most WAN configurations, each data center will have independent clusters (usually interconnected by Extend-TCP). This will increase the total number of discrete servers (four servers per data center, multiplied by the number of data centers).

Coherence is quite often deployed on smaller clusters (one, two or three physical servers) but this practice has increased risk if a server failure occurs under heavy load. As discussed in the network section of this document, Coherence clusters are ideally confined to a single switch (for example, fewer than 96 physical servers). In some use cases, applications that are compute-bound or memory-bound applications (as opposed to network-bound) may run acceptably on larger clusters.

Also note that given the choice between a few large JVMs and a lot of small JVMs, the latter may be the better option. There are several production environments of Coherence that span hundreds of JVMs. Some care is required to properly prepare for clusters of this size, but smaller clusters of dozens of JVMs are readily achieved. Please note that disabling UDP multicast (by using WKA) or running on slower networks (for example, 100Mbps Ethernet) will reduce network efficiency and make scaling more difficult.

Does it matter how JVMs are distributed among servers?

The following rules should be followed in determining how many servers are required for reliable high availability configuration and how to configure the number of *storage-enabled* JVMs.

1. There must be more than two servers. A grid with only two servers stops being machine-safe as soon as several JVMs on one server is not the same as the number of JVMs on the other server, so even if we start with two servers with equal number of JVMs, losing one JVM will force the grid out of machine-safe state. Four or more machines present the most stable topology, but deploying on just three servers would work if the other rules are adhered to.
2. For a server that has the largest number of JVMs in the cluster, that number of JVMs must not exceed the total number of JVMs on all the other servers in the cluster.

3. A server with the smallest number of JVMs should run at least half the number of JVMs as a server with the largest number of JVMs; this rule is particularly important for smaller clusters.
4. The margin of safety improves as the number of JVMs tends toward equality on all machines in the cluster; this is more of a "rule of thumb" than the preceding "hard" rules.

See also:

- ["Deploying to IBM BladeCenters"](#) on page 32-5
- ["Deploying to Virtual Machines"](#) on page 32-10

Operating System

During development, developers typically use a different operating system than the one that the application will be deployed to.

The top three operating systems for application development using Coherence are, in this order: Windows 2000/XP (~85%), Mac OS X (~10%) and Linux (~5%). The top four operating systems for production deployment are, in this order: Linux, Solaris, AIX and Windows. Thus, it is relatively unlikely that the development and deployment operating system will be the same.

Make sure that regular testing is occurring on the target operating system.

What are the supported and suggested server operating systems for deploying Coherence on?

Oracle tests on and supports various Linux distributions (including customers that have custom Linux builds), Sun Solaris, IBM AIX, Windows Vista/2003/2000/XP, Apple Mac OS X, OS/400 and z/OS. Additionally, Oracle supports customers running HP-UX and various BSD UNIX distributions.

If the server operating system decision is still in the future, the following are suggested for Coherence (as of December 2006):

For commodity x86 servers, Linux distributions based on the Linux 2.6 kernel are recommended. While it is expected that most 2.6-based Linux distributions will provide a good environment for running Coherence, the following are recommended by Oracle: Oracle Unbreakable Linux supported Linux including Oracle Enterprise Linux and Red Hat Enterprise Linux (version 4 or later) and Suse Linux Enterprise (version 10 or later). Oracle also routinely tests using distributions such as RedHat Fedora Core 5 and even Knoppix "Live CD".

Review and follow the instructions in [Chapter 32, "Platform-Specific Deployment Considerations"](#) for the operating system that Coherence will be deployed on.

Avoid using virtual memory (paging to disk).

In a Coherence-based application, primary data management responsibilities (for example, Dedicated Cache Servers) are hosted by Java-based processes. Modern Java distributions do not work well with virtual memory. In particular, garbage collection (GC) operations may slow down by several orders of magnitude if memory is paged to disk. With modern commodity hardware and a modern JVM, a Java process with a reasonable heap size (512MB-2GB) will typically perform a full garbage collection in a few seconds if all of the process memory is in RAM. However, this may grow to many minutes if the JVM is partially resident on disk. During garbage collection, the node will appear unresponsive for an extended period, and the choice for the rest of the

cluster is to either wait for the node (blocking a portion of application activity for a corresponding amount of time), or to mark the unresponsive node as "failed" and perform failover processing. Neither of these is a good option, and so it is important to avoid excessive pauses due to garbage collection. JVMs should be pinned into physical RAM, or at least configured so that the JVM will not be paged to disk.

Note that periodic processes (such as daily backup programs) may cause memory usage spikes that could cause Coherence JVMs to be paged to disk.

See also:

- ["Deploying to AIX"](#) on page 32-1
- ["Deploying to Linux"](#) on page 32-7
- ["Deploying to OS X"](#) on page 32-8
- ["Deploying to Solaris"](#) on page 32-8
- ["Deploying to Windows"](#) on page 32-10
- ["Deploying to z OS"](#) on page 32-11

JVM

During development, developers typically use the latest Sun JVM or a direct derivative such as the Mac OS X JVM.

The main issues related to using a different JVM in production are:

- Command line differences, which may expose problems in shell scripts and batch files;
- Logging and monitoring differences, which may mean that tools used to analyze logs and monitor live JVMs during development testing may not be available in production;
- Significant differences in optimal GC configuration and approaches to GC tuning;
- Differing behaviors in thread scheduling, garbage collection behavior and performance, and the performance of running code.

Make sure that regular testing is occurring on the JVM that will be used in production.

Which JVM configuration options should be used?

JVM configuration options vary over versions and between vendors, but the following are generally suggested:

- Using the `-server` option will result in substantially better performance.
- Using identical heap size values for both `-Xms` and `-Xmx` will yield substantially better performance on Sun and JRockit JVMs, and "fail fast" memory allocation. See the specific Deployment Considerations for various JVMs below.
- For naive tuning, a heap size of 1GB is a good compromise that balances per-JVM overhead and garbage collection performance.
 - Larger heap sizes are allowed and commonly used, but may require tuning to keep garbage collection pauses manageable.
- JVMs that experience an `OutOfMemoryError` can be left in an indeterministic state which can have adverse effects on a cluster. We recommend configuring JVMs to exit upon encountering an `OutOfMemoryError` instead of allowing the

JVM to attempt recovery. See the specific Deployment Considerations below for instructions on configuring this on common JVMs.

What are the supported and suggested JVMs for deploying Coherence on?

In terms of Oracle Coherence versions:

- Coherence is supported on Sun JDK 1.5 and 1.6 and JVMs corresponding to those versions.

Often the choice of JVM is dictated by other software. For example:

- IBM only supports IBM WebSphere running on IBM JVMs. Most of the time, this is the IBM "Sovereign" or "J9" JVM, but when WebSphere runs on Sun Solaris/Sparc, IBM builds a JVM using the Sun JVM source code instead of its own.
- Oracle WebLogic typically includes a JVM which is intended to be used with it. On some platforms, this is the Oracle WebLogic JRockit JVM.
- Apple Mac OS X, HP-UX, IBM AIX and other operating systems only have one JVM vendor (Apple, HP and IBM respectively).
- Certain software libraries and frameworks have minimum Java version requirements because they take advantage of relatively new Java features.

On commodity x86 servers running Linux or Windows, the Sun JVM is recommended. Generally speaking, the recent update versions are recommended. For example:

Note: Oracle recommends testing and deploying using the latest supported Sun JVM based on your platform and Coherence version. It has been observed that running Coherence on 1.5 and later JVMs exhibits significant performance improvements as compared to running on older JVMs.

Basically, at some point before going to production, a JVM vendor and version should be selected and well tested, and absent any flaws appearing during testing and staging with that JVM, that should be the JVM that is used when going to production. For applications requiring continuous availability, a long-duration application load test (for example, at least two weeks) should be run with that JVM before signing off on it.

Review and follow the instructions in [Chapter 32, "Platform-Specific Deployment Considerations"](#) for the JVM that Coherence will be deployed on.

Must all nodes run the same JVM vendor and version?

No. Coherence is pure Java software and can run in clusters composed of any combination of JVM vendors and versions, and Oracle tests such configurations.

Note that it is *possible* for different JVMs to have slightly different serialization formats for Java objects, meaning that it is possible for an incompatibility to exist when objects are serialized by one JVM, passed over the wire, and a different JVM (vendor and/or version) attempts to deserialize it. Fortunately, the Java serialization format has been very stable for several years, so this type of issue is extremely unlikely. However, it is highly recommended to test mixed configurations for consistent serialization before deploying in a production environment.

See also:

- ["Deploying to Oracle JRockit JVMs"](#) on page 32-2

- ["Deploying to IBM JVMs"](#) on page 32-6
- ["Deploying to Sun JVMs"](#) on page 32-9

Java Security Manager

The minimum set of privileges required for Coherence to function are specified in the `security.policy` file which is included as part of the Coherence installation. This file can be found in `coherence/lib/security/security.policy`. If using the Java Security Manager these privileges must be granted in order for Coherence to function properly.

Application Instrumentation

Be cautious when using instrumented management and monitoring solutions.

Some Java-based management and monitoring solutions use instrumentation (for example, bytecode-manipulation and `ClassLoader` substitution). While there are no known open issues with the latest versions of the primary vendors, Oracle has observed issues in the past.

Coherence Editions and Modes

During development, use the development mode.

The Coherence download includes a fully functional Coherence product supporting all editions and modes. The default configuration is for Grid Edition in Development mode.

Coherence may be configured to operate in either development or production mode. These modes do not limit access to features, but instead alter some default configuration settings. For instance, development mode allows for faster cluster startup to ease the development process.

It is recommended to use the development mode for all pre-production activities, such as development and testing. This is an important safety feature, because Coherence automatically prevents these nodes from joining a production cluster. The production mode must be explicitly specified when using Coherence in a production environment.

Coherence may be configured to support a , based on the customer license agreement.

Only the edition and the number of licensed CPUs specified within the customer license agreement can be used in a production environment.

When operating outside of the production environment it is allowable to run any Coherence edition. However, it is recommended that only the edition specified within the customer license agreement be used. This will protect the application from unknowingly making use of unlicensed features.

All nodes within a cluster must use the same license edition and mode.

Starting with Oracle Coherence 3.4, customer-specific license keys are no longer part of product deployment.

Be sure to obtain enough licenses for the all the cluster members in the production environment. The servers hardware configuration (number or type of processor

sockets, processor packages or CPU cores) may be verified using `ProcessorInfo` utility included with Coherence.

Example 33–1 Verifying Hardware Configuration

```
java -cp coherence.jar com.tangosol.license.ProcessorInfo
```

If the result of the `ProcessorInfo` program differs from the licensed configuration, send the program's output and the actual configuration to the "support" email address at Oracle.

How are the edition and mode configured?

There is a `<license-config>` configuration section in `tangosol-coherence.xml` (located in `coherence.jar`) for edition and mode related information.

Example 33–2 Sample Coherence License Configuration

```
<license-config>
  <edition-name system-property="tangosol.coherence.edition">GE</edition-name>
  <license-mode system-property="tangosol.coherence.mode">dev</license-mode>
</license-config>
```

In addition to preventing mixed mode clustering, the `license-mode` also dictates the operational override file which will be used. When in `dev` mode the `tangosol-coherence-override-dev.xml` file will be used, whereas the `tangosol-coherence-override-prod.xml` file will be used when the `prod` mode is specified. As the mode controls which override file is used, the `<license-mode>` configuration element is only usable in the base `tangosol-coherence.xml` file and not within the override files.

These elements are defined by the corresponding `coherence.dtd` in `coherence.jar`. It is possible to specify this edition on the command line using the command line override:

```
-Dtangosol.coherence.edition=RTC
```

Valid values are listed in [Table 33–1](#):

Table 33–1 Valid tangosol.coherence.edition Values

Value	Coherence Edition	Compatible Editions
GE	Grid Edition	RTC, DC
EE	Enterprise Edition	DC
SE	Standard Edition	DC
RTC	Real-Time Client	GE
DC	Data Client	GE, EE, SE

- Note: clusters running different editions may connect by using Coherence*Extend as a Data Client.

For more information on overrides, see [Appendix C, "Command Line Overrides"](#).

Ensuring that RTC nodes don't use Coherence TCMP

The RTC nodes can connect to clusters using either Coherence TCMP or Coherence Extend. If the intention is to connect over Extend it is advisable to disable TCMP on

that node to ensure that it only connects by using Extend. TCMP may be disabled using the system property `tangosol.coherence.tcmp.enabled`. See the `<enabled>` subelement of "[packet-publisher](#)" on page A-52.

Coherence Operational Configuration

Operational configuration relates to the configuration of Coherence at the cluster level including such things as:

- Cluster and member descriptors
- Network settings
- Security
 - Membership restrictions
 - Access Control
 - Encryption

The operational aspects are normally configured by using the `tangosol-coherence-override.xml` file. See "[Operational Configuration Deployment Descriptors](#)" on page A-1 for more information on this file.

The contents of this file will likely differ between development and production. It is recommended that these variants be maintained independently due to the significant differences between these environments. The production operational configuration file should not be the responsibility of the application developers, instead it should fall under the jurisdiction of the systems administrators who are far more familiar with the workings of the production systems.

All cluster nodes should use the same operational configuration descriptor. A centralized configuration file may be maintained and accessed by specifying the file's location as a URL using the `tangosol.coherence.override` system property. Any node specific values may be specified by using system properties. See [Appendix C, "Command Line Overrides"](#) for more information on the properties.

The override file should contain only the subset of configuration elements which you want to customize. This will not only make your configuration more readable, but will allow you to take advantage of updated defaults in future Coherence releases. All override elements should be copied exactly from the original `tangosol-coherence.xml`, including the `id` attribute of the element.

Member descriptors may be used to provide detailed identity information that is useful for defining the location and role of the cluster member. Specifying these items will aid in the management of large clusters by making it easier to identify the role of a remote nodes if issues arise.

Coherence Cache Configuration

Cache configuration relates to the configuration of Coherence at a per-cache level including such things as:

- Cache topology (`<distributed-scheme>`, `<replicated-scheme>`, `<near-scheme>`, and so on)
- Cache capacities (see `<high-units>` subelement of `<local-scheme>`)
- Cache redundancy level (`<backup-count>` subelement of `<distributed-scheme>`)

The cache configuration aspects are normally configured by using the `coherence-cache-config.xml` file. See "[Cache Configuration Deployment Descriptor](#)" on page B-1 for more information this file.

The default `coherence-cache-config.xml` file included within `coherence.jar` is intended only as an example and is not suitable for production use. It is suggested that you produce your own cache configuration file with definitions tailored to your application needs.

All cluster nodes should use the same cache configuration descriptor. A centralized configuration file may be maintained and accessed by specifying the file's location as a URL using the `tangosol.coherence.cacheconfig` system property.

Choose the cache topology which is most appropriate for each cache's usage scenario.

It is important to size limit your caches based on the allocated JVM heap size. Even if you never expect to fully load the cache, having the limits in place will help protect your application from `OutOfMemoryExceptions` if your expectations are later negated.

For a 1GB heap that at most $\frac{3}{4}$ of the heap be allocated for cache storage. With the default one level of data redundancy this implies a per server cache limit of 375MB for primary data, and 375MB for backup data. The amount of memory allocated to cache storage should fit within the tenured heap space for the JVM. See Sun's GC tuning guide for details.

It is important to note that when multiple cache schemes are defined for the same cache service name the first to be loaded will dictate the service level parameters. Specifically the `<partition-count>`, `<backup-count>`, and `<thread-count>` subelements of `<distributed-scheme>` are shared by all caches of the same service.

For multiple caches which use the same cache service it is recommended that the service related elements be defined only once, and that they be inherited by the various cache-schemes which will use them.

If you want different values for these items on a cache by cache basis then multiple services may be configured.

For partitioned caches Coherence will evenly distribute the storage responsibilities to all cache servers, regardless of their cache configuration or heap size. For this reason it is recommended that all cache server processes be configured with the same heap size. For machines with additional resources multiple cache servers may be used to effectively make use of the machine's resources.

To ensure even storage responsibility across a partitioned cache the `<partition-count>` subelement of `<distributed-scheme>`, should be set to a prime number which is at least the square of the number of cache servers which will be used.

For caches which are backed by a cache store it is recommended that the parent service be configured with a thread pool as requests to the cache store may block on I/O. The pool is enabled by using the `<thread-count>` subelement of `<distributed-scheme>` element. For non-`CacheStore`-based caches more threads are unlikely to improve performance and should left disabled.

Unless explicitly specified all cluster nodes will be storage enabled, that is, will act as cache servers. It is important to control which nodes in your production environment will be storage enabled and storage disabled. The `tangosol.coherence.distributed.localstorage` system property may be used to control this, setting it to either `true` or `false`. Generally, only dedicated

cache servers, all other cluster nodes should be configured as storage disabled. This is especially important for short lived processes which may join the cluster perform some work, and exit the cluster, having these nodes as storage enabled will introduce unneeded re-partitioning. See the `<local-storage>` subelement of `<distributed-scheme>` for more information about the system property.

Large Cluster Configuration

Are there special considerations for large clusters?

- The general recommendation for the `<partition-count>` subelement of `<distributed-scheme>` is to be a prime number close to the square of the number of storage enabled nodes. While is a good suggestion for small to medium sized clusters, for large clusters it can add too much overhead. For clusters exceeding 128 storage enabled JVMs, the partition count should be fixed, at roughly 16,381.
- Coherence clusters which consist of over 400 TCMP nodes need to increase the default maximum packet size Coherence will use. The default of 1468 should be increased relative to the size of the cluster, that is, a 600 node cluster would need the maximum packet size increased by 50%. A simple formula is to allow four bytes per node, that is, `maximum_packet_size >= maximum_cluster_size * 4B`. The maximum packet size is configured as part of the coherence operational configuration file, see "`packet-size`" on page A-54 for details on changing this setting.
- For large clusters which have hundreds of JVMs it is also recommended that `<multicast-listener>` be enabled, as it will allow for more efficient cluster wide transmissions. These cluster wide transmissions are rare, but when they do occur multicast can provide noticeable benefits in large clusters.

Death Detection

The Coherence death detection algorithms are based on sustained loss of connectivity between two or more cluster nodes. When a node identifies that it has lost connectivity with any other node it will consult with other cluster nodes to determine what action should be taken.

In attempting to consult with others, the node may find that it cannot communicate with any other nodes, and will assume that it has been disconnected from the cluster. Such a condition could be triggered by physically unplugging a node's network adapter. In such an event the isolated node will restart it's clustered services and attempt to rejoin the cluster.

If connectivity with other cluster nodes remains unavailable, the node may (depending on well known address configuration) form a new isolated cluster, or continue searching for the larger cluster. In either case when connectivity is restored the previously isolated cluster nodes will rejoin the running cluster. As part of rejoining the cluster, the nodes former cluster state is discarded, including any cache data it may have held, as the remainder of the cluster had already taken on ownership of that data (restoring from backups).

Without connectivity it is obviously not possible for a node to identify the state of other nodes. This means that from the point of view of a single node, local network adapter failure and network wide switch failure look identical, and are thus handled in the same way, as described above. The important difference is that in the case of a

switch failure all nodes are attempting to re-join the cluster, which is the equivalent of a full cluster restart, and all prior state and data is dropped.

Obviously dropping all data is not desirable, and thus if you want to avoid this as part of a sustained switch failure you must take additional precautions. Options include:

- **Extend allowable outage duration:** The maximum time a node(s) may be unresponsive before being removed from the cluster is configured by using the `<timeout-milliseconds>` subelement of `<packet-delivery>`, and defaults to one minute for production configurations. Increasing this value will allow the cluster to wait longer for connectivity to return. The downside of increasing this value it may also take longer to handle the case where just a single node has lost connectivity.
- **Persist data to external storage:** By using a Read Write Backing Map, the cluster persists data to external storage, and can retrieve it after a cluster restart. So long as write-behind is disabled (the `<write-delay>` subelement of `<read-write-backing-map-scheme>`) no data would be lost in the event of a switch failure. The downside here is that synchronously writing through to external storage increases the latency of cache update operations, and the external storage may become a bottleneck.
- **Delay node restart:** The cluster death detection action can be configured to delay the node restart until connectivity is restored. By delaying the restart until connectivity is restored an isolated node is allowed to continue running with whatever data it had available at the time of disconnect. When connectivity is restored the nodes will detect each other and form a new cluster. In forming a new cluster all but the most senior node will be required to restart. This results in behavior which is nearly identical to the default behavior because the majority of the nodes will restart, and drop their data. It may be beneficial for cases in which replicated caches are in use as the senior most node's copy of the data will survive the restart. To enable the delayed restart the `tangosol.coherence.departure.threshold` system property must be set to a value that is greater than the size of the cluster.

Note: When running on Microsoft Windows it is also necessary to ensure the Windows does not disable the network adapter when it is disconnected. To do this, add the following Windows registry DWORD, setting it to 1: `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\DisableDHCPMediaSense`. This setting also affects static IPs despite the name.

- **Add network level fault tolerance:** Adding a redundant layer to the cluster's network infrastructure allows for individual pieces of networking equipment to fail without disrupting connectivity. This is commonly achieved by using at least two network adapters per machine, and having each adapter connected to a separate switch. This is not a feature of Coherence but rather of the underlying operating system or network driver. The only change to Coherence is that it should be configured to bind to the virtual rather than physical network adapter. This form of network redundancy goes by different names depending on the operating system, see Linux bonding, Solaris trunking and Windows teaming for further details.

tangosol-license.xml Deprecated

As of Coherence 3.4, the `tangosol-license.xml` file is no longer used.

Part VI

Managing Coherence

Part VI contains the following chapters:

- [Chapter 34, "How to Manage Coherence Using JMX"](#)
- [Chapter 35, "JMX Reporter"](#)
- [Chapter 36, "How to Create a Custom Report"](#)
- [Chapter 37, "How to Modify Report Batch"](#)
- [Chapter 38, "Analyzing Reporter Content"](#)
- [Chapter 39, "How to Run a Report on Demand"](#)
- [Chapter 40, "Configuring Custom MBeans"](#)
- [Chapter 41, "How to Manage Custom MBeans Within the Cluster"](#)

How to Manage Coherence Using JMX

Coherence includes facilities for managing and monitoring Coherence resources by using the Java Management Extensions (JMX) API. JMX is a Java standard for managing and monitoring Java applications and services. It defines a management architecture, design patterns, APIs, and services for building general solutions to manage Java-enabled resources. This section assumes familiarity with JMX terminology. Those that are new to JMX, should refer to following article:

<http://java.sun.com/developer/technicalArticles/J2SE/jmx.html>

For details on the various MBean types registered by Coherence clustered services, see the `com.tangosol.net.management.Registry` interface in *Oracle Coherence Java API Reference* or see [Appendix E, "Coherence MBeans Reference."](#)

Note: Coherence Enterprise Edition and higher support clustered JMX, allowing access to JMX statistics for the entire cluster from any member. Coherence Standard Edition provides only local JMX information.

The following sections are included in this chapter:

- [Configuring the Coherence Management Framework](#)
- [Accessing Coherence MBeans](#)
- [Using Coherence MBeanConnector to Access MBeans](#)

Configuring the Coherence Management Framework

In general, the Coherence management framework is configured within the `<management-config>` element in a Coherence operational override file (`tangosol-coherence-override.xml`). For detailed information on each of the elements that can be configured for the Coherence management framework, see ["management-config"](#) on page A-32:

The following topics are included in this section:

- [Enabling JMX Management](#)
- [Filtering MBeans](#)
- [Configuring Management Refresh](#)

Enabling JMX Management

JMX management is enabled by configuring one or more nodes to host an MBean server which manages the managed objects of all the other cluster nodes. The use of dedicated JMX cluster members is a common pattern because it avoids loading JMX software into every single cluster member while still providing fault-tolerance should a single JMX member fail.

JMX management in the cluster is disabled by default. To enable JMX management, add a `<managed-nodes>` element within the `<management-config>` element in an operational override file. The `<managed-nodes>` element specifies whether a cluster node's JVM has an in-process MBean server and if so, whether the node allows management of other nodes' managed objects. The following example enables an MBean server on the current cluster node and is used to manage both local and remote managed objects:

```
<coherence>
  <management-config>
    <managed-nodes>all</managed-nodes>
  </management-config>
</coherence>
```

The `<managed-nodes>` element supports the following values:

- `none`—No MBean server is instantiated.
- `local-only`—Manage only MBeans which are local to the cluster node (that is, within the same JVM).
- `remote-only`—Manage MBeans on other remotely manageable cluster nodes. See `<allowed-remote-management>` subelement. This option requires Coherence Enterprise Edition or higher.
- `all`—Manage both local and remotely manageable cluster nodes. This option requires Coherence Enterprise Edition or higher.

By default, all nodes in the cluster will allow a remote MBean server to manage its MBeans. To restrict remote management of a node's MBeans, the `<allow-remote-management>` element must be set to `false`. The following example restricts a node from having its MBeans managed by a remote MBean server.

```
<coherence>
  <management-config>
    <allow-remote-management>false</allow-remote-management>
  </management-config>
</coherence>
```

Values for the `<managed-nodes>` and `<allow-remote-management>` elements can also be set using Java system properties:

```
-Dtangosol.coherence.management=all
-Dtangosol.coherence.management.remote=true
```

Filtering MBeans

The Coherence management framework provides the ability to filter MBeans before they are registered in the MBean server. An out-of-the box MBean filter is provided and custom filters can be created as required. The included MBean filter (`com.tangosol.net.management.ObjectNameExcludeFilter`) is used to exclude MBeans from being registered based on their JMX object name using standard

regex patterns. As configured out-of-the box, the filter excludes some platform MBeans from being registered in the management framework. MBean filters are defined using the `<mbean-filter>` element within the `<management-config>` element in an operational override file.

The following example shows the out-of-the box configuration:

```
<mbean-filter>
  <class-name>com.tangosol.net.management.ObjectNameExcludeFilter</class-name>
  <init-params>
    <init-param>
      <param-type>string</param-type>
      <param-value>
        .*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>
```

To enable the management service or platform MBeans, remove the corresponding object name from the list of names in the `<param-value>` element. To exclude an MBean from being registered, add the MBean object name to the list. The exclusion list may also be entered using the `tangosol.coherence.management.exclude` Java system property.

Configuring Management Refresh

The Coherence management framework provides the `<refresh-expiry>`, `<refresh-policy>`, and `<refresh-timeout>` elements to control the latency of management information. The elements are defined within the `<management-config>` element and should be set together.

Setting the Management Refresh Expiry

The `<refresh-expiry>` element specifies the minimum time interval between the remote retrieval of management information from remote nodes. The value of this element must be in the following format:

```
[\d]+[.[\d]+]?[MS|ms|S|s|M|m|H|h|D|d]?
```

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 1s. The refresh policy can also be set using the `tangosol.coherence.management.refresh.expiry` Java system property.

Setting the Management Refresh Policy

The `<refresh-policy>` element is used to specify the method which is used to refresh remote management information. Each policy uses a different refresh algorithm that can improve latency based on MBean usage patterns. [Table 34–1](#) describes each policy.

Table 34–1 Refresh Policies

Setting	Description
<code>refresh-ahead</code> (default)	MBeans are refreshed before they are requested based on prior usage patterns after the expiry delay has passed. This setting can reduce latency of the management information with a minor increase in network consumption. This setting is best when MBeans are accessed in a repetitive/programmatic pattern.
<code>refresh-behind</code>	Each MBean will be refreshed after the data is accessed. This method ensures optimal response time. However, the information returned will be offset by the last refresh time.
<code>refresh-expired</code>	Each MBean will be refreshed from the remote node when it is accessed and the expiry delay has passed from the last refresh. This setting is best used when MBeans are accessed in a random pattern.

The refresh policy can also be set using the `tangosol.coherence.management.refresh.policy` Java system property.

Setting the Management Refresh Timeout

The `<refresh-time>` element is used to specify 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>` value. The value of this element must be in the following format:

```
[\\d]+[\\.][\\d]+]?[MS|ms|S|s|M|m|H|h|D|d]?
```

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 250ms. The refresh policy can also be set using the `tangosol.coherence.management.refresh.timeout` Java system property.

Accessing Coherence MBeans

JMX management must be enabled on at least one Coherence node before accessing Coherence MBeans. See ["Configuring the Coherence Management Framework"](#) on page 34-1.

There are two options for viewing and manipulating Coherence MBeans: the JConsole utility and the HTML Adapter Web Application.

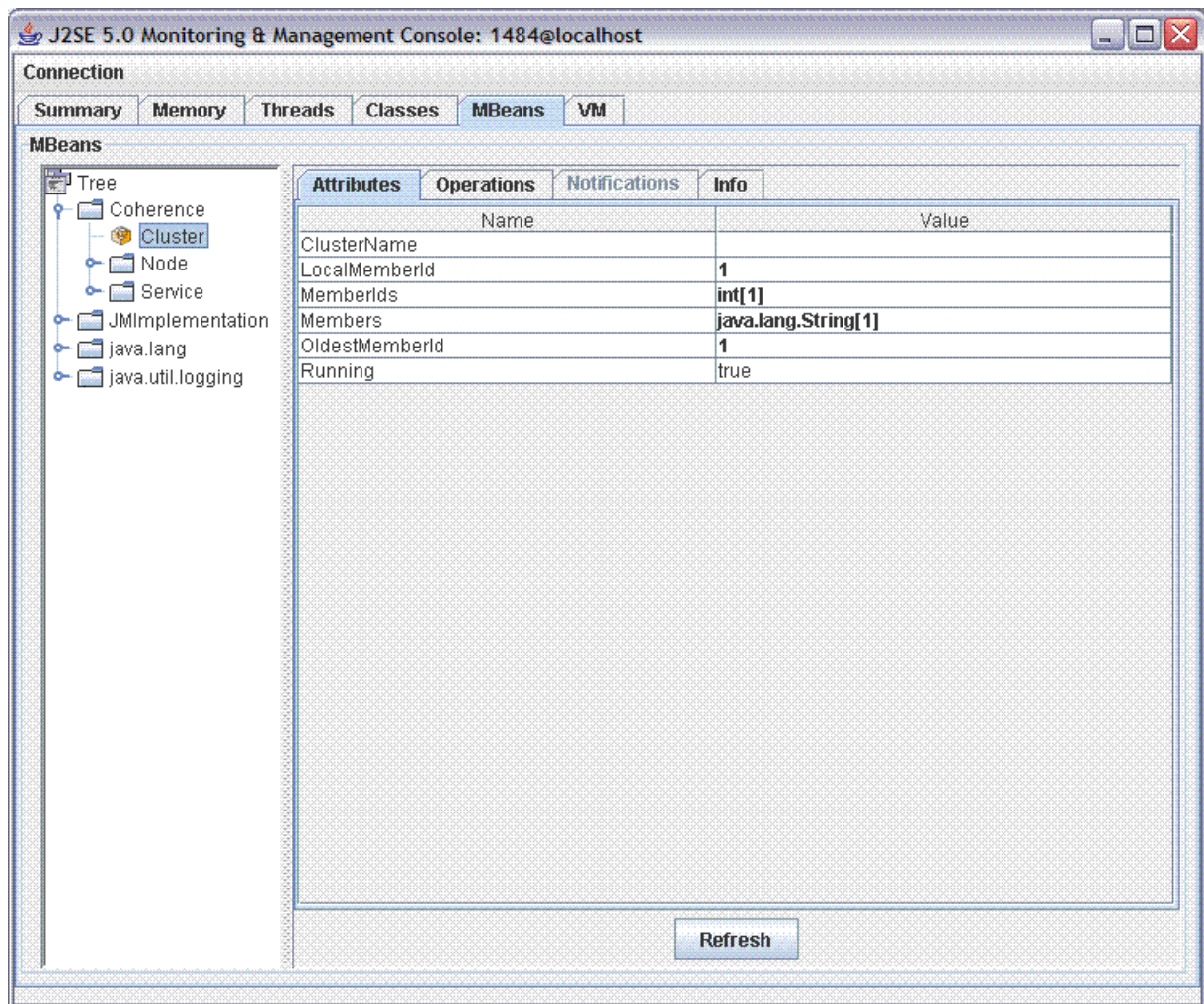
Viewing MBeans Using the JConsole Utility

The JConsole utility (located in the the JDK /bin directory) can be used to view and manipulate Coherence MBeans. To do so, set the following system properties when starting a Coherence node:

```
-Dcom.sun.management.jmxremote
-Dtangosol.coherence.management=all
-Dtangosol.coherence.management.remote=true
```

When the node has started, launch the JConsole utility and open a new connection to the JVM process running the Coherence command line application:

Figure 34–1 JConsole Utility to Access Coherence MBeans



Viewing MBeans Using the HTML Adapter Application

The HTML Adapter Web Application uses the HTTP adapter (HtmlAdaptorServer) that is shipped as part of the JMX reference implementation (jmxtools.jar). To run the Web application, start a Coherence node as follows:

On Windows:

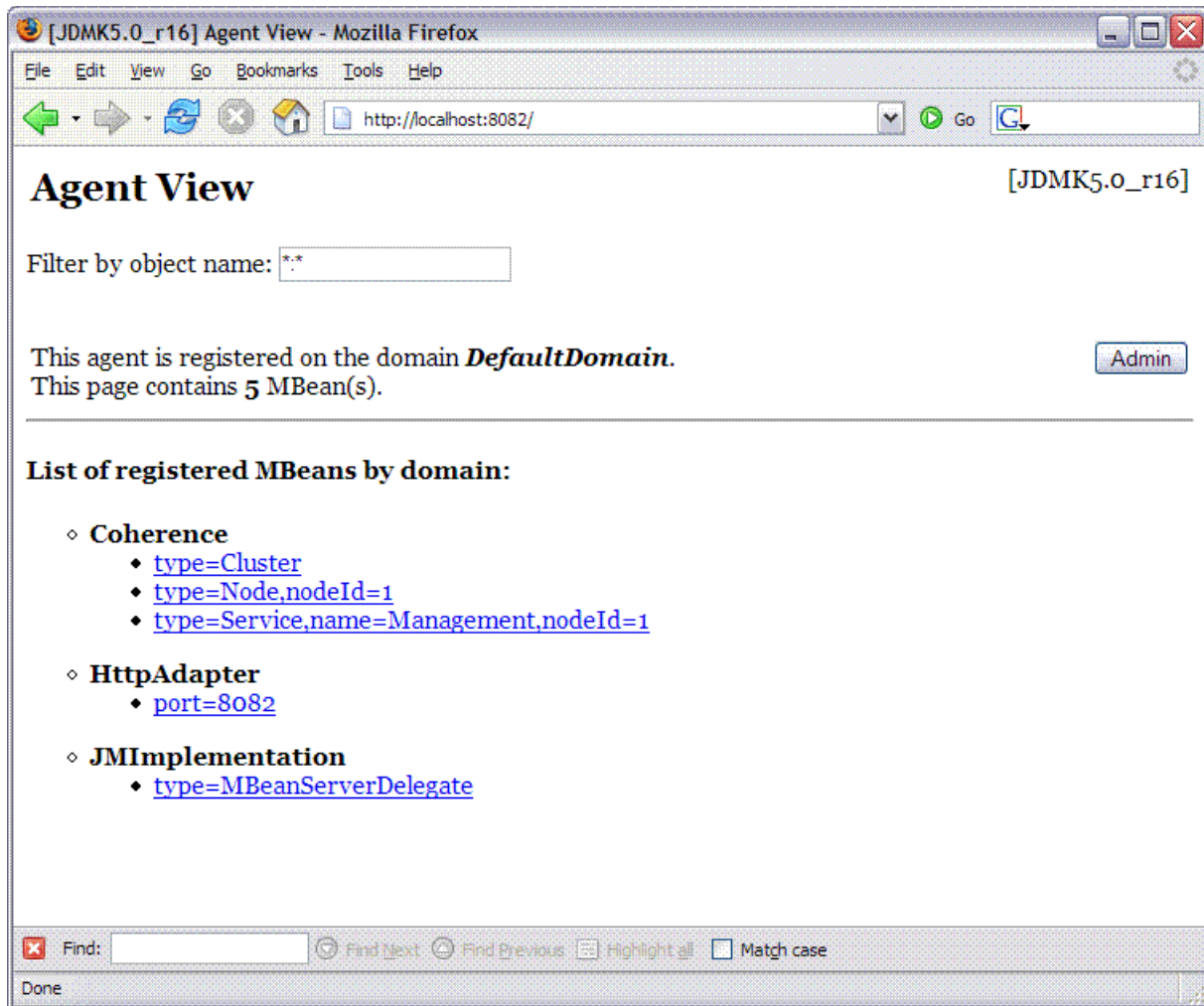
```
java -cp jmxri.jar;jmxtools.jar;coherence.jar -Dtangosol.coherence.management=all
-Dtangosol.coherence.management.remote=true com.tangosol.net.CacheFactory
```

On UNIX:

```
java -cp jmxri.jar:jmxtools.jar:coherence.jar -Dtangosol.coherence.management=all
-Dtangosol.coherence.management.remote=true com.tangosol.net.CacheFactory
```

After the Coherence command line application starts, enter `jmx 8082`. This starts an HTTP adaptor on `http://localhost:8082` in the cluster node's JVM and makes the cluster node an MBeanServer host:

Figure 34-2 Viewing the HttpAdapter Web Application in a Browser



Using Coherence MBeanConnector to Access MBeans

Coherence ships with a program to launch a cluster node as a dedicated MBean server host. This program provides access to Coherence MBeans by using the JMX Remote API using RMI or the HTTP server provided by Sun's JMX RI. The RMI and HTTP ports can be configured, allowing for access through a firewall. The server is started using the following command (note that it is broken up into multiple lines here only for formatting purposes; this is a single command entered on one line):

```
java -Dtangosol.coherence.management=all
-Dcom.sun.management.jmxremote.ssl=false
```

```
-Dcom.sun.management.jmxremote.authenticate=false
-cp coherence.jar;jmxri.jar;jmxtools.jar
com.tangosol.net.management.MBeanConnector [-http -rmi]
```

To allow access by using JMX RMI, include the `-rmi` flag. To allow access by using HTTP and a Web browser, include the `-http` flag. Both flags may be included; however at least one must be present for the node to start.

Table 34–2 describes optional properties that can be used for JMX RMI configuration:

Table 34–2 *Optional Properties that can be used for JMX RMI Configuration*

Property	Description
tangosol.coherence.management.remote.host	The host that the JMX server will bind to. Default is <code>localhost</code> . (NOTE: on Redhat Linux this may have to be changed to the host name or IP address)
tangosol.coherence.management.remote.registryport	The port used for the JMX RMI registry. Default is 9000.
tangosol.coherence.management.remote.connectionport	The port used for the JMX RMI connection. Default is 3000.

Table 34–3 describes optional properties that can be used for HTTP configuration.

Table 34–3 *Optional Properties that can be used for Http Configuration*

Property	Description
tangosol.coherence.management.remote.httpport	The port used for the HTTP connection. Default is 8888.

To connect by using JConsole with default settings, use the following command:

```
jconsole service:jmx:rmi://localhost:3000/jndi:rmi://localhost:9000/server
```

To connect by using HTTP with default settings, use the following URL:

```
http://localhost:8888
```

Note: Refer to the JMX Agent documentation below to setup secure access using authentication and SSL:

Java 1.5:

<http://java.sun.com/j2se/1.5.0/docs/guide/management/agent.html>

Java 1.6:

<http://java.sun.com/javase/6/docs/technotes/guides/management/agent.html>

JMX Reporter

Coherence provides a JMX reporting capability (the Reporter). The Reporter provides out-of-the-box reports that help administrators and developers manage capacity and trouble shoot problems. Custom reports can also be created.

Note: Plan for archiving and removing. Due to the volume of the information created by the Reporter, you must have a plan for archiving and/or removing the results BEFORE starting the Reporter.

Basic Configuration

Enabling the Reporter with basic content requires setting the system properties:

[Example 35–1](#) illustrates the properties on the "management" node.

Example 35–1 System Properties for Reporter on the "Management" Node

```
-Dtangosol.coherence.management.report.autostart=true  
-Dtangosol.coherence.management=all  
-Dcom.sun.management.jmxremote
```

[Example 35–2](#) illustrates the properties on the "managed" node.

Example 35–2 System Properties for Reporter on the "Managed" Node

```
-Dtangosol.coherence.management.remote=true
```

Basic configuration will create a single Reporter node that will log the JMX statistics for all nodes in the cluster. The log files will be placed in the working directory of the application.

Administration

The JMX Reporter is managed through an MBean under the Coherence Domain. The Reporter MBean provides information related to the status and performance of the Reporter. The MBean also provides the capability to start and stop the service and run a report on demand.

[Figure 35–1](#) illustrates the attributes available to the Reporter MBean. The JConsole is being used to view the MBean.

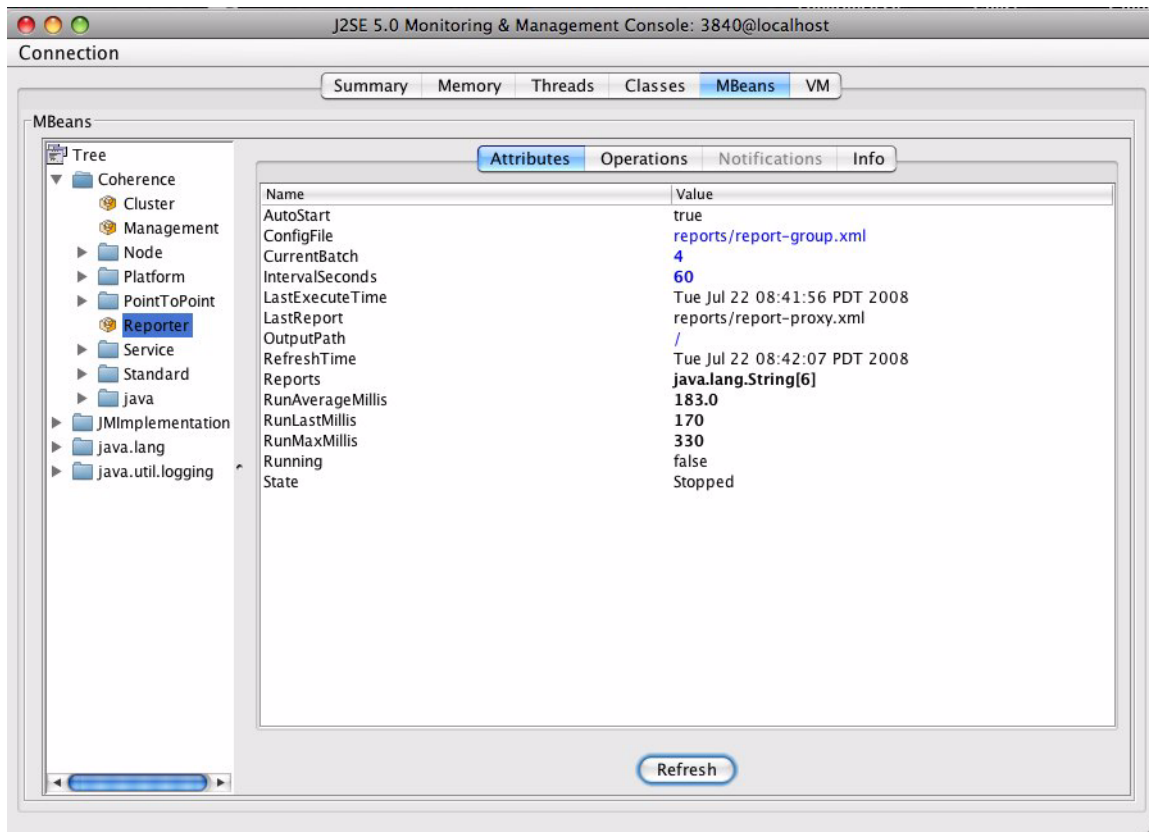
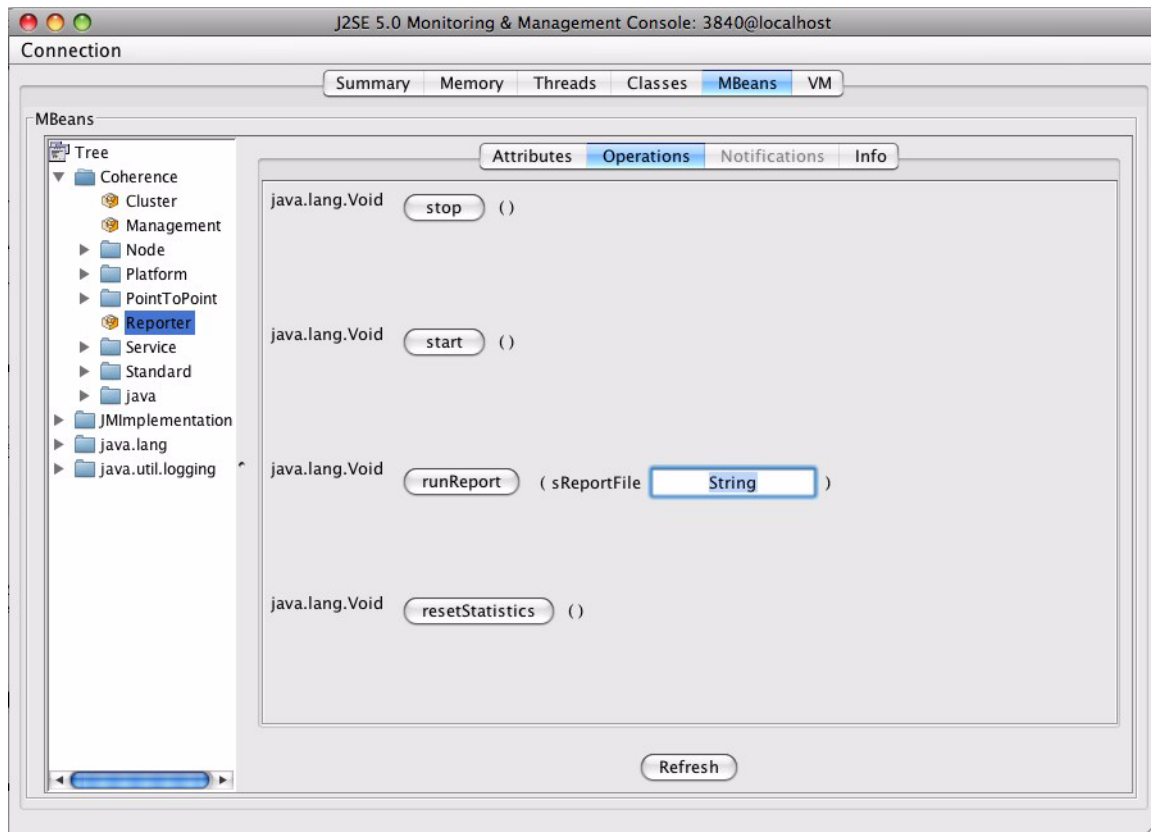
Figure 35–1 Reporter Attributes in JConsole

Figure 35–2 illustrates the operations available to the Reporter MBean. For a full description of the Reporter Attributes see the `Reporter` section of the javadoc.

Figure 35–2 Reporter Operations in JConsole

Data Analysis

Seven files are created each hour by the Reporter. Each file is prefixed with the date and hour the report was executed in a `YYYYMMDDHH` format. This allows for easy location and purging of unwanted information. The files generated are described in [Table 35–1](#):

Table 35–1 File Names Generated by Reporter

File Name	Description
<code>YYYYMMDDHH-memory-status.txt</code>	Contains memory and garbage collection information about each node.
<code>YYYYMMDDHH-network-health.txt</code>	Contains the publisher success rates and receiver success rates for the entire grid
<code>YYYYMMDDHH-network-health-detail.txt</code>	Contains the publisher success rates and receiver success rates for each node
<code>YYYYMMDDHH-node.txt</code>	Contains the list of nodes that were members of the grid
<code>YYYYMMDDHH-service.txt</code>	Contains Request and Task information for each service.
<code>YYYYMMDDHH-proxy.txt</code>	Contains utilization information about each proxy node in the grid
<code>YYYYMMDDHH-cache-usage.txt</code>	Contains cache utilization (put, get, and so on) statistic for each cache

See [Chapter 38, "Analyzing Reporter Content"](#) for a complete description of the data contained in each file. For reports specific to Coherence*Web, see "Running Performance Reports" in *Oracle Coherence User's Guide for Oracle Coherence*Web*.

Advanced Configuration

Creating Custom Reports

1. Create the custom report configuration file. See [Chapter 36, "How to Create a Custom Report."](#)
2. Update report batch to execute the report. See [Chapter 37, "How to Modify Report Batch."](#)
3. Run on demand. See [Chapter 39, "How to Run a Report on Demand."](#)

Running Reporter in a Distributed Configuration

A distributed configuration is only recommended in situations where grid stability is an issue. In this configuration, the distributed reporters will run independently, and the execution times will not align. Therefore, grid level analysis is extremely difficult but node level analysis during periods when nodes may be leaving or joining the grid will still be available.

When running in distributed mode, each node logs local JMX statistics while allowing for centralized management of the Reporters. To enable this configuration set the following system properties

On the "managing" node:

Example 35–3 System Properties for Reporter in Distributed Mode on the "Managing" Node

```
-Dtangosol.coherence.management.report.autostart=false  
-Dtangosol.coherence.management.report.distributed=true  
-Dtangosol.coherence.management=all  
-Dcom.sun.management.jmxremote
```

On the "managed" node:

Example 35–4 System Properties for Reporter in Distributed Mode on the "Managed" Node

```
-Dtangosol.coherence.management.report.autostart=true  
-Dtangosol.coherence.management.report.distributed=true  
-Dtangosol.coherence.management=local-only  
-Dtangosol.coherence.management.remote=true
```

How to Create a Custom Report

The Coherence reporting feature provides a capable query definition that allows for any information residing in the Coherence JMX data source to be logged to a text file. After a custom report has been created, it can be included in a report batch and executed on a specified time interval by the `ReportControl` MBean. For a complete description of the report configuration XML file see the `report-config.dtd` which is packaged in the `coherence.jar` file.

Configuring a Report File

To correctly generate the report file, several elements must be configured. These elements are described in [Table 36-1](#).

Table 36-1 *Elements to Configure an Output File*

Element	Optional/ Required	Description
<file-name>	Required	The file name to create or update when the report is executed. For more information, see " file-name Element ".
<delim>	Optional	The column delimiter for the report. Valid values are {tab}, {space} or a printable character. The default value is {tab}. If a string longer than one character is entered, the first character in the string is used.
<hide-headers>	Optional	A boolean element to determine if the headers are to be included in the report. If <code>true</code> , the column headers and the report description are not included in the file. The default value is <code>false</code> .

file-name Element

The value of this element will have the output path from the `<report-path>` element pre-pended to it and the report will be generated in this location. If the Coherence node cannot access this path, then the file will not be created.

file-name Macros

There are pre-defined macros that you can use with the `file-name` element. These macros can add a node name, a batch number, or a date to the generated file name.

Table 36–2 *Macros that can be Used with the file-name Element*

Macro	Description
batch	Will include a batch Identifier into the filename of the report. If the information is kept for a short amount of time or is frequently uploaded into an RDBMS.
date	Will include the date (with the format YYYYMMDD), into the file name of the report. This is used mostly when the data will only be kept for a certain period and then will be discarded.
node	Will include the node ID into the file name string. This configuration setting is helpful when many nodes are executing the same report and the output files will be integrated for the analysis.

file-name Macro Examples

The following example will create a file `20090101_network_status.txt` on January 1, 2009. The filename will change with the system time on the node executing the report.

```
<file-name>{{date}}_network_status.txt</file-name>
```

The following example will create a file `00012_network_status.txt` when the report is executed on node 12. Note that due to the volatile nature of the Node Id, long term storage in this manner is not recommended.

```
<file-name>{node}_network_status.txt</file-name>
```

The following example will create a file `0000000021_network_status.txt` on the 21st execution of the report. Note that due to the volatile nature of the batch, long term storage in this manner is not recommended.

```
<file-name>{batch}_network_status.txt</file-name>
```

Specifying Data Columns

Data columns can be sourced from JMX Attributes, ObjectName key part, JMX composite attributes, JMX joined attributes, Report macros, and Report Constants.

How to Include an Attribute

To include data from MBeans returned from the query-pattern, the report must have a column with an attribute source. This is the most common item that will be included in the report.

[Example 36–1](#) illustrates how to include the RoleName attribute from the query pattern `Coherence:type=Node, *`.

Example 36–1 *Including an Attribute Obtained from a Query Pattern*

```
<column id = "RoleName">
  <type>attribute</type>
  <name>RoleName</name>
  <header>Role Name</header>
</column>
```

How to Include Part of the Key

A value that is present in an `ObjectName` key can be obtained from the `ObjectNames` returned from the query-pattern. This value can subsequently be included in the report.

[Example 36–2](#) illustrates how to include the `nodeId` key part from the query pattern `Coherence:type=Node,*`.

Example 36–2 Including Part of an `ObjectName` Key in a Report

```
<column id="NodeId">
  <type>key</type>
  <name>nodeId</name>
  <header>Node Id</header>
</column>
```

How to Include Information from Composite Attributes

JMX composite values can be used to include part of a composite data attribute in a report.

[Example 36–9](#) illustrates how to include the `startTime` of the `LastGCInfo` attribute from the query pattern `java.lang:type=GarbageCollector,*`.

Example 36–3 Including Information from a Composite Attribute in a Report

```
<column id="LastGCStart">
  <type>attribute</type>
  <name>LastGcInfo/startTime</name>
  <header>Last GC Start Time</header>
</column>
```

How to Include Information from Multiple MBeans

A JMX join attribute is required when a report requires information from multiple MBeans. The major considerations when creating a join is to determine both the primary query, the join query and the foreign key. The primary query should be the query that returns the appropriate number of rows for the report. The join query pattern must reference a single MBean and can not contain a wild card (*). The foreign key is determined by what attributes from the primary query that are required to complete the join query string.

The reporter feature that enables joins between MBeans is a column substitution macro. The column substitution allows for the resulting value from a column to be included as part of a string. A column substitution macro is a column ID attribute surrounded by curly braces "{ }". The reporter does not check for cyclical references and will fail during execution if a cycle is configured.

Including Multiple MBean Information Example

You can draw information from more than one MBean and include it in a report. This requires a join between the MBeans.

Note: The major limitation of `join` attributes is that the result of the join must have only one value.

For example, if a report requires the `TotalGets` from the Cache MBean (`Coherence:type=cache, *`) and `RoleName` from the Node MBean (`Coherence:type=Node, *`), then a join attribute must be used.

Since a greater number of MBeans will come from the Cache MBean, `Coherence:type=Cache, *` would be the primary query and the `RoleName` would be the join attribute. The foreign key for this join is the `nodeId` key part from the Cache MBean and it must be included in the report. The configuration for this scenario is illustrated in [Example 36–4](#).

Example 36–4 Including Information from Multiple MBeans in a Report

```
<column id="RoleName">
  <type>attribute</type>
  <name>RoleName</name>
  <header>Role Name</header>
  <query>
    <pattern>Coherence:type=Node,nodeId={NodeFK}</pattern>
  </query>
</column>

<column id="NodeFK">
  <type>key</type>
  <name>nodeId</name>
  <header>Node Id</header>
</column>
```

How to Use Report Macros

There are three report macros that can be included in a report:

- **Report Time** (`report-time`)—is the time and date that the report was executed. This information is useful for time series analysis.
- **Report Batch/Count** (`report-count`)—is a long identifier that can be used to correlate information from different reports executed at the same time.
- **Reporting Node** (`report-node`)—is used when integrating information from the same report executed on different nodes or excluding the executing node information from the report.

To include the execution time into the report:

Example 36–5 Including Execution Time in a Report

```
<column id="ReportTime">
  <type>global</type>
  <name>{report-time}</name>
  <header>Report Time</header>
</column>
```

To include the Report Batch/Count:

Example 36–6 Including the Report Batch/Count in a Report

```
<column id="ReportBatch">
  <type>global</type>
  <name>{report-count}</name>
  <header>batch</header>
</column>
```


To include the execution node:

Example 36–7 Including the Execution Node

```
<column id="ReportNode">
  <type>global</type>
  <name>{report-node}</name>
  <header>ExecNode</header>
  <hidden>true</hidden>
</column>
```

How to Include Constant Values

Report constants can be used to either static values or report parameters. These constants can be either double or string values. Often, these are used in filters to limit the results to a particular data set or in calculations.

[Example 36–8](#) illustrates how to include a constant double of 1.0 in a report:

Example 36–8 Including a Constant Numeric Value in a Report

```
<column id="One">
  <type>constant</type>
  <header>Constant1</header>
  <data-type>double</data-type>
  <value>1.0</value>
  <hidden>true</hidden>
</column>
```

[Example 36–9](#) illustrates how to include the constant string `dist-Employee` in a report:

Example 36–9 Including a Constant String in a Report

```
<column id="EmployeeCacheName">
  <type>constant</type>
  <header>Employee Cache Name</header>
  <data-type>string</data-type>
  <value>dist-Employee</value>
  <hidden>true</hidden>
</column>
```

Including Queries in a Report

The query is the foundation of the information included in a report. Each query includes a query pattern, column references, and an optional filter reference. The query pattern is a string that is a JMX `ObjectName` query string. This string can return one or more MBeans. The column references must be defined in the `<columns>` section of the report definition file. The filter reference must be defined in the `<filters>` section of the report section.

[Example 36–10](#) illustrates how to include the list all the Node IDs and RoleNames in the cluster where the RoleName equals `CoherenceServer`.

Example 36–10 Including a List of the Cluster's NodeIDs and RoleNames in a Report

```
<filters>
  <filter id="equalsRef">
    <type>equals</type>
    <params>
```

```
        <column-ref>RoleRef</column-ref>
        <column-ref>StringRef</column-ref>
    </params>
</filter>
</filters>

<query>
    <pattern>Coherence:type=Node,*</pattern>
    <filter-ref>equalsRef</filter-ref>
</query>

<row>
    <column id ="NodeRef">
        <type>key</type>
        <name>nodeId</name>
        <header>Node Id</header>
    </column>

    <column id ="RoleRef">
        <name>RoleName</name>
        <header>Role</header>
    </column>

    <column id = "StringRef">
        <type>constant</type>
        <name>ConstString</name>
        <data-type>string</data-type>
        <value>CoherenceServer</value>
        <hidden>true</hidden>
    </column>

</row>
```

Using Filters to Construct Reports

Filters limit the data returned in the Report. Filters are either comparison filters or composite filters. Comparison Filters evaluate the results of two columns while composite filters evaluate the boolean results from one or two filters. Comparison filters are `equals`, `greater`, and `less`.

Composite Filter types are `and`, `or`, and `not`. Each composite filter evaluates the filter parameters first to last and apply standard boolean logic. Composite filter evaluation uses standard short circuit logic. Cyclic references checks are not performed during execution. If a cyclic reference occurs, it will create a runtime error.

[Example 36–11](#) illustrates how to define an `equals` filter where `RoleRef` and `StringRef` are defined columns.

Example 36–11 Using an Equals Filter for a Report

```
<filters>
    <filter id="equals">
        <type>equals</type>
        <params>
            <column-ref>RoleRef</column-ref>
            <column-ref>StringRef</column-ref>
        </params>
    </filter>
</filters>
```

[Example 36–12](#) illustrates how to define a filter where the number of PacketsResent are greater than PacketsSent (assuming PacketsResent and PacketsSent are valid column references).

Example 36–12 Defining a "Greater Than" Filter for a Report

```
<filters>
  <filter id="greaterRef">
    <type>greater</type>
    <params>
      <column-ref>PacketsResent</column-ref>
      <column-ref>PacketsSent</column-ref>
    </params>
  </filter>
</filters>
```

[Example 36–13](#) illustrates how to define an filter where the number of PacketsResent are less than PacketsSent (assuming PacketsResent and PacketsSent are valid column references).

Example 36–13 Defining a "Less Than" Filter for a Report

```
<filters>
  <filter id="greaterRef">
    <type>less</type>
    <params>
      <column-ref>PacketsResent</column-ref>
      <column-ref>PacketsSent</column-ref>
    </params>
  </filter>
</filters>
```

[Example 36–14](#) illustrates how to define an and filter (assuming all column-ref values are valid).

Example 36–14 Defining an "And" Filter for a Report

```
<filters>
  <filter id="equalsRef">
    <type>equals</type>
    <params>
      <column-ref>RoleRef</column-ref>
      <column-ref>StringRef</column-ref>
    </params>
  </filter>

  <filter id="greaterRef">
    <type>greater</type>
    <params>
      <column-ref>PacketsResent</column-ref>
      <column-ref>PacketsSent</column-ref>
    </params>
  </filter>

  <filter>
    <type>and</type>
    <params>
      <filter-ref>greaterRef</filter-ref>
      <filter-ref>equalsRef</filter-ref>
    </params>
  </filter>
```

```
    </filter>
</filters>
```

[Example 36–15](#) illustrates how to define an or filter (assuming all column-ref values are valid).

Example 36–15 Defining an "Or" Filter for a Report

```
<filters>
  <filter id="equalsRef">
    <type>equals</type>
    <params>
      <column-ref>RoleRef</column-ref>
      <column-ref>StringRef</column-ref>
    </params>
  </filter>

  <filter id="greaterRef">
    <type>greater</type>
    <params>
      <column-ref>PacketsResent</column-ref>
      <column-ref>PacketsSent</column-ref>
    </params>
  </filter>

  <filter>
    <type>or</type>
    <params>
      <filter-ref>greaterRef</filter-ref>
      <filter-ref>equalsRef</filter-ref>
    </params>
  </filter>
</filters>
```

[Example 36–16](#) illustrates how to define a not equals filter, where RoleRef and StringRef are defined columns.

Example 36–16 Defining a "Not Equals" Filter for a Report

```
<filters>
  <filter id="equals">
    <type>equals</type>
    <params>
      <column-ref>RoleRef</column-ref>
      <column-ref>StringRef</column-ref>
    </params>
  </filter>

  <filter id = "Not">
    <type>not</type>
    <params>
      <filter-ref>equals</filter-ref>
    </params>
  </filter>
</filters>
```

Using Functions to Construct a Report

Reporter functions allow mathematical calculations to be performed on data elements within the same row of the report. The supported functions are *Add*, *Subtract*, *Multiply*, and *Divide*. Function columns can then be included as parameters into other function columns.

Function Examples

[Example 36–17](#) illustrates how to add columns (*Attribute1* and *Attribute2*) and place the results into a third column (*Addition*).

Example 36–17 Adding Column Values and Including Results in a Different Column

```
<column id="AttributeID1">
  <name>Attribute1</name>
</column>

<column id="AttributeID2">
  <name>Attribute2</name>
</column>

<column id="Addition">
  <type>function</type>
  <name>Add2Columns</name>
  <header>Adding Columns</header>
  <function-name>add</function-name>
  <params>
    <column-ref>AttributeID1</column-ref>
    <column-ref>AttributeID2</column-ref>
  </params>
</column>
```

[Example 36–18](#) illustrates how to subtract one column value (*Attribute2*) from another (*Attribute1*) and place the results into a third column (*Subtraction*).

Example 36–18 Subtracting Column Values and Including Results in a Different Column

```
<column id="AttributeID1">
  <name>Attribute1</name>
</column>

<column id="AttributeID2">
  <name>Attribute2</name>
</column>

<column id="Subtraction">
  <type>function</type>
  <name>Subtract2Columns</name>
  <header>Difference</header>
  <function-name>subtract</function-name>
  <params>

    <column-ref>AttributeID1</column-ref>
    <column-ref>AttributeID2</column-ref>
  </params>
</column>
```

[Example 36–19](#) illustrates how to multiply column values (*Attribute1* and *Attribute2*) place the results into a third column (*Multiplication*).

Example 36–19 Multiplying Column Values and Including Results in a Different Column

```
<column id="AttributeID1">
  <name>Attribute1</name>
</column>

<column id="AttributeID2">
  <name>Attribute2</name>
</column>

<column id="Multiplication">
  <type>function</type>
  <name>Multiply2Columns</name>
  <header>Multiply Columns</header>
  <function-name>multiply</function-name>
  <params>
    <column-ref>AttributeID1</column-ref>
    <column-ref>AttributeID2</column-ref>
  </params>
</column>
```

[Example 36–20](#) illustrates how to divide one column (Attribute1) by another (Attribute2) into a third column (Division). The result of all division is a Double data type.

Example 36–20 Dividing Column Values and Including Results in a Different Column

```
<column id="AttributeID1">
  <name>Attribute1</name>
</column>

<column id="AttributeID2">
  <name>Attribute2</name>
</column>

<column id="Division">
  <type>function</type>
  <name>Dividing2Columns</name>
  <header>Division</header>
  <function-name>Divide</function-name>
  <params>
    <column-ref>AttributeID1</column-ref>
    <column-ref>AttributeID2</column-ref>
  </params>
</column>
```

Using Aggregates to Construct a Report

Reporter aggregates allow for multiple rows to be aggregated into a single value or row. [Table 36–3](#) describes the available aggregate types.

Table 36–3 Reporter Aggregate Types

Type	Description
avg	Calculate the mean value for all values in the column.
max	Return the maximum value for all values in the column.
min	Return the minimum value for all values in the column.
sum	Add all the values from a column.

Aggregate Examples

Sum the values in the `size` column

Example 36–21 Adding the Values in a Column

```
<column id ="SumRef">
  <type>function</type>
  <function-name>sum</function-name>
  <column-ref>size</column-ref>>
  <header>Sum</header>
</column>
```

Average the values in the `size` column

Example 36–22 Calculating the Average of Values in a Column

```
<column id ="AverageRef">
  <type>function</type>
  <header>Average</header>
  <function-name>avg</function-name>
  <column-ref>size</column-ref>>
</column>
```

Find the maximum the value in the `size` column

Example 36–23 Finding the Maximum Value in a Column

```
<column id ="MaximumRef">
  <type>function</type>
  <header>Maximum</header>
  <function-name>max</function-name>
  <column-ref>size</column-ref>>
</column>
```

Find the minimum the value in the `size` column

Example 36–24 Finding the Minimum Value in a Column

```
<column id ="MinimumRef">
  <type>function</type>
  <header>Minimum</header>
  <function-name>min</function-name>
  <column-ref>size</column-ref>>
</column>
```

Constructing Delta Functions

Many numeric attributes in the Coherence report are cumulative. These values are reset only when the `resetStatistics` operation is executed on the MBean. To determine the state of the system without resetting the statistics, the Reporter uses a delta function. The delta function subtracts the prior value of a column from the current value of a column and returns the difference.

The prior values for a report are stored in a map on the Reporter client. This map is keyed by the "delta key". By default, the delta key is the MBean name for the attribute. However, when one-to-one relationship does not exist between the MBean and the rows in the report, or the MBean name is subject to change between executions of the report, the delta key will be calculated using the columns provided in the `<params>` section.

Note: Accuracy of Delta Functions: delta functions are only correct when the report is running as part of a report batch.

Delta Function Examples

[Example 36–25](#) illustrates how to include a delta calculation of an attribute. (Assume `PacketsSent` is a defined column)

Example 36–25 Delta Calculation for an Attribute

```
<column id="DeltaPacketsSent">
  <type>function</type>
  <name>PacketsSent</name>
  <header>Delta Sent</header>
  <function-name>delta</function-name>
  <column-ref>PacketsSent</column-ref>
</column>
```

[Example 36–26](#) illustrates how to include a delta calculation of an attribute with an alternate delta key. (Assume `PacketsSent`, `NodeID` and `TimeStamp` are defined columns)

Example 36–26 Delta Calculation for an Attribute with an Alternate Delta Key

```
<column id="DeltaPacketsSent">
  <type>function</type>
  <name>PacketsSent</name>
  <header>Delta Sent</header>
  <function-name>delta</function-name>
  <column-ref>PacketsSent</column-ref>
  <params>
    <column-ref>NodeID</column-ref>
    <column-ref>TimeStamp</column-ref>
  </params>
</column>
```

How to Modify Report Batch

Configuring a report batch is one of the steps in creating a custom report. You typically configure it after creating report configuration files. This configuration file determines what reports the reporter executes, how often the reports get executed, and where the reports are saved. If a single report can be used with different parameters, these parameters are also configured in the report batch. For more information on report configuration files, see [Chapter 36, "How to Create a Custom Report"](#).

Report Batch Deployment Descriptor

Use the report batch deployment descriptor to specify the various options for creating custom reports.

Document Location

The name and location of the descriptor defaults to `report-group.xml`. The default descriptor (packaged in `coherence.jar`) will be used unless a custom file is found in the application's classpath.

Document Root

The root element of the POF user type descriptor is `report-group`. This is where you may begin specifying the format of the custom report.

System Properties

[Table 37-1](#) describes the system properties that can be used to control report batch from the command line.

Table 37-1 System Properties for Controlling Report Batch

Property	Default	Description
<code>tangosol.coherence.management.report.configuration</code>	<code>reports/report-group.xml</code>	The XML file containing the Reporter configuration settings, such as the list of reports, the report frequency, and so on.
<code>tangosol.coherence.management.report.autostart</code>	<code>false</code>	Flag to automatically start the reporter when the node is started.
<code>tangosol.coherence.management.report.distributed</code>	<code>false</code>	Determines if the reporter is running in a central model (<code>false</code>) or on every node in the cluster (<code>true</code>).

Document Format

The report batch descriptor should begin with the following DOCTYPE declaration:

```
<!DOCTYPE report-group SYSTEM "report-group.dtd">
```

[Example 37–1](#) illustrates the nesting of elements in a report batch document.

Example 37–1 Format of a Report Batch Configuration File (report-group.xml)

```
<report group>
  <frequency/>
  <output-directory/>
  <report-list>
    <location/>
  <report-config>
    <init-params>
      <init-param>
        </init-param>
    </report-config>
</report-group>
```

Report Batch Element Index

Table 37-2 describes the relationship between the report batch elements.

Table 37-2 Report Batch Elements

Element	Used in:
frequency	report-group
location	report-list
init-param	init-params
init-params	report-config
output-directory	report-group
param-name	init-param
param-type	init-param
param-value	init-param
report-config	report-group
report-group	root element
report-list	report-group

frequency

Used in: [report-group](#)

Description

Required. A string containing the number of seconds, minutes between each execution of the report batch. 10s will run the report every 10 seconds. 5m will run the report every 5 minutes. Selecting an appropriate frequency is critical. If the frequency is too short, the reporter can generate a large amount of data and consume significant disk space. If the frequency is too long, the information will not be useful. It is recommended that a process for purging and archiving historical information is in place before configuring the reporter.

location

Used in: [report-list](#)

Description

Required. The path to the report configuration file. For more information on this file, see [Chapter 36, "How to Create a Custom Report"](#).

init-param

Used in: [init-params](#)

Description

The `init-param` element contains an initialization parameter for a report. The parameter consists of either a parameter name or type, and its value.

init-params

Used in: [report-config](#)

Description

Optional. The `init-params` element contains a list of initialization parameters.

output-directory

Used in: [report-group](#)

Description

Optional. The directory path to prepend to the output file names from the report configuration files. The username which the node is executing **must** have read write access to this path.

param-name

Used in: [init-param](#)

Description

The `param-name` element specifies the name of the initialization parameter.

param-type

Used in: [init-param](#)

Description

The `param-type` element specifies the Java type of the initialization parameter. Supported types are:

- `string`—indicates that the value is a `java.lang.String`
- `long`—indicates that the value is a `java.lang.Long`
- `double`—indicates that the value is a `java.lang.Double`

param-value

Used in: [init-param](#)

Description

The `param-value` element specifies a value of the initialization parameter. The value is in a format specific to the type of the parameter.

report-config

Used in: [report-group](#)

Description

The `report-config` contains the configuration file name and the initialization parameters for the report.

report-group

Used in: *root element*

Description

Describes the report list, the frequency, the report parameters, and the output directory for the batch.

report-list

Used in: [report-group](#)

Description

Required. The list of reports to include in the batch. This element contains the `<report-config>` subelement.

Analyzing Reporter Content

Coherence provides out-of-box reports that helps administrators and developers better analyze usage and configuration issues that may occur. For reports specific to Coherence*Web, see "Running Performance Reports" in *Oracle Coherence User's Guide for Oracle Coherence*Web*.

The following sections are included in this chapter:

- [Network Health Report](#)
- [Network Health Detail Report](#)
- [Memory Status Report](#)
- [Cache Size Report](#)
- [Cache Usage Report](#)
- [Service Report](#)
- [Node List Report](#)
- [Proxy Report](#)

Network Health Report

The Network Health report contains the primary aggregates for determining the health of the network communications. The network health file is a tab delimited file that is prefixed with the date and hour in YYYYMMDDHH format and post fixed with `-network-health.txt`. For example `2009013113-network-health.txt` would be created on January 31, 2009 at 1:00 PM. [Table 38-1](#) describes the content of the Network Health report.

Table 38-1 Contents of the Network Health Report

Column	Type	Description
Batch Counter	Long	A sequential counter to help integrate information between related files. This value does reset when the reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
Report Time	Date	The system time when the report executed.
Min Node Rx Success	Double	The minimum receiver success rate for a node in the cluster. If this value is considerably less (10%) than the Grid Rx Success rate. Further analysis using the Network Health Detail should be done.

Table 38–1 (Cont.) Contents of the Network Health Report

Column	Type	Description
Grid Rx Success	Double	The receiver success rate for the grid as a whole. If this value is below 90%. Further analysis of the network health detail should be done.
Min Node Tx Success	Double	The minimum publisher success rate for a node in the cluster. If this value is considerably less (10%) than the Grid Rx Success rate. Further analysis using the Network Health Detail should be done.
Grid TX Success	Double	The publisher success rate for the grid as a whole. If this value is below 90%. Further analysis of the network health detail should be done.

Network Health Detail Report

The Network Health report supporting node level details for determining the health of the network communications. The network health detail file is a tab delimited file that is prefixed with the date and hour in YYYYMMDDHH format and post fixed with `-network-health-detail.txt`. For example `2009013114-network-health.txt` would be created on January 31, 2009 at 2:00PM. [Table 38–2](#) describes the content of the Network Health Detail report.

Table 38–2 Contents of the Network Health Detail Report

Column	Data Type	Description
Batch Counter	Long	A sequential counter to help integrate information between related files. This value does reset when the reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
Report Time	Date	The system time when the report executed.
Node Id	Long	The node for the network statistics.
Tx Success	Double	The publisher success rate for the node. If this value is within 2%-3% of the "Min Node Tx Success" and more than 10% less than the "Grid Tx Success" for the batch in the Network Health File, the corresponding node may be having difficulty communicating with the cluster. Constrained CPU, constrained network bandwidth or high network latency could cause this to occur.
RX Success	Double	The receiver success rate for the node. If this value is within 2%-3% of the "Min Node Rx Success" and more than 10% less than the "Grid Tx Success" for the batch in the Network Health File, the corresponding node may be having difficulty communicating with the cluster. Constrained CPU, constrained network bandwidth or high network latency could cause this to occur.
Packets Sent	Double	The total number of network packets sent by the node.
Current Packets Sent	Long	The number of packets sent by the node since the prior execution of the report.

Table 38–2 (Cont.) Contents of the Network Health Detail Report

Column	Data Type	Description
Packets Resent	Long	The total number of network packets resent by the node. Packets will be resent when the receiver of the packet receives an invalid packet or when an acknowledge packet is not sent within the appropriate amount of time.
Current Packet Resent	Long	The number of network packets resent by the node since the prior execution of the report.
PacketsRepeated	Long	The total number of packets received more than once.
Current Packets Repeated	Long	The number of packets received since the last execution of the report.
Packets Received	Long	The total number of packets received by the node.
Current Packets Received	Long	The total number of packets received by the node since the last execution of the report.

Memory Status Report

The Memory Status report must be run as part of a report batch. The values are helpful in understanding memory consumption on each node and across the grid. For data to be included nodes must be configured to publish platform MBean information. The memory status file is a tab delimited file that is prefixed with the date and hour in YYYYMMDDHH format and post fixed with `-memory-status.txt`. For example `2009013115-memory-status.txt` would be created on January 31, 2009 at 3:00 PM. [Table 38–3](#) describes the content of the Memory Status report.

Table 38–3 Contents of the Memory Status Report

Column	Data Type	Description
Batch Counter	Long	A sequential counter to help integrate information between related files. This value does reset when the reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
Report Time	Date	The system time when the report executed.
Node Id	Long	The node for the memory statistics.
Gc Name	String	The name of the Garbage Collector information.
CollectionCount	Long	The number of garbage collections that have happened since the virtual machine started.
Delta Collection Count	Long	The number of garbage collections that have occurred since the last execution of the report.
CollectTime	Long	The number of milliseconds the JVM has spent on garbage collection since the start of the JVM.
Delta Collect Time	Long	The number of milliseconds the JVM has spent on garbage collection since the last execution of the report.
Last GC Start Time	Long	The start time of the last Garbage Collection.
Last GC Stop Time	Long	The stop time of the last garbage collection.
Heap Committed	Long	The number of heap bytes committed at the time of report.

Table 38–3 (Cont.) Contents of the Memory Status Report

Column	Data Type	Description
Heap Init	Long	The number of heap bytes initialized at the time of the report.
Heap Max	Long	The Maximum number of bytes used by the JVM since the start of the JVM.
Heap Used	Long	The bytes used by the JVM at the time of the report.

Cache Size Report

The cache size report can be executed either on demand or it can be added as part of the report batch and the Caches should have the `<unit-calculator>` subelement of `<local-scheme>` set to `BINARY`. The cache size file is a tab delimited file that is prefixed with the date and hour in `YYYYMMDDHH` format and post fixed with `-cache-size.txt`. For example `2009013101-cache-size.txt` would be created on January 31, 2009 at 1:00 AM. [Table 38–4](#) describes the content of the Cache Size report.

Table 38–4 Contents of the Cache Size Report

Column	Data Type	Description
Batch Counter	Long	A sequential counter to help integrate information between related files. This value does reset when the reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
Report Time	Date	The system time when the report executed.
Cache Name	String	The name of the cache.
MemoryMB	Double	The MB consumed by the objects in the cache. This does not include indexes or over head.
Avg Object Size	Double	The Average memory consumed by each object.
Cache Size	Double	The number of objects in the cache.
Memory Bytes	Double	The number of bytes consumed by the objects in the cache. This does not include indexes or over head.

Cache Usage Report

The cache usage report provides information about cache usage (gets, puts, evictions, and so on).

The report is a tab-delimited file that is prefixed with the date and hour in `YYYYMMDDHH` format and post fixed with `-cache-usage.txt`. For example, `2010013113-cache-usage.txt` would be created on January 31, 2010 1:00 pm. [Table 38–5](#) describes the content of the cache utilization report.

Table 38–5 Contents of the Cache Usage Report

Column	Data Type	Description
Batch Counter	Long	A sequential counter to help integrate information between related files. This value resets when the Reporter restarts, and is not consistent across nodes. However, it is helpful when trying to integrate files.
Report Time	Date	The system time when the report is executed.

Table 38–5 (Cont.) Contents of the Cache Usage Report

Column	Data Type	Description
Service	String	The name of the cache service.
Cache Name	String	The name of the cache.
Tier	String	Value can be either <code>front</code> or <code>back</code> . Describes whether the cache resides in the front-tier (local cache) or back tier (remote cache).
Total Puts	Double	The total number of puts for the cache across the cluster since the last time the report was executed.
Total Puts Milliseconds	Double	The total number of milliseconds spent per <code>put()</code> invocation (<code>PutsMillis</code>) across the cluster since the last time the report was executed.
Total Gets	Double	The total number of gets for the cache across the cluster since the last time the report was executed.
Total Gets Milliseconds	Double	The total number of milliseconds spent per <code>get()</code> invocation (<code>GetsMillis</code>) across the cluster since the last time the report was executed.
Total Hits	Double	The total number of hits for the cache across the cluster since the last time the report was executed.
Total Hits Milliseconds	Double	The total number of milliseconds spent per <code>get()</code> invocation that is a hit (<code>HitsMillis</code>) across the cluster since the last time the report was executed.
Total Misses	Double	The total number of misses for the cache across the cluster since the last time the report was executed.
Total Misses Milliseconds	Double	The total number of milliseconds spent per <code>get()</code> invocation that is a miss (<code>MissesMillis</code>) across the cluster since the last time the report was executed.
Total Writes	Double	The total number of storage writes for the cache across the cluster since the last time the report was executed.
Total Writes Milliseconds	Double	The total number of milliseconds spent in storage write operations (<code>WritesMillis</code>) across the cluster since the last time the report was executed.
Total Reads	Double	The total number of reads from a cache store for the cache across the cluster since the last time the report was executed.
Total Read Milliseconds	Double	The total number milliseconds on cache store reads for the cache across the cluster since the last time the report was executed.
Total Failures	Long	The total number of storage failures for the cache across the cluster since the last time the report was executed.
Total Queue	Long	The sum of the queue link sizes across the cluster.
evictions	Long	The total number of evictions for the cache across the cluster since the last time the report was executed.
Cache Prunes	Long	The total number of prunes for the cache across the cluster since the last time the report was executed.
Cache Prunes Milliseconds	Long	The total number of milliseconds spent in the prune operation (<code>PrunesMillis</code>) across the cluster since the last time the report was executed.

Service Report

The service report provides information to the requests processed, request failures, and request backlog, tasks processed, task failures and task backlog. Request Count and Task Count are useful to determine performance and throughput of the service. RequestPendingCount and Task Backlog are useful in determining capacity issues or blocked processes. Task Hung Count, Task Timeout Count, Thread Abandoned Count, Request Timeout Count are the number of unsuccessful executions that have occurred in the system. [Table 38–6](#) describes the contents of the Service report.

Table 38–6 *Contents of the Service Report*

Column	Data Type	Description
Batch Counter	Long	A sequential counter to help integrate information between related files. This value does reset when the reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
Report Time	Date	The system time when the report executed.
Service	String	The service name.
Node Id	String	The numeric node identifier.
Refresh Time	Date	The system time when the service information was updated from a remote node.
Request Count	Long	The number of requests since the last report execution.
RequestPendingCount	Long	The number of pending requests at the time of the report.
RequestPendingDuration	Long	The duration for the pending requests at the time of the report.
Request Timeout Count	Long	The number of request timeouts since the last report execution.
Task Count	Long	The number of tasks executed since the last report execution.
Task Backlog	Long	The task backlog at the time of the report execution.
Task Timeout Count	Long	The number of task timeouts since the last report execution.
Task Hung Count	Long	The number of tasks that hung since the last report execution.
Thread Abandoned Count	Long	The number of threads abandoned since the last report execution.

Node List Report

Due to the transient nature of the node identifier (`nodeId`), the reporter logs out a list of nodes and the user defined `<member-identity>` information. The node list file is a tab delimited file that is prefixed with the date and hour in `YYYYMMDDHH` format and post fixed with `-nodes.txt`. For example `2009013101-nodes.txt` would be created on January 31, 2009 at 1:00 AM. [Table 38–7](#) describes the content of the Node List report.

Table 38–7 Contents of the Node List Report

Column	Data Type	Description
Batch Counter	Long	A sequential counter to help integrate information between related files. This value does reset when the reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
Report Time	Date	The system time when the report executed.
Node Id	String	The numeric node identifier.
Unicast Address	String	The Unicast address for the node.
Member Name	String	The member name for the node.
Process Name	String	The process name for the node.
Role Name	String	The role name for the node.
Machine Name	String	The machine name for the node.
Rack Name	String	The rack name for the node.
Site Name	String	The site name for the node.
Refresh Time	Date/Time	The time which the information was refreshed from a remote node. If the time is not the same as the refresh time on other rows in the batch, the node did not respond in a timely matter. This is often caused by a node performing a garbage collection. Any information regarding a node with an "old" refresh date is questionable.

Proxy Report

The proxy file provides information about proxy servers and the information being transferred to clients. The Proxy file is a tab delimited file that is prefixed with the date and hour in YYYYMMDDHH format and post fixed with `-report-proxy.txt`. For example `2009013101-report-proxy.txt` would be created on January 31, 2009 at 1:00 AM. [Table 38–8](#) describes the content of the Proxy report.

Table 38–8 Contents of the Proxy Report

Column	Type	Description
Batch Counter	Long	A sequential counter to help integrate information between related files. This value does reset when the reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
Report Time	Date	The system time when the report executed.
Node Id	String	The numeric node identifier.
Service Name	String	The name of the proxy service.
HostIp	String	The IP Address and Port of the proxy service.
Connection Count	Long	The current number of connections to the proxy service.
Outgoing Byte Backlog	Long	The number of bytes queued to be sent by the proxy service.
Outgoing Message Backlog	Long	The number of messages queued by the proxy service.

Table 38–8 (Cont.) Contents of the Proxy Report

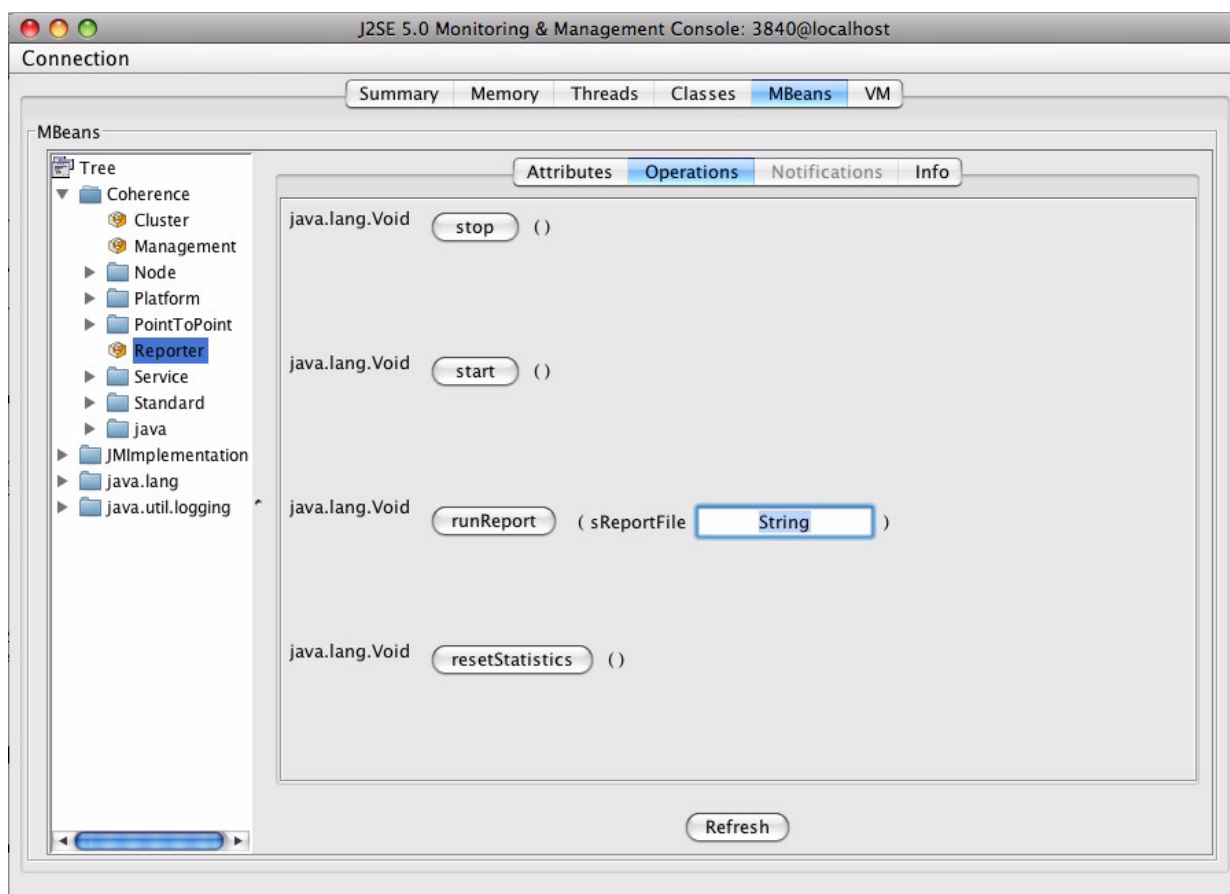
Column	Type	Description
Bytes Sent	Long	The number of bytes sent by the proxy service since the last execution of the report.
Bytes Received	Long	The number of bytes received by the proxy service since the last execution of the report.
Messages Sent	Long	The number of messages sent by the proxy service since the last execution of the report.
Messages Received	Long	The number of messages received by the proxy service since the last execution of the report.

How to Run a Report on Demand

A report can be run on demand by using either JConsole or the JMX HTTP Adapter. The Reporter MBean operations contain a `runReport(String sReportPath)` method. The report path can either be a resource in `coherence.jar` or a file URL.

Figure 39-1 illustrates the reporter operations in JConsole.

Figure 39-1 Reporter Operations in JConsole



How to Run ReportControl MBean at Node Startup

When set to true, the `tangosol.coherence.management.report.autostart` system property allows the ReportControl MBean to start execution when the node is started. This property must be used with the

`tangosol.coherence.management.report.group` system property and the configuration of the custom MBean XML file.

In [Example 39-1](#), the `tangosol.coherence.management.report.autostart` system property is set to `true`.

Example 39-1 `tangosol.coherence.management.report.autostart` System Property

```
-Dtangosol.coherence.management.report.autostart=true
```

How to Configure the ReportControl MBean

The report group system property, `tangosol.coherence.management.report.group` configures the ReportControl MBean with the specified configuration file. This property must be used in correlation with the `tangosol.coherence.management.report.autostart` property and the configuration of the custom MBean XML file.

In [Example 39-2](#), the `tangosol.coherence.management.report.group` property points to the custom MBean XML file `report-batch.xml`.

Example 39-2 `tangosol.coherence.management.report.group` System Property

```
-Dtangosol.coherence.management.report.group=./report-batch.xml
```

Configuring Custom MBeans

This chapter provides information on configuring standard, MX, and JMX MBeans.

Creating an MBean XML Configuration File

Custom MBeans are configured in an XML configuration file. The elements in the file describe the MBean type, MBean implementation, and the target MBean `ObjectName`. The current release of Coherence supports these types of custom MBeans.

- Standard MBeans
- MXBeans
- JMX MBeans

See [Appendix A, "Operational Configuration Elements"](#) for a complete descriptions of the elements used in this chapter.

Configuring Standard MBeans

The configuration in [Example 40–1](#) will create a `Coherence:type=Query,nodeId=<nodeId>` using the standard MBean `com.oracle.customMBeans.Query` class for the node. This example specifies an MBean class, an MBean name, and enables the MBean to be registered in the instance.

Example 40–1 Using an MBean to Create a Query Node

```
<mbeans>
  <mbean id="100">
    <mbean-class>com.oracle.customMBeans.Query</mbean-class>
    <mbean-name>type=Query</mbean-name>
    <enabled>true</enabled>
  </mbean>
</mbeans>
```

Configuring MXBeans

The configuration in [Example 40–2](#) will execute the standard Java method `getMemoryMXBean` in the `java.lang.management.ManagementFactory` class and use the result to create a `Coherence:type=java,SubSystem=Memory,nodeId=<nodeId>` for the node. The following example specifies an MBean factory, an accessor method name on the factory, an MBean name, and enables the MBean to be registered in the instance.

Example 40–2 Getting an MBean for the Memory System of a Java Virtual Machine

```
<mbeans>
  <mbean id="2">
    <mbean-factory>java.lang.management.ManagementFactory</mbean-factory>
    <mbean-accessor>getMemoryMXBean</mbean-accessor>
    <mbean-name>type=java,SubSystem=Memory</mbean-name>
    <enabled>true</enabled>
  </mbean>
</mbeans>
```

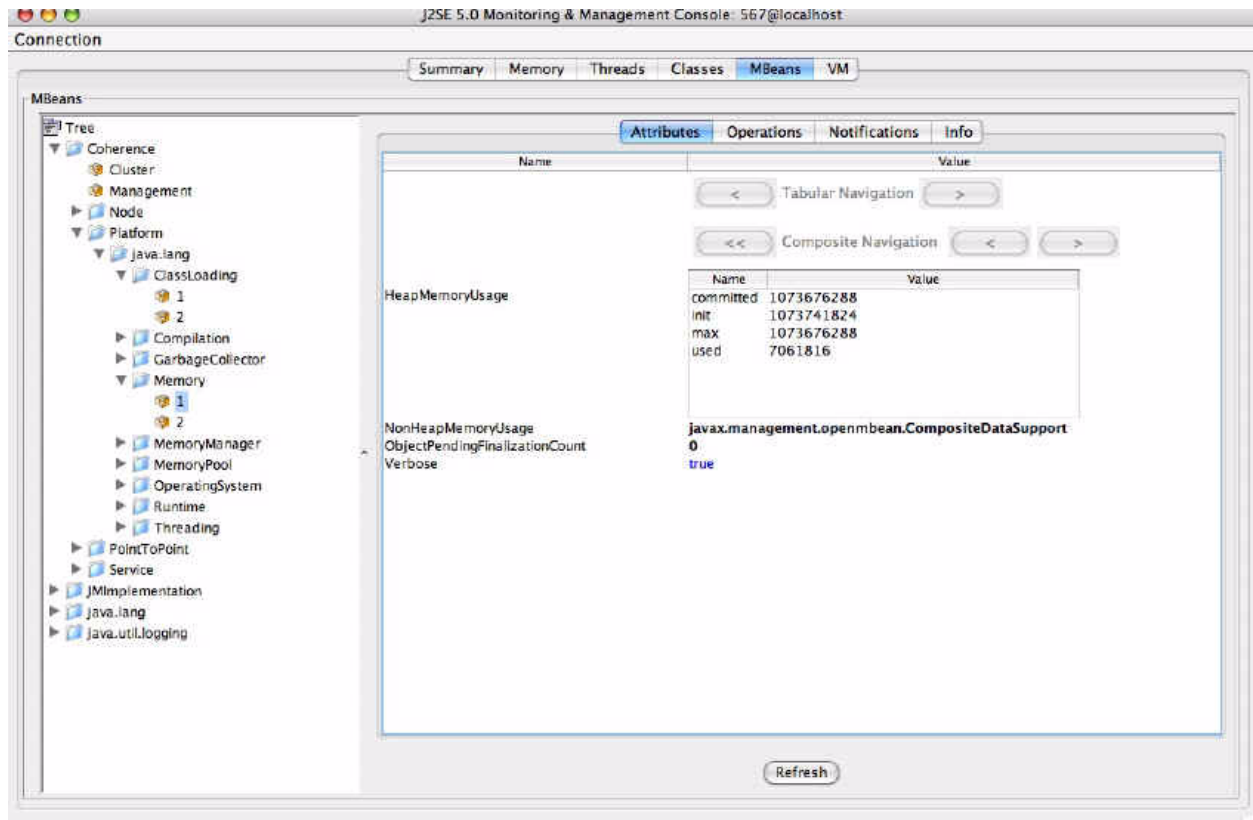
Configuring JMX MBeans

JMX MBeans are MBeans that exist in a local MBean server that need to be added to the Coherence Management structure. This allows consolidation of MBeanServer information into a single source. The configuration in [Example 40–3](#) executes the JMX query, `java.lang:*`, on the node's local MBean server and uses the results to create corresponding MBeans on the centralized Coherence MBean server. The following example specifies a JMX MBean query, an MBean name, and enables the MBean to be registered in the instance.

Example 40–3 Executing a JMX Query and Creating an MBean on the MBean Server

```
<mbeans>
  <mbean id="1">
    <mbean-query>java.lang:*</mbean-query>
    <mbean-name>type=Platform</mbean-name>
    <enabled>true</enabled>
  </mbean>
</mbeans>
```

[Figure 40–1](#) illustrates the results on the query in JConsole.

Figure 40–1 MBean Query Displayed in the JConsole

Enabling a Custom MBean Configuration File

You can enable the custom MBean configuration file by setting a system property or by including a specially named file in the class path.

Setting a System Property

Coherence provides the following system property to specify the name and location of a custom MBean configuration file. Setting this system property will cause the Coherence node to load the MBeans defined in the file represented by `filename`.

Example 40–4 System Property to Load an MBean

```
-Dtangosol.coherence.mbeans=<filename>
```

Adding a Custom MBean Configuration File to the Class Path

By convention, Coherence recognizes the configuration file named `custom-mbeans.xml` as containing a custom MBean configuration. If you name your custom MBean configuration file `custom-mbeans.xml` and include it in the class path, then the Coherence node will load the configured MBeans.

How to Manage Custom MBeans Within the Cluster

In addition to managing Coherence with JMX, Coherence provides the ability to manage and monitor custom MBeans (that is, application-level MBeans) within the Coherence JMX Management and Monitoring framework. This enables you to manage or monitor any application-level MBean from any JVM, node, or end-point within the cluster.

In addition to the standard Coherence managed object types, any dynamic or standard MBean type may be registered using the `com.tangosol.net.management.Registry` interface.

Custom MBean Configuration

Coherence can be configured to load platform and standard MBeans on connection to the cluster. This allows administrators and support personnel to update and view system and application information from all nodes in a cluster from a single location. This feature also eliminates the need for JMX programs to connect to multiple sources to gather information.

How to Add a Standard MBean to Coherence

The following instructions describe how to add a standard MBean to Coherence:

1. Create a standard MBean.
2. Add a standard MBean Class or JAR to the Coherence classpath (including central management node).
3. Create a custom MBean XML configuration file (see ["Creating an MBean XML Configuration File"](#) on page 40-1).
4. Modify node startup scripts to reference `custom-mbean.xml` (see ["Enabling a Custom MBean Configuration File"](#) on page 40-3).

How to Programmatically Add a Standard MBean to Coherence

[Example 41-1](#) illustrates sample code that programmatically adds a standard MBean to Coherence.

Example 41-1 Adding a Standard MBean to Coherence Programmatically

```
Registry    registry = CacheFactory.ensureCluster().getManagement();
Custom      bean     = new Custom();
String      sName    = registry.ensureGlobalName("type=Custom");
```

```
registry.register(sName, bean);
```

Using Static MBean Names

[Example 41-1](#) uses the `ensureGlobalName` method when adding a custom MBean to Coherence. The method is used to add the `nodeId=...` portion to the end of the MBean's `ObjectName`. This is required in order to have unique names on the centralized MBean server. If an application requires static MBean names, use a MBean query to add MBeans from a local MBean server to the Coherence management system. This will create the MBeans on the managed node with the static name and then add the `,nodeId=...` portion to the name of the MBean when aggregating on the managing node.

To use static MBean names:

1. Register the MBeans on the local MBean server of the managed node using the `registerMBean` or `createMBean` methods prior to joining the cluster. See the `MBeanServer` interface JavaDocs for information on using these methods:
<http://java.sun.com/j2se/1.5.0/docs/api/javax/management/MBeanServer.html>
2. Use the `MBeanHelper.findMBeanServer()` method to obtain the same MBean server that Coherence is using.
3. Configure the `custom-mbeans.xml` file to query the newly registered MBeans. See ["Configuring JMX MBeans"](#) on page 40-2.

Note: Refer to the Sun documentation below to ensure an environment is set up properly to access the local MBean server.

<http://java.sun.com/j2se/1.5.0/docs/guide/management/agent.html>

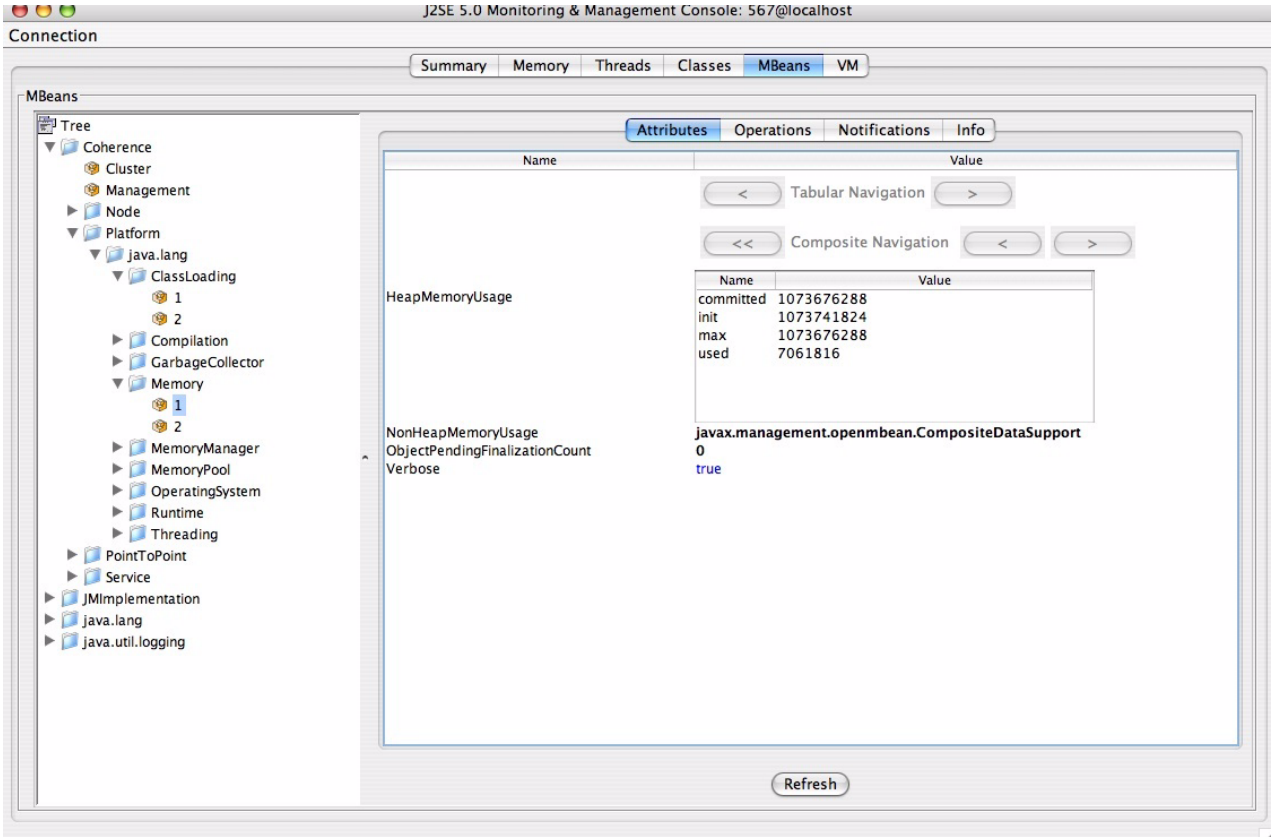
How to Add a the Results of a JMX Query to Coherence

The following instructions describe how to add the results of a JMX query to Coherence.

1. Create a custom MBean XML file (see ["Creating an MBean XML Configuration File"](#) on page 40-1).
2. Configure node startup script to include JMX MBean Server
3. Configure a node startup script to reference `custom-mbean.xml` (see ["Enabling a Custom MBean Configuration File"](#) on page 40-3).

[Figure 41-1](#) illustrates an example of running a JMX Query in JConsole.

Figure 41–1 JMX Query Run in JConsole



Part VII

Tuning Coherence

Part VII contains the following chapters:

- [Chapter 42, "Evaluating Performance and Scalability"](#)
- [Chapter 43, "Performing a Multicast Connectivity Test"](#)
- [Chapter 44, "Performing a Datagram Test for Network Performance"](#)
- [Chapter 45, "Performance Tuning"](#)
- [Chapter 46, "Using the Service Guardian"](#)
- [Chapter 47, "Using Quorum"](#)
- [Chapter 48, "Scaling Out Your Data Grid Aggregations Linearly"](#)

Evaluating Performance and Scalability

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

Measuring Latency and Throughput

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

Demonstrating Scalability

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

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

Tuning Your Environment

To get the most out of your cluster it is important that you've tuned of your environment and JVMs. [Chapter 45, "Performance Tuning,"](#) provides good start to getting the most out of your environment. For example, Coherence includes a registry script for Windows (`optimize.reg`), which will adjust a few critical settings and allow Windows to achieve much higher data rates.

Measurements on a Large Cluster

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

Figure 42–1 Coherence Throughput versus Number of Machines

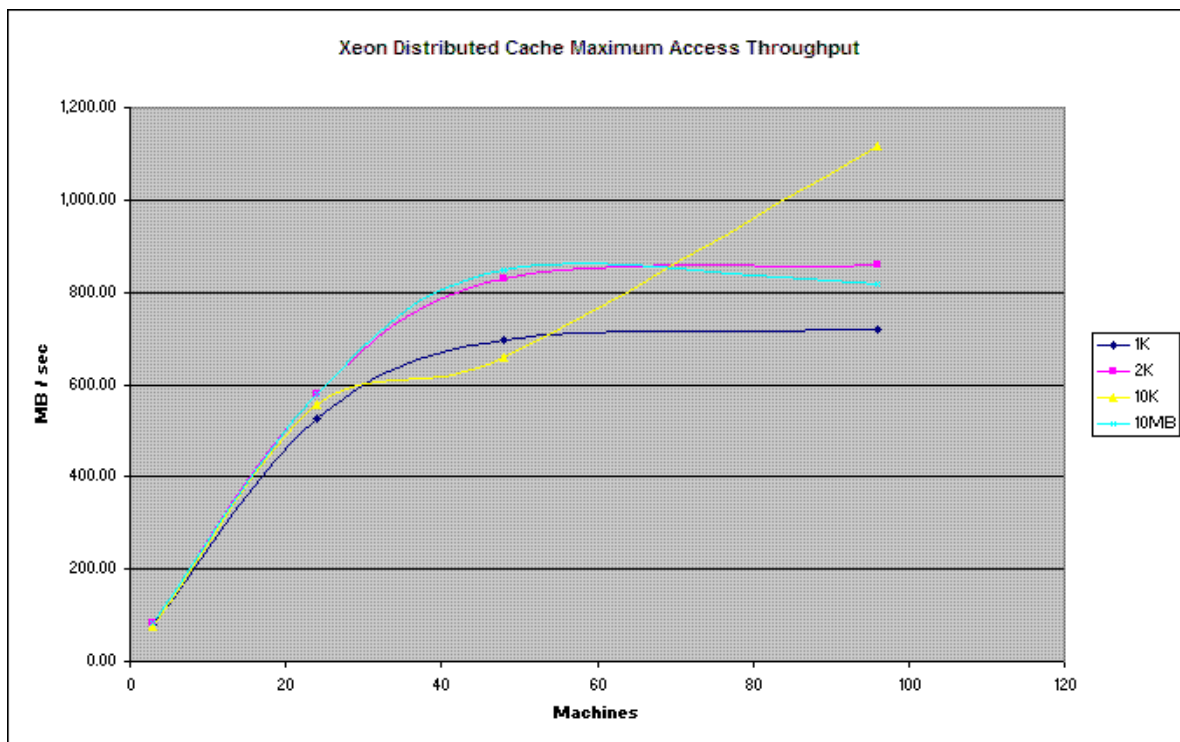
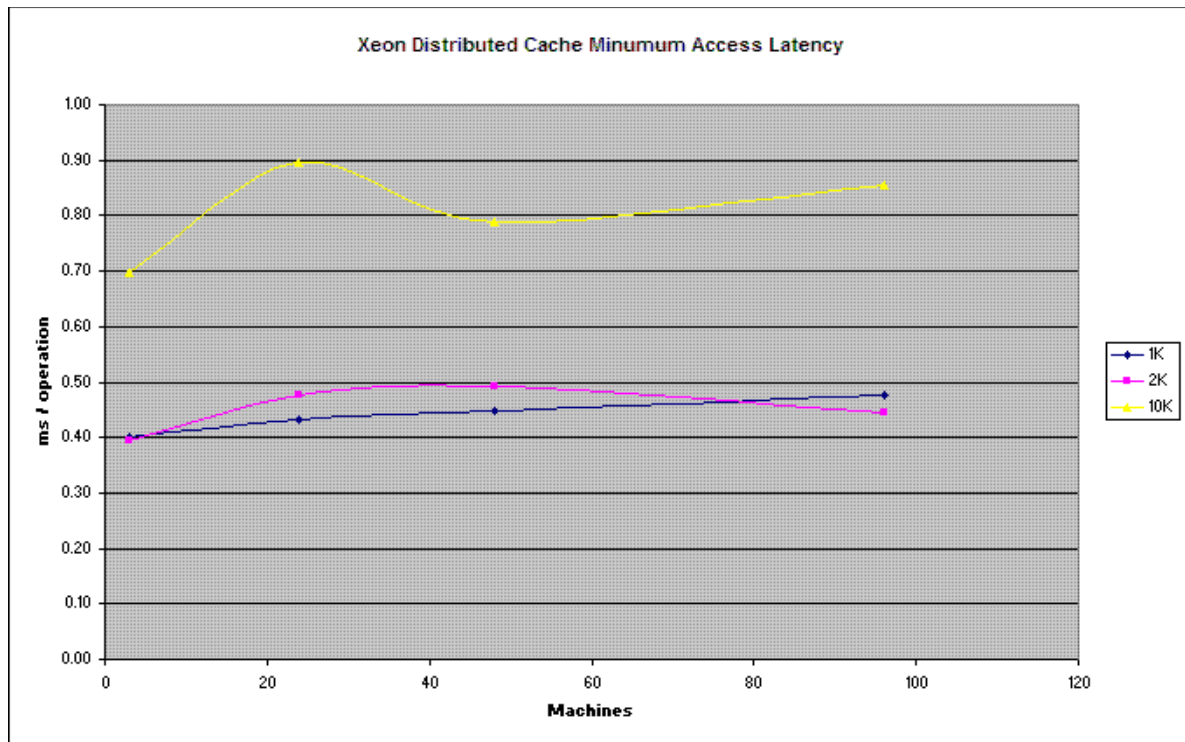


Figure 42-2 Coherence Latency versus Number of Machines

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

Performing a Multicast Connectivity Test

Included with Coherence is a Multicast Test utility, which helps you determine if multicast is enabled between two or more computers. This is a connectivity test, not a load test, each instance will by default only transmit a single multicast packet once every two seconds. For network load testing, see [Chapter 44, "Performing a Datagram Test for Network Performance."](#)

Running the Multicast Test Utility

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

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

[Table 43–1](#) describes the available command line options for the Multicast Test utility.

Table 43–1 *Command Line Options for the Multicast Test Utility*

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

Sample Commands

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

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

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

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

Multicast Test Example

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

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

From a command prompt, enter the following command:

Example 43–1 Command to Determine a Multicast Address

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

After pressing ENTER, you should see the Multicast Test utility display how it is sending sequential multicast packets and receiving them. [Example 43–2](#) illustrates sample output.

Example 43–2 Sequential Multicast Packets Sent by the Multicast Test Utility

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

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

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

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

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

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

```
multicast-test.sh
```

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

Example 43–3 Sample Multicast Test Results from Server A

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



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

and something like the following on Server B:

Example 43-4 Sample Multicast Test Results on Server B

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

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

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

Troubleshooting Multicast Communications

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

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

- Received ???—If the test reports receiving "???" this is an indication that it is receiving multicast packets which did not originate from an instance of the Multicast test. This will occur if you run the test with the same multicast address as a running Coherence cluster, or any other multicast application.
- Multiple NICs—If your machines have multiple network interfaces you may try specifying an explicit interface by using the `-local test` parameter. For instance if `Server A` has two interfaces with IP addresses 195.0.0.1 and 195.0.100.1, including `-local 195.0.0.1` on the test command line would ensure that the multicast packets used the first interface. You may also need to explicitly set your machines routing table to forward multicast traffic through the desired network interface. This can be done by issuing the command in [Example 43-5](#):

Example 43-5 Command to Set Machine Routing Table

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

Where `eth1` is the device which will be designated to transmit multicast traffic.

- AIX—On AIX systems you may run into the following multicast issues:
 - IPv6—In addition to specifying `java.net.preferIPv4Stack=true` you may need to configure the OS to perform IPv4 name resolution by adding `hosts=local,bind4` to your `/etc/netsvc.conf` file.
 - Virtual IP (VIPA)—AIX does not support multicast with VIPA. If using VIPA either bind multicast to a non-VIPA device, or run Coherence with multicast disabled. See "[well-known-addresses](#)" on page A-86 for details.
 - MTU—Configure the MTU for the multicast device to 1500 bytes.
- Cisco Switches—See "[Deploying to Cisco Switches](#)" on page 32-2 for the list of known issues.
- Foundry Switches—See "[Deploying to Foundry Switches](#)" on page 32-5 for the list of known issues.

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

Performing a Datagram Test for Network Performance

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

To achieve maximum performance it is suggested that you tune your environment based on the results of these tests. See [Chapter 45, "Performance Tuning"](#) for more information.

Running the Datagram Test Utility

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

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

[Table 44–1](#) describes the available command line options for the Datagram Test utility.

Table 44–1 *Command Line Options for the Datagram Test Utility*

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

Table 44–1 (Cont.) Command Line Options for the Datagram Test Utility

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

Sample Commands for a Listener and a Publisher

The following command line is for a listener:

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

The following command line is for a publisher:

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

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

Datagram Test Example

Presume that you want to test network performance between two servers— Server A with IP address `195.0.0.1` and Server B with IP address `195.0.0.2`. One server will act as a packet publisher and the other as a packet listener, the publisher will transmit packets as fast as possible and the listener will measure and report performance statistics. First start the listener on Server A.

Example 44–1 Command to Start a Listener

```
datagram-test.sh
```

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

Example 44–2 Output from Starting a Listener

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

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

report an error and exit. You can either decrease the requested buffer size using the `-rxBufferSize` parameter or increase your operating system network buffer settings. For best performance it is recommended that you increase the operating system buffers. See [Chapter 33, "Production Checklist"](#) for tuning your operating system for Coherence.

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

Example 44-3 Command to Start a Publisher

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

After pressing ENTER, you should see the new Datagram test instance on `Server B` start both a listener and a publisher. Note in this configuration `Server B` listener will not be used. The output illustrates in [Example 44-4](#) should appear in the `Server B` command window.

Example 44-4 Datagram Test—Starting a Listener and a Publisher on a Server

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

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

no packet burst limit
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
```

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

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

Reporting

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

Publisher Statistics

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

Example 44-5 Sample Publisher Report

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

now: 98 MB/sec, 69735 packets/sec

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

Listener Statistics

[Table 44–2](#) describes the statistics that can be reported by the listener.

Table 44–2 Listener Statistics

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

As with the publisher both current and lifetime statistics are report. [Example 44–6](#) displays a typical report:

Example 44–6 Sample Lifetime Statistics

Lifetime:

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

Now:

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

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

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

Throttling

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

Bidirectional Testing

You may also run the test in a bidirectional mode where both servers act as publishers and listeners. To do this simply restart test instances, supplying the instance on Server A with Server B's address, by running the command in [Example 44-7](#) on Server A.

Example 44-7 *Running Datagram Test in Bi-Directional Mode*

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

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

Distributed Testing

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

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

On Server A:

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

On Server B:

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

On Server C:

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

On Server D:

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

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

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

Performance Tuning

To achieve maximum performance with Coherence it is suggested that you test and tune your operating environment. Testing is covered in [Chapter 44, "Performing a Datagram Test for Network Performance."](#)

Tuning recommendations are available for:

- [Operating System Tuning](#)
- [Network Tuning](#)
- [JVM Tuning](#)
- [Coherence Network Tuning](#)
- [Data Access Patterns](#)

Operating System Tuning

- [Socket Buffer Sizes](#)
- [High Resolution timesource \(Linux\)](#)
- [Datagram size \(Microsoft Windows\)](#)
- [Thread Scheduling \(Microsoft Windows\)](#)
- [Swapping](#)

Socket Buffer Sizes

To help minimization of packet loss, the operating system socket buffers need to be large enough to handle the incoming network traffic while your Java application is paused during garbage collection. By default Coherence will attempt to allocate a socket buffer of 2MB. If your operating system is not configured to allow for large buffers Coherence will use smaller buffers. Most versions of UNIX have a very low default buffer limit, which should be increased to at least 2MB.

Starting with Coherence 3.1 you will receive the following warning if the operating system failed to allocate the full size buffer.

Example 45–1 Message Indicating OS Failed to Allocate the Full Buffer Size

```
UnicastUdpSocket failed to set receive buffer size to 1428 packets (2096304
bytes); actual size is 89 packets (131071 bytes). Consult your OS documentation
regarding increasing the maximum socket buffer size. Proceeding with the actual
value may cause sub-optimal performance.
```

Though it is safe to operate with the smaller buffers it is recommended that you configure your operating system to allow for larger buffers.

On Linux execute (as root):

```
sysctl -w net.core.rmem_max=2096304
sysctl -w net.core.wmem_max=2096304
```

On Solaris execute (as root):

```
ndd -set /dev/udp udp_max_buf 2096304
```

On AIX execute (as root):

```
no -o rfc1323=1
no -o sb_max=4194304
```

Note: Note that AIX only supports specifying buffer sizes of 1MB, 4MB, and 8MB. Additionally there is an issue with IBM's 1.5 JVMs which may prevent them from allocating socket buffers larger than 64K. This issue has been addressed in IBM's 1.5 SR3 SDK.

On Windows:

Windows does not impose a buffer size restriction by default.

Other:

For information on increasing the buffer sizes for other operating systems please refer to your operating system's documentation.

You may configure Coherence to request alternate sized buffers for packet publishers and unicast listeners by using the `coherence/cluster-config/packet-publisher/packet-buffer/maximum-packets` and `coherence/cluster-config/unicast-listener/packet-buffer/maximum-packets` elements. For more information, see "[packet-buffer](#)" on page A-48.

High Resolution timesource (Linux)

Linux has several high resolution timesources to choose from, the fastest TSC (Time Stamp Counter) unfortunately is not always reliable. Linux chooses TSC by default, and during boot checks for inconsistencies, if found it switches to a slower safe timesource. The slower time sources can be 10 to 30 times more expensive to query than the TSC timesource, and may have a measurable impact on Coherence performance. Note that Coherence and the underlying JVM are not aware of the timesource which the operating system is using. It is suggested that you check your system logs (`/var/log/dmesg`) to verify that the following is not present.

[Example 45-2](#) illustrates a sample timesource log.

Example 45-2 Log Message from a Linux Timesource

```
kernel: Losing too many ticks!
kernel: TSC cannot be used as a timesource.
kernel: Possible reasons for this are:
kernel:   You're running with Speedstep,
kernel:   You don't have DMA enabled for your hard disk (see hdparm),
kernel:   Incorrect TSC synchronization on an SMP system (see dmesg).
kernel: Falling back to a sane timesource now.
```

As the log messages suggest, this can be caused by a variable rate CPU (SpeedStep), having DMA disabled, or incorrect TSC synchronization on multi CPU machines. If present it is suggested that you work with your system administrator to identify and correct the cause allowing the TSC timesource to be utilized.

In some older versions of Linux there is a bug related to unsynchronized TSCs in multiprocessor systems. This bug can result in the clock apparently jumping forward by 4398 seconds, and then immediately jumping back to the correct time. This bug can trigger erroneous Coherence packet transmission timeouts resulting in nodes being incorrectly removed from the cluster. If you experience cluster disconnects along with a warning of a "4398 second" timeout it is suggested that you upgrade your Linux kernel. An example of such a log warning is as follows:

```
A potential communication problem has been detected. A packet has failed to be
delivered (or acknowledged) after 4398 seconds ...
```

See the following resource for more details on this Linux TSC issue:

<http://lkml.org/lkml/2007/8/23/96>

https://bugzilla.redhat.com/show_bug.cgi?id=452185

Datagram size (Microsoft Windows)

Microsoft Windows supports a fast IO path which is used when sending "small" datagrams. The default setting for what is considered a small datagram is 1024 bytes; increasing this value to match your network MTU (normally 1500) can significantly improve network performance.

To adjust this parameter:

1. Run Registry Editor (regedit)
2. Locate the following registry key
HKLM\System\CurrentControlSet\Services\AFD\Parameters
3. Add the following new DWORD value Name: FastSendDatagramThreshold
Value: 1500 (decimal)
4. Reboot

Note: Included in Coherence 3.1 and above is an `optimize.reg` script which will perform this change for you, it can be found in the `coherence/bin` directory of your installation. After running the script you must reboot your computer for the changes to take effect.

See Appendix C of

<http://technet.microsoft.com/en-us/library/bb726981.aspx> for more details on this parameter

Thread Scheduling (Microsoft Windows)

Windows (including NT, 2000 and XP) is optimized for desktop application usage. If you run two console ("DOS box") windows, the one that has the focus can use almost 100% of the CPU, even if other processes have high-priority threads in a running state. To correct this imbalance, you must configure the Windows thread scheduling to less-heavily favor foreground applications.

1. Open the Control Panel.
2. Open **System**.
3. Select the **Advanced** tab.
4. Under **Performance** select **Settings**.
5. Select the **Advanced** tab.
6. Under **Processor** scheduling, choose **Background services**.

Note: Coherence includes an `optimize.reg` script which will perform this change for you, it can be found in the `coherence/bin` directory of your installation.

Swapping

Ensure that you have sufficient memory such that you are not making active use of swap space on your machines. You may monitor the swap rate using tools such as `vmstat` and `top`. If you find that you are actively moving through swap space this will likely have a significant impact on Coherence's performance. Often this will manifest itself as Coherence nodes being removed from the cluster due to long periods of unresponsiveness caused by them having been "swapped out".

Network Tuning

- [Network Interface Settings](#)
- [Bus Considerations](#)
- [Network Infrastructure Settings](#)
- [Ethernet Flow-Control](#)
- [Path MTU](#)

Network Interface Settings

Verify that your Network card (NIC) is configured to operate at it's maximum link speed and at full duplex. The process for doing this varies between operating systems.

On Linux execute (as root):

```
ethtool eth0
```

See the man page on `ethtool` for further details and for information on adjust the interface settings.

On Solaris execute (as root):

```
kstat ce:0 | grep link_
```

This will display the link settings for interface 0. Items of interest are `link_duplex` (2 = full), and `link_speed` which is reported in Mbps.

Note: If running on Solaris 10, please review Sun issues 102712 and 102741 which relate to packet corruption and multicast disconnections. These will most often manifest as either `EOFExceptions`, "Large gap" warnings while reading packet data, or frequent packet timeouts. It is highly recommend that the patches for both issues be applied when using Coherence on Solaris 10 systems.

On Windows:

1. Open the Control Panel.
2. Open **Network Connections**.
3. Open the **Properties** dialog for desired network adapter.
4. Select **Configure**.
5. Select the **Advanced** tab.
6. Locate the driver specific property for **Speed & Duplex**.
7. Set it to either auto or to a specific speed and duplex setting.

Bus Considerations

For 1Gb and faster PCI network cards the system's bus speed may be the limiting factor for network performance. PCI and PCI-X busses are half-duplex, and all devices will run at the speed of the slowest device on the bus. Standard PCI buses have a maximum throughput of approximately 1Gb/sec and thus are not capable of fully using a full-duplex 1Gb NIC. PCI-X has a much higher maximum throughput (1GB/sec), but can be hobbled by a single slow device on the bus. If you find that you are not able to achieve satisfactory bidirectional data rates it is suggested that you evaluate your machine's bus configuration. For instance simply relocating the NIC to a private bus may improve performance.

Network Infrastructure Settings

If you experience frequent multi-second communication pauses across multiple cluster nodes you may need to increase your switch's buffer space. These communication pauses can be identified by a series of Coherence log messages identifying communication delays with multiple nodes which are not attributable to local or remote GCs.

Example 45–3 *Message Indicating a Communication Delay*

```
Experienced a 4172 ms communication delay (probable remote GC) with Member(Id=7,
Timestamp=2006-10-20 12:15:47.511, Address=192.168.0.10:8089, MachineId=13838);
320 packets rescheduled, PauseRate=0.31, Threshold=512
```

Some switches such as the Cisco 6500 series support configuration the amount of buffer space available to each Ethernet port or ASIC. In high load applications it may be necessary to increase the default buffer space. On Cisco this can be accomplished by executing:

```
fabric buffer-reserve high
```

See Cisco's documentation for additional details on this setting.

Ethernet Flow-Control

Full duplex Ethernet includes a flow-control feature which allows the receiving end of a point to point link to slow down the transmitting end. This is implemented by the receiving end sending an Ethernet PAUSE frame to the transmitting end, the transmitting end will then halt transmissions for the interval specified by the PAUSE frame. Note that this pause blocks **all** traffic from the transmitting side, even traffic destined for machines which are not overloaded. This can induce a head of line blocking condition, where one overloaded machine on a switch effectively slows down all other machines. Most switch vendors will recommend that Ethernet flow-control be disabled for inter switch links, and at most be used on ports which are directly connected to machines. Even in this setup head of line blocking can still occur, and thus it is advisable to disable Ethernet flow-control all together. Higher level protocols such as TCP/IP and Coherence TCMP include their own flow-control mechanisms which are not subject to head of line blocking, and also negate the need for the lower level flow-control.

See <http://www.networkworld.com/netresources/0913flow2.html> for more details on this subject

Path MTU

By default Coherence assumes a 1500 byte network MTU, and uses a default packet size of 1468 based on this assumption. Having a packet size which does not fill the MTU will result in an under used network. If your equipment uses a different MTU, then configure Coherence by specifying a packet size which is 32 bytes smaller than the network path's minimal MTU. The packet size may be specified in `coherence/cluster-config/packet-publisher/packet-size/maximum-length` and `preferred-length` configuration elements. For more information on these elements, see "[packet-size](#)" on page A-54.

If you are unsure of your equipment's MTU along the full path between nodes you can use either the standard `ping` or `tracert` utilities to determine it. To do this, execute a series of `ping` or `tracert` operations between the two machines. With each attempt you will specify a different packet size, starting from a high value and progressively moving downward until the packets start to make it through without fragmentation. You will need to specify a particular packet size, and to not fragment the packets.

On Linux execute:

```
ping -c 3 -M do -s 1468 serverb
```

On Solaris execute:

```
tracert -F serverb 1468
```

On Windows execute:

```
ping -n 3 -f -l 1468 serverb
```

On other operating systems: Consult the documentation for the `ping` or `tracert` command to see how to disable fragmentation, and specify the packet size.

If you receive a message stating that packets must be fragmented then the specified size is larger than the path's MTU. Decrease the packet size until you find the point at which packets can be transmitted without fragmentation. If you find that you need to use packets smaller than 1468 you may want to contact your network administrator to get the MTU increased to at least 1500.

JVM Tuning

- [Basic Sizing Recommendation](#)
- [Heap Size Considerations](#)
- [GC Monitoring & Tuning](#)

Basic Sizing Recommendation

The recommendations in this section are sufficient for general use cases and require minimal setup effort. The primary issue to consider when sizing your JVMs is a balance of available RAM versus garbage collection (GC) pause times.

Cache Servers

The standard, safe recommendation for Coherence cache servers is to run a fixed size heap of up to 1GB. Additionally, it is recommended to utilize an incremental garbage collector to minimize GC pause durations. Lastly, run all Coherence JVMs in server mode, by specifying the `-server` on the JVM command line. This allows for several performance optimizations for long running applications.

For example:

```
java -server -Xms1g -Xmx1g -Xincgc -Xloggc: -cp coherence.jar  
com.tangosol.net.DefaultCacheServer
```

This sizing allows for good performance without the need for more elaborate JVM tuning. For more information on garbage collection, see "[GC Monitoring & Tuning](#)" on page 45-11.

Note: It is possible to run cache servers with larger heap sizes; though it becomes more important to monitor and tune the JVMs to minimize the GC pauses. It may also be necessary to alter the storage ratios such that the amount of scratch space is increased to facilitate faster GC compactions. Additionally it is recommended that you make use of an up to date JVM version such as HotSpot 1.6 as it includes significant improvements for managing large heaps. See "[Heap Size Considerations](#)" on page 45-7.

TCMP Clients

Coherence TCMP clients should be configured similarly to cache servers as long GCs could cause them to be misidentified as being dead.

Extends Clients

Coherence Extend clients are not technically speaking cluster members and, as such, the effect of long GCs is less detrimental. For extend clients it is recommended that you follow the existing guidelines as set forth by the application in which you are embedding coherence.

Heap Size Considerations

This section will help you decide:

- How many CPUs you will need for your system
- How much memory you will need for each system

- How many JVMs to run per system
- How much heap to configure with each JVM

Since all applications are different, this section should be read as guidelines. You will need to answer the following questions in order to choose the configuration that is right for you:

- How much data will be stored in Coherence caches?
- What are the application requirements in terms of latency and throughput?
- How CPU or Network intensive is the application?

Sizing is an imprecise science. There is no substitute for frequent performance and stress testing.

The following topics are included in this section:

- [General Guidelines](#)
- [Moving the Cache Out of the Application Heap](#)

General Guidelines

Running with a fixed sized heap will save your JVM from having to grow the heap on demand and will result in improved performance. To specify a fixed size heap use the `-Xms` and `-Xmx` JVM options, setting them to the same value. For example:

```
java -server -Xms4G -Xmx4G ...
```

A JVM process consumes more system memory than the specified heap size. The heap size settings specify the amount of heap which the JVM makes available to the application, but the JVM itself will also consume additional memory. The amount consumed differs depending on the OS, and JVM settings. For instance, a HotSpot JVM running on Linux configured with a 1GB JVM will consume roughly 1.2GB of RAM. It is important to externally measure the JVMs memory utilization to ensure that RAM is not over committed. Tools such as `top`, `vmstat`, and Task Manager are useful in identifying how much RAM is actually being utilized.

Storage Ratios

The basic recommendation for how much data can be stored within a cache server of a given size is to use up to 1/3rd of the heap for primary cache storage. This leaves another 1/3rd for backup storage, and the final 1/3rd for scratch space. Scratch space is then used for things such as holding classes, temporary objects, network transfer buffers, and GC compaction. You may instruct Coherence to limit primary storage on a per-cache basis by configuring the `<high-units>` element and specifying a `BINARY` value for the `<unit-calculator>` element. These settings are automatically applied to backup storage as well.

Ideally both the primary and backup storage will also fit within the JVMs tenured space (for HotSpot based JVMs). See HotSpot's Tuning Garbage Collection guide for details on sizing the collectors generations:

http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html

Cache Topologies and Heap Size

For large datasets, Partitioned or Near caches are recommended. As the scalability of the Partitioned cache is linear for both reading and writing, varying the number of Coherence JVMs will not significantly affect cache performance. Using a Replicated cache will put significant pressure on GC.

Deciding How Many JVMs to Run Per System

The number of JVMs (nodes) to run per system depends on the system's number of processors/cores and amount of memory. As a starting point, we recommend starting with a plan to run one JVM for every two cores. This recommendation balances the following factors:

- Multiple JVMs per server allow Coherence to make more efficient use of network resources. Coherence's packet-publisher and packet-receiver have a fixed number of threads per JVM; as you add cores, you'll want to add JVMs to scale across these cores.
- Too many JVMs will increase contention and context switching on processors.
- Too few JVMs may not be able to handle available memory and may not fully utilize the NIC.
- Especially for larger heap sizes, JVMs must have available processing capacity to avoid long GC pauses.

Depending on your application, you can add JVMs up toward one per core. The recommended number of JVMs and amount of configured heap may also vary based on the number of processors/cores per socket and on the machine architecture.

Sizing Your Heap

When considering heap size, it is important to find the right balance. The lower bound is determined by per-JVM overhead (and also, manageability of a potentially large number of JVMs). For example, if there is a fixed overhead of 100MB for infrastructure software (e.g. JMX agents, connection pools, internal JVM structures), then the use of JVMs with 256MB heap sizes will result in close to 40% overhead for non-cache data. The upper bound on JVM heap size is governed by memory management overhead, specifically the maximum duration of GC pauses and the percentage of CPU allocated to GC (and other memory management tasks).

GC can affect the following:

- The latency of operations against Coherence. Larger heaps will cause longer and less predictable latency than smaller heaps.
- The stability of the cluster. With very large heaps, lengthy long garbage collection pauses can trick TCMP into believing a cluster member is dead since the JVM is unresponsive during GC pauses. Although TCMP takes GC pauses into account when deciding on member health, at some point it may decide the member is dead.

We offer the following guidelines:

- For Sun 1.5 JVM, we recommend not allocating more than 1GB and 2GB heap respectively.
- For Sun 1.6 JVM or JRockit, we recommend allocating up to a 4GB heap. We also recommend using Sun's Concurrent Mark and Sweep GC or JRockit's Deterministic GC.

The length of a GC pause scales worse than linearly to the size of the heap. That is, if you double the size of the heap, pause times due to GC will, in general, more than double. GC pauses are also impacted by application usage:

- Pause times increase as the amount of live data in the heap increases. We recommend not exceeding 70% live data in your heap. This includes primary data, backup data, indexes, and application data.

- High object allocation rates will increase pause times. Even "simple" Coherence applications can cause high object allocation rates since every network packet generates many objects.
- CPU-intensive computation will increase contention and may also contribute to higher pause times.

Depending on your latency requirements, you can increase allocated heap space beyond the above recommendations, but be sure to stress test your system.

Moving the Cache Out of the Application Heap

Using dedicated Coherence cache server instances for Partitioned cache storage will minimize the heap size of application JVMs as the data is no longer stored locally. As most Partitioned cache access is remote (with only $1/N$ of data being held locally), using dedicated cache servers does not generally impose much additional overhead. Near cache technology may still be used, and it will generally have a minimal impact on heap size (as it is caching an even smaller subset of the Partitioned cache). Many applications are able to dramatically reduce heap sizes, resulting in better responsiveness.

Local partition storage may be enabled (for cache servers) or disabled (for application server clients) with the `tangosol.coherence.distributed.localstorage` Java property (for example,

`-Dtangosol.coherence.distributed.localstorage=false`).

It may also be disabled by modifying the `<local-storage>` setting in the `tangosol-coherence.xml` (or `tangosol-coherence-override.xml`) file as follows:

Example 45–4 Disabling Partition Storage

```
<!--
Example using tangosol-coherence-override.xml
-->
<coherence>
  <cluster-config>
    <services>
      <!--
      id value must match what's in tangosol-coherence.xml for DistributedCache
      service
      -->
      <service id="3">
        <init-params>
          <init-param id="4">
            <param-name>local-storage</param-name>
            <param-value system-property="tangosol.coherence.distributed.
              localstorage">false</param-value>
          </init-param>
        </init-params>
      </service>
    </services>
  </cluster-config>
</coherence>
```

At least one storage-enabled JVM must be started before any storage-disabled clients access the cache.

GC Monitoring & Tuning

Lengthy GC pause times can negatively impact the Coherence cluster and are, for the most part, indistinguishable from node death. During these pauses, a Java application is unable to send or receive packets and in the case of receiving the operating system buffered packets, the packets may be discarded and need to be retransmitted. For these reasons, it is very important that cluster nodes are sized and/or tuned to ensure that their GC times remain minimal. As a good rule of thumb, a node should spend less than 10% of its time paused in GC, normal GC times should be under 100ms, and maximum GC times should be around 1 second.

GC activity can be monitored in a number of ways; some standard mechanisms include:

- JVM switch `-verbose:gc`
- JVM switch `-Xloggc:` (similar to verbose GC but includes timestamps)
- JVM switch `-Xprof:` (profiling information- profiling activities should be distinguished between testing and production deployments and its effects on resources and performance should be monitored as well)
- Over JMX via tools such as JConsole

Log messages will be generated when one cluster node detects that another cluster node has been unresponsive for a period, generally indicating that a target cluster node was in a GC cycle.

Example 45–5 Message Indicating Target Cluster Node is in Garbage Collection Mode

```
Experienced a 4172 ms communication delay (probable remote GC) with Member(Id=7,
Timestamp=2006-10-20 12:15:47.511, Address=192.168.0.10:8089, MachineId=13838);
320 packets rescheduled, PauseRate=0.31, Threshold=512
```

PauseRate indicates the percentage of time for which the node has been considered unresponsive since the statistics were last reset. Nodes reported as unresponsive for more than a few percent of their lifetime may be worth investigating for GC tuning.

Coherence Network Tuning

Coherence includes configuration elements for throttling the amount of traffic it will place on the network; see the documentation for [<traffic-jam>](#) and [<flow-control>](#), these settings are used to control the rate of packet flow within and between cluster nodes.

Validation

To determine how these settings are affecting performance you need to check if you're cluster nodes are experiencing packet loss and/or packet duplication. This can be obtained by looking at the following JMX statistics on various cluster nodes:

- `ClusterNodeMBean.PublisherSuccessRate`—If less than 1.0, packets are being detected as lost and being resent. Rates below 0.98 may warrant investigation and tuning.
- `ClusterNodeMBean.ReceiverSuccessRate`—If less than 1.0, the same packet is being received multiple times (duplicates), likely due to the publisher being overly aggressive in declaring packets as lost.
- `ClusterNodeMBean.WeakestChannel`—Identifies the remote cluster node which the current node is having most difficulty communicating with.

For information on using JMX to monitor Coherence see [Chapter 34, "How to Manage Coherence Using JMX."](#)

Data Access Patterns

- [Data Access Distribution \(hot spots\)](#)
- [Cluster-node Affinity](#)
- [Read/Write Ratio and Data Sizes](#)
- [Interleaving Cache Reads and Writes](#)

Data Access Distribution (hot spots)

When caching a large data set, typically a small portion of that data set will be responsible for most data accesses. For example, in a 1000 object datasets, 80% of operations may be against a 100 object subset. The remaining 20% of operations may be against the other 900 objects. Obviously the most effective return on investment will be gained by caching the 100 most active objects; caching the remaining 900 objects will provide 25% more effective caching while requiring a 900% increase in resources.

However, if every object is accessed equally often (for example in sequential scans of the datasets), then caching will require more resources for the same level of effectiveness. In this case, achieving 80% cache effectiveness would require caching 80% of the datasets versus 10%. (Note that sequential scans of partially cached data sets will generally defeat MRU, LFU and MRU-LFU eviction policies). In practice, almost all non-synthetic (benchmark) data access patterns are uneven, and will respond well to caching subsets of data.

In cases where a subset of data is active, and a smaller subset is particularly active, Near caching can be very beneficial when used with the "all" invalidation strategy (this is effectively a two-tier extension of the above rules).

Cluster-node Affinity

Coherence's Near cache technology will transparently take advantage of cluster-node affinity, especially when used with the "present" invalidation strategy. This topology is particularly useful when used with a sticky load-balancer. Note that the "present" invalidation strategy results in higher overhead (as opposed to "all") when the front portion of the cache is "thrashed" (very short lifespan of cache entries); this is due to the higher overhead of adding/removing key-level event listeners. In general, a cache should be tuned to avoid thrashing and so this is usually not an issue.

Read/Write Ratio and Data Sizes

Generally speaking, the following cache topologies are best for the following use cases:

- Replicated cache—small amounts of read-heavy data (for example, metadata)
- Partitioned cache—large amounts of read/write data (for example, large data caches)
- Near cache—similar to Partitioned, but has further benefits from read-heavy tiered access patterns (for example, large data caches with hotspots) and "sticky" data access (for example, sticky HTTP session data). Depending on the synchronization method (expiry, asynchronous, synchronous), the worst case performance may range from similar to a Partitioned cache to considerably worse.

Interleaving Cache Reads and Writes

Interleaving refers to the number of cache reads between each cache write. The Partitioned cache is not affected by interleaving (as it is designed for 1:1 interleaving). The Replicated and Near caches by contrast are optimized for read-heavy caching, and prefer a read-heavy interleave (for example, 10 reads between every write). This is because they both locally cache data for subsequent read access. Writes to the cache will force these locally cached items to be refreshed, a comparatively expensive process (relative to the near-zero cost of fetching an object off the local memory heap). Note that with the Near cache technology, worst-case performance is still similar to the Partitioned cache; the loss of performance is relative to best-case scenarios.

Note that interleaving is related to read/write ratios, but only indirectly. For example, a Near cache with a 1:1 read/write ratio may be extremely fast (all writes followed by all reads) or much slower (1:1 interleave, write-read-write-read...).

Using the Service Guardian

The following sections are included in this chapter:

- [Overview](#)
- [Configuring the Service Guardian](#)
- [Issuing Manual Guardian Heartbeats](#)

Overview

The service guardian is a mechanism that detects and attempts to resolve deadlocks in Coherence threads. Deadlocked threads on a member may result in many undesirable behaviors that are visible to the rest of the cluster, such as the inability to add new nodes to the cluster and the inability to service requests by nodes currently in the cluster.

The service guardian receives periodic heartbeats that are issued by Coherence-owned and created threads. Should a thread fail to issue a heartbeat before the configured timeout, the service guardian takes corrective action. Both the timeout and corrective action (recovery) can be configured as required.

Note: The term deadlock does not necessarily indicate a true deadlock; a thread that does not issue a timely heartbeat may be executing a long running process or waiting on a slow resource. The service guardian does not have the ability to distinguish a deadlocked thread from a slow one.

Interfaces That Are Executed By Coherence

Implementations of the following interfaces are executed by Coherence-owned threads. Any processing in an implementation that exceeds the configured guardian timeout results in the service guardian attempting to recover the thread. The list is not exhaustive and only provides the most common interfaces that are implemented by end users.

```
com.tangosol.net.Invocable  
com.tangosol.net.cache.CacheStore  
com.tangosol.util.Filter  
com.tangosol.util.InvocableMap.EntryAggregator  
com.tangosol.util.InvocableMap.EntryProcessor  
com.tangosol.util.MapListener  
com.tangosol.util.MapTrigger
```

Understanding Recovery

The service guardian's recovery mechanism uses a series of steps to determine if a thread is deadlocked. Corrective action is taken if the service guardian concludes that the thread is deadlocked. The action to take can be configured and custom actions can be created if required. The recovery mechanism is outlined below:

- **Soft Timeout** – The recovery mechanism first attempts to interrupt the thread just before the configured timeout is reached. The following example log message demonstrates a soft timeout message:

```
<Error> (thread=DistributedCache, member=1): Attempting recovery (due to soft timeout) of Daemon{Thread="Thread[WriteBehindThread:CacheStoreWrapper (com.tangosol.examples.rwbm.TimeoutTest),5,WriteBehindThread:CacheStoreWrapper (com.tangosol.examples.rwbm.TimeoutTest)]", State=Running}
```

If the thread can be interrupted and it results in a heartbeat, normal processing resumes.

- **Hard Timeout** – The recovery mechanism attempts to stop a thread once the configured timeout is reached. The following example log message demonstrates a hard timeout message:

```
<Error> (thread=DistributedCache, member=1): Terminating guarded execution (due to hard timeout) of Daemon{Thread="Thread[WriteBehindThread:CacheStoreWrapper (com.tangosol.examples.rwbm.TimeoutTest),5,WriteBehindThread:CacheStoreWrapper (com.tangosol.examples.rwbm.TimeoutTest)]", State=Running}
```

- Lastly, if the thread cannot be stopped, the recovery mechanism performs an action based on the configured failure policy. Actions that can be performed include: shutting down the cluster service, shutting down the JVM, and performing a custom action. The following example log message demonstrates an action taken by the recovery mechanism:

```
<Error> (thread=Termination Thread, member=1): Write-behind thread timed out; stopping the cache service
```

Configuring the Service Guardian

The service guardian is enabled out-of-the box and has two configured items: the timeout value and the failure policy. The timeout value is the length of time the service guardian will wait to receive a heartbeat from a thread before starting recovery. The failure policy is the corrective action that the service guardian takes once it concludes that the thread is deadlocked.

Setting the Guardian Timeout

The service guardian timeout can be set in three different ways based on the level of granularity that is required:

- **All threads** – This option allows a single timeout value to be applied to all Coherence-owned threads on a cluster node. This is the out-of-box configuration and is set at 305000 milliseconds by default.
- **Threads per service type** – This option allows different timeout values to be set for specific service types. The timeout value is applied to the threads of all service instances. If a timeout is not specified for a particular service type, then the timeout defaults to the timeout that is set for all threads.

- Threads per service instance – This option allows different timeout values to be set for specific service instances. If a timeout is not set for a specific service instance, then the service's timeout value, if specified, is used; otherwise, the timeout that is set for all threads is used.

Setting the timeout value to 0 will stop threads from being guarded. In general, the service guardian timeout value should be set equal to or greater than the timeout value for packet delivery.

Note: The guardian timeout can also be used for cache store implementations that are configured with a read-write-backing-map scheme. In this case, the `<cachestore-timeout>` element is set to 0, which defaults the timeout to the guardian timeout. See ["read-write-backing-map-scheme"](#) on page B-85.

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:

```
<coherence>
  <cluster-config>
    <service-guardian>
      <timeout-milliseconds>120000</timeout-milliseconds>
    </service-guardian>
  </cluster-config>
</coherence>
```

The `<timeout-milliseconds>` value can also be set using the `tangosol.coherence.guard.timeout` system property.

Setting the Guardian Timeout Per Service Type

To set the guardian timeout per service type, override the service's `guardian-timeout` initialization parameter in an operational override file. The following example sets the guardian timeout for the `DistributedCache` service to 120000 milliseconds:

```
<coherence>
  <services>
    <service id="3">
      <init-params>
        <init-param id="18">
          <param-name>guardian-timeout</param-name>
          <param-value>120000</param-value>
        </init-param>
      </init-params>
    </service>
  </services>
</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 can be used to set the guardian timeout, respectively:

```
tangosol.coherence.distributed.guard.timeout
tangosol.coherence.replicated.guard.timeout
tangosol.coherence.optimistic.guard.timeout
tangosol.coherence.invocation.guard.timeout
tangosol.coherence.proxy.guard.timeout
```

Setting the Guardian Timeout Per Service Instance

To set the guardian timeout per service instance, add a `<guardian-timeout>` element to a cache scheme definition in the cache configuration file. The following example sets the guardian timeout for a distributed cache scheme to 120000 milliseconds.

```
...
<distributed-scheme>
  <scheme-name>example-distributed</scheme-name>
  <service-name>DistributedCache</service-name>
  <backing-map-scheme>
    <local-scheme>
      <scheme-ref>example-binary-backing-map</scheme-ref>
    </local-scheme>
  </backing-map-scheme>
  <autostart>true</autostart>
  <guardian-timeout>120000</guardian-timeout>
</distributed-scheme>
...
```

The `<guardian-timeout>` element can be used in the following schemes:

`<distributed-scheme>`, `<replicated-scheme>`, `<optimistic-scheme>`,
`<transaction-scheme>`, `<invocation-scheme>`, and `<proxy-scheme>`.

Using the Timeout Value From the PriorityTask API

Custom implementations of the `Invocable`, `EntryProcessor`, and `EntryAggregator` interface can implement the `com.tangosol.net.PriorityTask` interface. In this case, the service guardian attempts recovery after the task has been executing for longer than the value returned by `getExecutionTimeoutMillis()`. See [Chapter 27, "Priority Tasks,"](#) for more information on using the API.

The execution timeout can be set using the `<task-timeout>` element within an `<invocation-scheme>` element defined in the cache configuration file. For the `Invocation` service, the `<task-timeout>` element specifies the timeout value for `Invocable` tasks that implement the `PriorityTask` interface, but don't explicitly specify the execution timeout value; that is, the `getExecutionTimeoutMillis()` method returns 0.

If the `<task-timeout>` element is set to 0, the default guardian timeout is used. See [Appendix B, "Cache Configuration Elements"](#) for more information on the different cache schemes that support the use of the `<task-timeout>` element.

Setting the Guardian Service Failure Policy

The service failure policy determines the corrective action that the service guardian takes after it concludes that a thread is deadlocked. The following policies are available:

- `exit-cluster` – This policy attempts to recover threads that appear to be unresponsive. If the attempt fails, an attempt is made to stop the associated service. If the associated service cannot be stopped, this policy causes the local node to stop the cluster services. This is the default policy if no policy is specified.
- `exit-process` – This policy attempts to recover threads that appear to be unresponsive. If the attempt fails, an attempt is made to stop the associated service. If the associated service cannot be stopped, this policy cause the local node to exit the JVM and terminate abruptly.
- `logging` – This policy logs any detected problems but takes no corrective action.
- `custom` – the name of a Java class that provides an implementation for the `com.tangosol.net.ServiceFailurePolicy` interface. See ["Enabling a Custom Guardian Failure Policy"](#) on page 46-6.

The service guardian failure policy can be set three different ways based on the level of granularity that is required:

- `All threads` – This option allows a single failure policy to be applied to all Coherence-owned threads on a cluster node. This is the out-of-box configuration.
- `Threads per service type` – This option allows different failure policies to be set for specific service types. The policy is applied to the threads of all service instances. If a policy is not specified for a particular service type, then the timeout defaults to the timeout that is set for all threads.
- `Threads per service instance` – This option allows different failure policies to be set for specific service instances. If a policy is not set for a specific service instance, then the service's policy, if specified, is used; otherwise, the policy that is set for all threads is used.

Setting the Guardian Failure Policy for All Threads

To set a guardian failure policy, add a `<service-failure-policy>` element to an operational override file within the `<service-guardian>` element. The following example sets the failure policy to `exit-process`:

```
<coherence>
  <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:

```
<coherence>
  <services>
    <service id="3">
      <init-params>
        <init-param id="19">
          <param-name>service-failure-policy</param-name>
          <param-value>logging</param-value>
        </init-param>
      </init-params>
    </service>
  </services>
```

```
    </service>
  </services>
</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>
  <backing-map-scheme>
    <local-scheme>
      <scheme-ref>example-binary-backing-map</scheme-ref>
    </local-scheme>
  </backing-map-scheme>
  <autostart>true</autostart>
  <guardian-timeout>120000</guardian-timeout>
  <service-failure-policy>logging</service-failure-policy>
</distributed-scheme>
...
```

The `<service-failure-policy>` element can be used in the following schemes: `<distributed-scheme>`, `<replicated-scheme>`, `<optimistic-scheme>`, `<transaction-scheme>`, `<invocation-scheme>`, and `<proxy-scheme>`.

Enabling a Custom Guardian Failure Policy

To use a custom failure policy, include an `<instance>` subelement and provide a fully qualified class name that implements the `ServiceFailurePolicy` interface. See "[instance](#)" on page A-26 for detailed instructions on using the `<instance>` element. The following example enables a custom failure policy that is implemented in the `MyFailurePolicy` class. Custom failure policies can be enabled for all threads (as shown below) or can be enabled per service instance within a cache scheme definition.

```
<coherence>
  <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 in order 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 will perform object instantiation. The following example gets a custom failure policy instance using the `getPolicy` method on the `MyPolicyFactory` class.

```
<coherence>
  <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.

```
<coherence>
  <cluster-config>
    <service-guardian>
      <service-failure-policy>
        <instance>
          <class-name>package.MyFailurePolicy</class-name>
          <init-params>
            <init-param>
              <param-name>iMaxTime</param-name>
              <param-value>2000</param-value>
            </init-param>
          </init-params>
        </instance>
      </service-failure-policy>
    </service-guardian>
  </cluster-config>
</coherence>
```

Issuing Manual Guardian Heartbeats

The `com.tangosol.net.GuardSupport` class provides heartbeat methods that applications can use to manually issue heartbeats to the guardian:

```
GuardSupport.heartbeat();
```

For known long running operations, the heartbeat can be issued with the number of milliseconds that should pass before the operation is considered "stuck:"

```
GuardSupport.heartbeat(long cMillis);
```

Using Quorum

The following sections are included in this chapter:

- [Overview](#)
- [Using the Cluster Quorum](#)
- [Using the Partitioned Cache Quorums](#)
- [Using the Proxy Quorum](#)
- [Enabling Custom Action Policies](#)

Overview

A quorum, in Coherence, refers to the minimum number of service members that are required in a cluster before a service action is allowed or disallowed. Quorums are beneficial because they automatically provide assurances that a cluster behaves in an expected way when member thresholds are reached. For example, a partitioned cache backup quorum might require at least 5 storage-enabled members before the partitioned cache service is allowed to back up partitions.

Quorums are service-specific and defined within a quorum policy; there is a cluster quorum policy for the Cluster service, a partitioned quorum policy for the Partitioned Cache service, and a proxy quorum policy for the Proxy service. Quorum thresholds are set on the policy using a cache configuration file.

Each quorum provides benefits with respect to its particular service. However, in general, quorums:

- control service behavior at different service member levels
- mandate the minimum service member levels that are required for service operations
- ensure an optimal cluster and cache environment for a particular application or solution

Using the Cluster Quorum

The cluster quorum policy defines a single quorum (the timeout survivor quorum) for the Cluster Service. The timeout survivor quorum mandates the minimum number of cluster members that must remain in the cluster when the cluster service is terminating suspect members. A member is considered suspect if it has not responded to network communications and is in imminent danger of being disconnected from the cluster. The quorum can be specified generically across all members or constrained to members that have a specific role in the cluster, such as client or server members. See

the `<role-name>` element in "[member-identity](#)" on page A-38 for more information on defining role names for cluster members.

This quorum is typically used in environments where network performance varies. For example, intermittent network outages may cause a high number of cluster members to be removed from the cluster. Using this quorum, a certain number of members are maintained during the outage and are available once the network recovers. This behavior also minimizes the manual intervention required to restart members. Naturally, requests that require cooperation by the nodes that are not responding will not be able to complete and will be either blocked for the duration of the outage or will be timed out.

Configuring the Cluster Quorum Policy

The timeout survivor quorum threshold is configured in an operational override file using the `<timeout-survivor-quorum>` element and optionally the `role` attribute. This element must be used within a `<cluster-quorum-policy>` element. The following example demonstrates configuring the timeout survivor quorum threshold to ensure that 5 cluster members with the `server` role are always kept in the cluster while removing suspect members:

```
<cluster-config>
  <member-identity>
    <role-name>server</role-name>
  </member-identity>
  <cluster-quorum-policy>
    <timeout-survivor-quorum role="Server">5</timeout-survivor-quorum>
  </cluster-quorum-policy>
</cluster-config>
```

Using the Partitioned Cache Quorums

The partitioned cache quorum policy defines four quorums for the partitioned cache service (DistributedCache) that mandate how many service members are required before different partitioned cache service operations can be performed:

- **Distribution Quorum** – This quorum mandates the minimum number of storage-enabled members of a partitioned cache service that must be present before the partitioned cache service is allowed to perform partition distribution.
- **Restore Quorum** – This quorum mandates the minimum number of storage-enabled members of a partitioned cache service that must be present before the partitioned cache service is allowed to restore lost primary partitions from backup.
- **Read Quorum** – This quorum specifies the minimum number of storage-enabled members of a partitioned cache service that must be present in order 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 specifies the minimum number of storage-enabled members of a partitioned cache service that must be present in order to process write requests. A write request is any request that may mutate the state or contents of a cache.

These quorums are typically used to indicate at what service member levels different service operations are best performed given the intended use and requirements of a distributed cache. For example, a small distributed cache may only require three

storage-enabled members to adequately store data and handle projected request volumes. While; a large distributed cache may require 10, or more, storage-enabled members to adequately store data and handle projected request volumes. Optimal member levels are tested during development and then set accordingly to ensure that the minimum service member levels are provisioned in a production environment.

If the number of storage-enabled nodes running the service drops below the configured level of read or write quorum, the corresponding client operation will be 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.

```
...
<cache-schemes>
  <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>
      <restore-quorum>3</restore-quorum>
      <distribution-quorum>4</distribution-quorum>
      <read-quorum>3</read-quorum>
      <write-quorum>5</write-quorum>
    </partitioned-quorum-policy-scheme>
    <autostart>true</autostart>
  </distributed-scheme>
  ...
</cache-schemes>
```

The `<partitioned-quorum-policy-scheme>` element also supports the use of scheme references. In the below example, a `<partitioned-quorum-policy-scheme>`, with the name `partitioned-cache-quorum`, is referenced from within the `<distributed-scheme>` element:

```
...
<cache-schemes>
  <partitioned-quorum-policy-scheme>
    <scheme-name>partitioned-cache-quorum</scheme-name>
    <restore-quorum>3</restore-quorum>
    <distribution-quorum>4</distribution-quorum>
    <read-quorum>3</read-quorum>
    <write-quorum>5</write-quorum>
  </partitioned-quorum-policy-scheme>

  <distributed-scheme>
    <scheme-name>partitioned-cache-with-quorum</scheme-name>
```

```
<service-name>PartitionedCacheWithQuorum</service-name>
<backing-map-scheme>
  <local-scheme/>
</backing-map-scheme>
<partitioned-quorum-policy-scheme>
  <scheme-ref>partitioned-cache-quorum</scheme-ref>
</partitioned-quorum-policy-scheme>
<autostart>true</autostart>
</distributed-scheme>
...
```

Using the Proxy Quorum

The proxy quorum policy defines a single quorum (the connection quorum) for the proxy service. The connection quorum mandates the minimum number of proxy service members that must be available before the proxy service can allow client connections.

This quorum is typically used to ensure enough proxy service members are available to optimally support a given set of TCP clients. For example, a small number of clients may efficiently connect to a cluster using two proxy services. While; a large number of clients may require 3 or more proxy services to efficiently connect to a cluster. Optimal levels are tested during development and then set accordingly to ensure that the minimum service member levels are provisioned in a production environment.

Configuring the Proxy Quorum Policy

The connection quorum threshold is configured in a cache configuration file within the `<proxy-quorum-policy-scheme>` element. The element must be used within a `<proxy-scheme>` element. The following example demonstrates configuring the connection quorum threshold to ensures that 3 proxy service members are present in the cluster before the proxy service is allowed to accept TCP client connections:

```
...
<caching-schemes>
  <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>
    <autostart>true</autostart>
    <proxy-quorum-policy-scheme>
      <connect-quorum>3</connect-quorum>
    </proxy-quorum-policy-scheme>
  </proxy-scheme>
  ...
</caching-schemes>
```

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:

```
...
```

```

<キャッシング-schemes>
  <proxy-quorum-policy-scheme>
    <scheme-name>proxy-quorum</scheme-name>
    <connect-quorum>3</connect-quorum>
  </proxy-quorum-policy-scheme>

  <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>
    <autostart>true</autostart>
    <proxy-quorum-policy-scheme>
      <scheme-ref>proxy-quorum</scheme-ref>
    </proxy-quorum-policy-scheme>
  </proxy-scheme>
  ...

```

Enabling Custom Action Policies

Custom action policies can be used instead of the default quorum policies for the Cluster service, Partitioned Cache service, and Proxy service. Custom action policies must implement the `com.tangosol.net.ActionPolicy` interface.

To enable a custom policy, add a `<class-name>` element within a quorum policy scheme element that contains the fully qualified name of the implementation class. The following example adds a custom action policy to the partitioned quorum policy for a distributed cache scheme definition:

```

...
<キャッシング-schemes>
  <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 be used to 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 will perform object instantiation. For example.

```

...
<キャッシング-schemes>
  <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-name>
  <method-name>createPolicy</method-name>
</partitioned-quorum-policy-scheme>
<autostart>true</autostart>
</distributed-scheme>
...
```

Scaling Out Your Data Grid Aggregations Linearly

Coherence provides a data grid by dynamically, transparently, and automatically partitioning the data set across all storage enabled nodes in a cluster. We have been doing some scale out testing on our new Dual 2.3GHz PowerPC G5 Xserve cluster and here is one of the tests that we have performed using the data grid aggregation feature.

The `InvocableMap` interface tightly binds the concepts of a data grid (that is, partitioned cache) and the processing of the data stored in the grid. When you take the `InvocableMap` and combine it with the linear scalability of Coherence itself you get an **extremely powerful** solution. The following tests show that you can take an application that Coherence provides you (the developer, the engineer, the architect, and so on) the ability to build an application when that can scale out to handle any SLA requirement, any increase in throughput requirements. For example, the following test demonstrate Coherence's ability to linearly increase the number of trades aggregated per second as you increase hardware. That is, if one machine can aggregate **X** trades per second, if you add a second machine to the data grid you will be able to aggregate **2X** trades per second, if you add a third machine to the data grid you will be able to aggregate **3X** trades per second and so on.

All of the Data Grid capabilities described below are features of Coherence Enterprise Edition and higher.

The Data

First, we need some data to aggregate. [Example 48-1](#) illustrates a `Trade` object with a three properties `Id`, `Price`, and `Symbol`.

Example 48-1 Trade Object Defining Three Properties

```
package com.tangosol.examples.coherence.data;

import com.tangosol.util.Base;
import com.tangosol.util.ExternalizableHelper;
import com.tangosol.io.ExternalizableLite;

import java.io.IOException;
import java.io.NotActiveException;
import java.io.DataInput;
import java.io.DataOutput;
```

```
/**
 * Example Trade class
 *
 * @author erm 2005.12.27
 */
public class Trade
    extends Base
    implements ExternalizableLite
{
    /**
     * Default Constructor
     */
    public Trade()
    {
    }

    public Trade(int iId, double dPrice, String sSymbol)
    {
        setId(iId);
        setPrice(dPrice);
        setSymbol(sSymbol);
    }

    public int getId()
    {
        return m_iId;
    }

    public void setId(int iId)
    {
        m_iId = iId;
    }

    public double getPrice()
    {
        return m_dPrice;
    }

    public void setPrice(double dPrice)
    {
        m_dPrice = dPrice;
    }

    public String getSymbol()
    {
        return m_sSymbol;
    }

    public void setSymbol(String sSymbol)
    {
        m_sSymbol = sSymbol;
    }

    /**
     * Restore the contents of this object by loading the object's state from the
     * passed DataInput object.
     *
     * @param in the DataInput stream to read data from to restore the
     *          state of this object
     */
}
```

```

* @throws IOException      if an I/O exception occurs
* @throws NotActiveException if the object is not in its initial state, and
*                           therefore cannot be deserialized into
*/
public void readExternal(DataInput in)
    throws IOException
{
    m_iId    = ExternalizableHelper.readInt(in);
    m_dPrice = in.readDouble();
    m_sSymbol = ExternalizableHelper.readSafeUTF(in);
}

/**
 * Save the contents of this object by storing the object's state into the
 * passed DataOutput object.
 *
 * @param out the DataOutput stream to write the state of this object to
 *
 * @throws IOException if an I/O exception occurs
 */
public void writeExternal(DataOutput out)
    throws IOException
{
    ExternalizableHelper.writeInt(out, m_iId);
    out.writeDouble(m_dPrice);
    ExternalizableHelper.writeSafeUTF(out, m_sSymbol);
}

private int    m_iId;
private double m_dPrice;
private String m_sSymbol;
}

```

Configure a Partitioned Cache

The [Cache Configuration Elements](#) are used to configure a cache. [Example 48–2](#) defines one wildcard cache-mapping that maps to one caching-scheme which has unlimited capacity:

Example 48–2 Mapping a cache-mapping to a caching-scheme with Unlimited Capacity

```

<?xml version="1.0"?>

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

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>*</cache-name>
      <scheme-name>example-distributed</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <!--
    Distributed caching scheme.
    -->
    <distributed-scheme>
      <scheme-name>example-distributed</scheme-name>
      <service-name>DistributedCache</service-name>
    </distributed-scheme>
  </caching-schemes>
</cache-config>

```

```
<backing-map-scheme>
  <class-scheme>
    <scheme-ref>unlimited-backing-map</scheme-ref>
  </class-scheme>
</backing-map-scheme>

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

<!--
Backing map scheme definition used by all the caches that do
not require any eviction policies
-->
<class-scheme>
  <scheme-name>unlimited-backing-map</scheme-name>

  <class-name>com.tangosol.util.SafeHashMap</class-name>
  <init-params></init-params>
</class-scheme>
</caching-schemes>
</cache-config>
```

Add an Index to the Price Property

[Example 48–3](#) illustrates the code to add an index to the `Price` property. Adding an index to this property increases performance by allowing Coherence to access the values directly rather than having to deserialize each item to accomplish the calculation

Example 48–3 Adding an Index to the Price Property

```
ReflectionExtractor extPrice = new ReflectionExtractor("getPrice");
m_cache.addIndex(extPrice, true, null);
```

In our tests the aggregation speed **was increased by more than 2x** after an index was applied.

Code to perform a Parallel Aggregation

[Example 48–4](#) illustrates the code to perform a parallel aggregation across all JVMs in the data grid. The aggregation is initiated and results received by **a single client**. That is, a single "low-power" client is able to use the full processing power of the cluster/data grid in aggregate to perform this aggregation in parallel with **just one line of code**.

Example 48–4 Perform a Parallel Aggregation Across all JVMs in the Grid

```
Double DResult;
DResult = (Double) m_cache.aggregate((Filter) null, new DoubleSum("getPrice"));
```

The Testing Environment and Process

Performing a "Test Run"

A test run does several things:

1. Loads 200,000 trade objects into the data grid.
2. Adds indexes to `Price` property.
3. Performs a **parallel** aggregation of **all** trade objects stored in the data grid. This aggregation step is done 20 times to obtain an "average run time" to ensure that the test takes into account garbage collection.
4. Loads 400,000 trade objects into the data grid.
5. Repeats steps 2 and 3.
6. Loads 600,000 trade objects into the data grid.
7. Repeats steps 2 and 3.
8. Loads 800,000 trade objects into the data grid.
9. Repeats steps 2 and 3.
10. Loads 1,000,000 trade objects into the data grid.
11. Repeats steps 2 and 3.

Client Considerations: The test client itself is run on an Intel Core Duo iMac which is marked as local storage disabled. The command line used to start the client was:

```
java ... -Dtangosol.coherence.distributed.localstorage=false -Xmx128m -Xms128m
com.tangosol.examples.coherence.invocable.TradeTest
```

This "Test Suite" (and Subsequent Results) Includes Data from Four "Test Runs":

1. Start 4 JVMs **on one** Xserve - Perform a "test run"
2. Start 4 JVMs **on each of two** Xserves - Perform a "test run"
3. Start 4 JVMs **on each of three** Xserves - Perform a "test run"
4. Start 4 JVMs **on each of four** Xserves - Perform a "test run"

Server Considerations: In this case a "JVM" refers to a cache server instance (that is, a data grid node) that is a standalone JVM responsible for managing/storing the data. I used the `DefaultCacheServer` helper class to accomplish this.

The command line used to start the server was:

```
java ... -Xmx384m -Xms384m -server com.tangosol.net.DefaultCacheServer
```

JDK Version

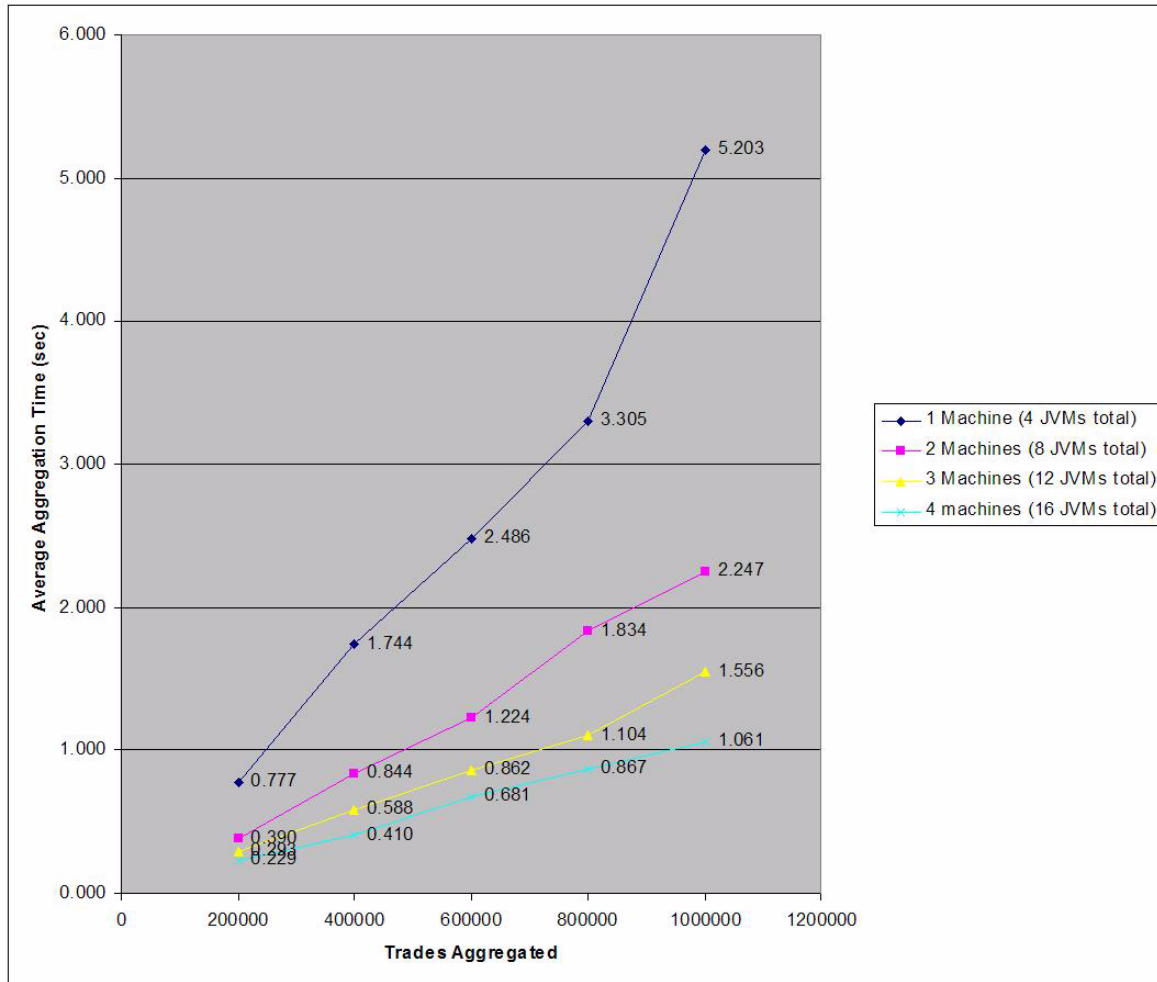
The JDK used on both the client and the servers was Java 2 Runtime Environment, Standard Edition (build 1.5.0_05-84)

The Results

As you can see in the following graph the average aggregation time for the aggregations decreases **linearly** as more cache servers/machines are added to the data grid!

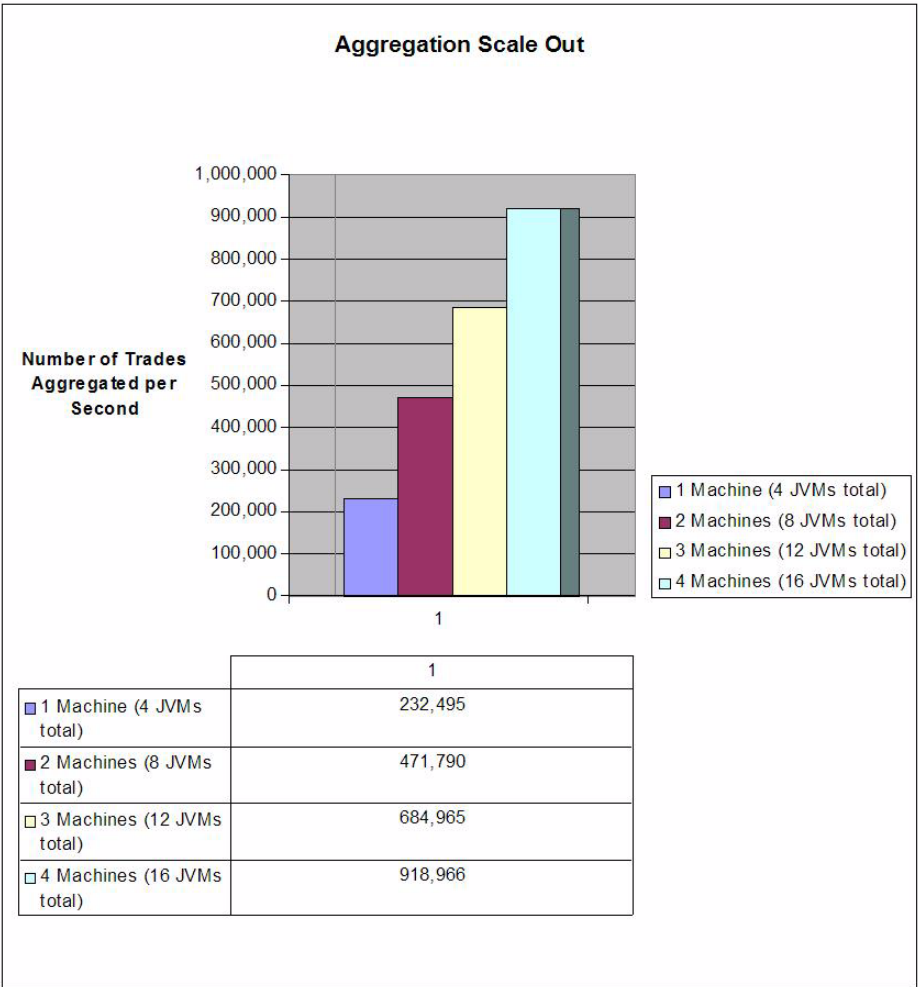
Note: The lowest and highest times were not used in the calculations below resulting in a data set of eighteen results used to create an average.

Figure 48–1 Average Aggregation Time



Similarly, the following graph illustrates how the aggregations per second scales **linearly** as you add more machines! When moving from 1 machine to 2 machines the trades aggregated per second **double**, when moving from 2 machines to 4 machines the trades aggregated per second **double again**.

Figure 48-2 Aggregation Scale-Out



Note: FAILOVER!

The above aggregations will complete *successfully* and *correctly* even if one of the cache servers *or* an *entire machine* fails *during* the aggregation!

Conclusion

Combining the Coherence data grid (that is, partitioned cache) with the InvocableMap features enables:

- Applications to scale out data grid calculations **linearly**;
- Groups to meet increasingly aggressive SLAs by dynamically/transparently adding more resources to the data grid. That is, if you need to achieve **1,837,932** trade aggregations per second all that is required is to start 16 more cache servers across four more machines.

Operational Configuration Elements

This section describes the elements that control the operational and runtime settings used by Oracle Coherence. These settings are used to create, configure and maintain Coherence clustering, communication, and data management services. This section also describes the deployment descriptor files in which these elements can appear.

Operational Configuration Deployment Descriptors

The elements that control the operational and runtime settings to create and configure clustering, communication, and data management services can be specified in either of two deployment descriptors.

The `tangosol-coherence.xml` descriptor is where you specify the operational and runtime elements that control clustering, communication, and data management services. The optional `tangosol-coherence-override.xml` override file is where you specify only the subset of the operational descriptor which you want to adjust. See ["Operational Override File \(tangosol-coherence-override.xml\)"](#) on page A-2 for more information.

For information on configuring caches see [Appendix B, "Cache Configuration Elements."](#)

Document Location

When deploying Coherence, it is important to make sure that the `tangosol-coherence.xml` descriptor is present and situated in the application classpath (like with any other resource, Coherence will use the first one it finds in the classpath). By default (as Oracle ships the software) `tangosol-coherence.xml` is packaged into in the `coherence.jar`.

Document Root

The root element of the operational descriptor is `<coherence>` and is where a cluster and services are configured.

Document Format

Coherence Operational Configuration deployment descriptor should begin with the following DOCTYPE declaration:

Example A-1 Operational Configuration Deployment Descriptor DOCTYPE Declaration

```
<!DOCTYPE coherence SYSTEM "coherence.dtd">
```

Note: When deploying Coherence into environments where the default character set is EBCDIC rather than ASCII, please make sure that this descriptor file is in ASCII format and is deployed into its runtime environment in the binary format.

Operational Override File (tangosol-coherence-override.xml)

Though it is acceptable to supply an alternate definition of the default `tangosol-coherence.xml` file, the preferred approach to operational configuration is to specify an override file. The override file contains only the subset of the operational descriptor which you want to adjust. The default name for the override file is `tangosol-coherence-override.xml`, and the first instance found in the classpath will be used. The format of the override file is the same as for the operational descriptor, except that all elements are optional, any missing element will simply be loaded from the operational descriptor.

Multiple levels of override files may also be configured, allowing for additional fine tuning between similar deployment environments such as staging and production. For example, this feature is used to provide alternate configurations, such as the logging verbosity, based on the deployment type (evaluation, development, production). For more information on logging verbosity, see the `<severity-level>` subelement in ["logging-config"](#) on page A-29. See also the `tangosol-coherence-override-eval.xml`, `tangosol-coherence-override-dev.xml`, and `tangosol-coherence-override-prod.xml` files, within `coherence.jar`.

Note: It is recommended that you supply an override file rather than a custom operational descriptor, thus specifying only the settings you want to adjust.

Command Line Override

Oracle Coherence provides a very powerful command line override feature which allows any element defined in this descriptor to be overridden from the Java command line if it has a system-property attribute defined in the descriptor. This feature enables you to use the same operational descriptor (and override file) across all cluster nodes and customize each node using the system properties. See [Appendix C, "Command Line Overrides"](#) for more information on this feature.

Element Index

Table A–1 lists all non-terminal elements which may be used from within the operational configuration.

Table A–1 Non-Terminal Operational Configuration Elements

Element	Used in:
access-controller	security-config
address-provider	well-known-addresses
authorized-hosts	cluster-config
cache-factory-builder-config	coherence
callback-handler	security-config
cluster-config	coherence
cluster-quorum-policy	cluster-config
coherence	<i>root element</i>
configurable-cache-factory-config	coherence
filters	cluster-config
flow-control	packet-delivery
host-range	authorized-hosts
identity-asserter	security-config
identity-manager	ssl
identity-transformer	security-config
incoming-message-handler	cluster-config
init-param	init-params
init-params	access-controller, address-provider, callback-handler, configurable-cache-factory-config, filters, services
instance	socket-provider, service-failure-policy
key-store	identity-manager, trust-manager
license-config	coherence
logging-config	coherence
management-config	coherence
mbean	mbeans
mbeans	management-config
mbean-filter	management-config
member-identity	cluster-config
multicast-listener	cluster-config
notification-queueing	packet-publisher
outgoing-message-handler	cluster-config
outstanding-packets	flow-control
packet-buffer	multicast-listener, packet-publisher, unicast-listener

Table A–1 (Cont.) Non-Terminal Operational Configuration Elements

Element	Used in:
packet-bundling	packet-delivery
packet-delivery	packet-publisher
packet-pool	incoming-message-handler, packet-publisher
packet-publisher	cluster-config
packet-size	packet-publisher
packet-speaker	cluster-config
pause-detection	flow-control
provider	ssl, identity-manager, trust-manager
reporter	management-config
security-config	coherence
serializers	cluster-config
service-guardian	cluster-config
services	cluster-config
shutdown-listener	cluster-config
socket-address	well-known-addresses
socket-provider	socket-providers, unicast-listener
socket-providers	cluster-config
ssl	socket-provider
tcp-ring-listener	cluster-config
traffic-jam	packet-publisher
trust-manager	ssl
unicast-listener	cluster-config
volume-threshold	packet-speaker
well-known-addresses	unicast-listener

access-controller

Used in: [security-config](#).

[Table A-2](#) describes the subelements you can define within the `access-controller` element.

Table A-2 *access-controller Subelements*

Element	Required/ Optional	Description
<code><class-name></code>	Required	Specifies the name of a Java class that implements <code>com.tangosol.net.security.AccessController</code> interface, which will be used by the security framework to check access rights for clustered resources and encrypt/decrypt node-to-node communications regarding those rights. See Chapter 30, "Securing Coherence" for more information. Default value is <code>com.tangosol.net.security.DefaultController</code> .
<code><init-params></code>	Optional	<p>Contains one or more initialization parameter(s) for a class that implements the <code>AccessController</code> interface. For the default <code>AccessController</code> implementation the parameters are the paths to the key store file and permissions description file, specified as follows:</p> <pre> <init-params> <init-param id="1"> <param-type>java.io.File</param-type> <param-value system-property="tangosol.coherence.security.keystore"></param-value> </init-param> <init-param id="2"> <param-type>java.io.File</param-type> <param-value system-property="tangosol.coherence.security.permissions"></param-value> </init-param> </init-params> </pre> <p>Preconfigured overrides based on the default <code>AccessController</code> implementation and the default parameters as specified above are <code>tangosol.coherence.security.keystore</code> and <code>tangosol.coherence.security.permissions</code>. For more information on preconfigured overrides, see Appendix C, "Command Line Overrides." For more information on the elements you can define within the <code>init-param</code> element, see "init-param" on page A-24.</p>

address-provider

Used in: [well-known-addresses](#)

Description

Contains the configuration information for an address factory that implements the `com.tangosol.net.AddressProvider` interface.

Elements

[Table A-3](#) describes the subelements you can define within the `address-provider` element.

Table A-3 *address-provider Subelements*

Element	Required/ Optional	Description
<class-name>	Optional	Specifies the fully qualified name of a class that implements the <code>com.tangosol.net.AddressProvider</code> interface. This element cannot be used together 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 <code>com.tangosol.net.AddressProvider</code> interface. This element cannot be used together with the <class-name> element and is used together with the <method-name> element.
<method-name>	Optional	Specifies the name of a static factory method on the factory class which will perform object instantiation.
<init-params>	Optional	Specifies initialization parameters which are accessible by implementations which support the <code>com.tangosol.run.xml.XmlConfigurable</code> interface, or which include a public constructor with a matching signature. Initialization parameters can be specified for both the <class-name> element and the <class-factory-name> element.

authorized-hosts

Used in: [cluster-config](#).

Description

If specified, restricts cluster membership to the cluster nodes specified in the collection of unicast addresses, or address range. The unicast address is the address value from the authorized cluster nodes' [unicast-listener](#) element. Any number of `host-address` and `host-range` elements may be specified.

Elements

[Table A-4](#) describes the subelements you can define within the `authorized-hosts` element.

Table A-4 *authorized-hosts Subelements*

Element	Required/ Optional	Description
<code><host-address></code>	Optional	Specifies an IP address or hostname. If any are specified, only hosts with specified host-addresses or within the specified host-ranges will be allowed to join the cluster. The content override attributes <code>id</code> can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document.
<code><host-range></code>	Optional	Specifies a range of IP addresses. If any are specified, only hosts with specified host-addresses or within the specified host-ranges will be allowed to join the cluster. The content override attributes <code>id</code> can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document.
<code><host-filter></code>	Optional	Specifies class configuration information for a <code>com.tangosol.util.Filter</code> implementation that is used by the cluster to determine whether to accept a new cluster member. The <code>evaluate()</code> method will be passed the <code>java.net.InetAddress</code> of the client. Implementations should return <code>true</code> to allow the new member to join the cluster.

The content override attributes `xml-override` and `id` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "[Element Attributes](#)" on page A-88.

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 will use a custom `ConfigurableCacheFactory` implementation from two classloaders.

```
CacheFactoryBuilder cfb = CacheFactory.getCacheFactoryBuilder();

//load the first configuration
cfb.getConfigurableCacheFactory("example-config.xml", loader0);
CacheFactory.ensureCluster();
NamedCache cache = CacheFactory.getCache("dist-example");

//load the second configuration
cfb.getConfigurableCacheFactory("example-config1.xml", loader1);
CacheFactory.ensureCluster();
NamedCache cache1 = CacheFactory.getCache("dist-example1");
```

Elements

[Table A-5](#) describes the elements you can define within the `cache-factory-builder-config` element.

Table A-5 *cache-factory-builder-config Subelements*

Element	Required/ Optional	Description
<class-name>	Required	Specifies the name of a Java class that implements the <code>com.tangosol.net.CacheFactoryBuilder</code> interface. Default value is <code>com.tangosol.net.DefaultCacheFactoryBuilder</code> .
<init-params>	Optional	Contains initialization parameters for the cache factory builder implementation.

callback-handler

Used in: [security-config](#).

[Table A-6](#) describes the elements you can define within the `callback-handler` element.

Table A-6 *callback-handler Subelement*

Element	Required/ Optional	Description
<code><class-name></code>	Required	Specifies the name of a Java class that provides the implementation for the <code>javax.security.auth.callback.CallbackHandler</code> interface.
<code><init-params></code>	Optional	Contains one or more initialization parameter(s) for a <code>CallbackHandler</code> implementation.

cluster-config

Used in: [<coherence>](#)

Description

Contains the cluster configuration information, including communication and service parameters.

Elements

[Table A-7](#) describes the subelements you can define within the `cluster-config` element.

Table A-7 *cluster-config Subelements*

Element	Required/ Optional	Description
<member-identity>	Optional	Specifies detailed identity information that is useful for defining the location and role of the cluster member.
<unicast-listener>	Required	Specifies the configuration information for the Unicast listener, used for receiving point-to-point network communications.
<multicast-listener>	Required	Specifies the configuration information for the Multicast listener, used for receiving point-to-multipoint network communications.
<tcp-ring-listener>	Required	Specifies configuration information for the TCP Ring listener, used to death detection.
<shutdown-listener>	Required	Specifies the action to take upon receiving an external shutdown request.
<service-guardian>	Required	Specifies the configuration information for the service guardians, used for detecting and resolving service deadlock.
<packet-speaker>	Required	Specifies configuration information for the Packet speaker, used for network data transmission.
<packet-publisher>	Required	Specifies configuration information for the Packet publisher, used for managing network data transmission.
<incoming-message-handler>	Required	Specifies configuration information for the Incoming message handler, used for dispatching incoming cluster communications.
<outgoing-message-handler>	Required	Specifies configuration information for the Outgoing message handler, used for dispatching outgoing cluster communications.
<authorized-hosts>	Optional	Specifies the hosts which are allowed to join the cluster.
<services>	Required	Specifies the declarative data for all available Coherence services.
<filters>	Optional	Specifies data transformation filters, which can be used to perform custom transformations on data being transferred between cluster nodes.

Table A–7 (Cont.) cluster-config Subelements

Element	Required/ Optional	Description
<serializers>	Optional	Specifies any number of serializer class configurations that implement <code>com.tangosol.io.Serializer</code> .
<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.

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "[Element Attributes](#)" on page A-88 for more information on this attribute.

cluster-quorum-policy

Used in: [<cluster-config>](#)

Description

The `cluster-quorum-policy` element contains quorum policy settings for the Cluster service.

Element

[Table A-8](#) describes the subelements you can define within the `cluster-quorum-policy` element.

Table A-8 *cluster-quorum-policy-scheme Subelements*

Element	Required/ Optional	Description
<code><timeout-survivor-quorum></code>	Optional	<p>Specifies the minimum number of cluster members that must remain in order to terminate one or more cluster members due to a detected network timeout, irrespective of the root cause. Use the <code>role</code> attribute to specify this value for cluster members of a given role (as defined in the <code><role-name></code> element). For example:</p> <pre><timeout-survivor-quorum role="Server">50 </timeout-survivor-quorum></pre> <p>The value must be a non-negative integer.</p>
<code><class-name></code>	Optional	<p>Specifies a class that provides custom quorum policies. This element cannot be used together with the <code><timeout-survivor-quorum></code> or the <code><class-factory-name></code> element.</p> <p>The class must implement the <code>com.tangosol.net.ActionPolicy</code> interface. Initialization parameters can be specified using the <code><init-params></code> element.</p>
<code><class-factory-name></code>	Optional	<p>Specifies a factory class for creating custom action policy instances. This element cannot be used together with the <code><timeout-survivor-quorum></code> or <code><class-name></code> elements.</p> <p>This element is used together with the <code><method-name></code> element. The action policy instances must implement the <code>com.tangosol.net.ActionPolicy</code> interface. In addition, initialization parameters can be specified using the <code><init-params></code> element.</p>

coherence

root element

Description

The coherence element is the root element of the operational deployment descriptor `tangosol-coherence.xml`.

Elements

[Table A-9](#) describes the elements you can define within the coherence element.

Table A-9 *coherence Subelements*

Element	Required/ Optional	Description
<code><cluster-config></code>	Required	Contains the cluster configuration information. This element is where most communication and service parameters are defined.
<code><logging-config></code>	Required	Contains the configuration information for the logging facility.
<code><configurable-cache-factory-config></code>	Required	Contains configuration information for the configurable cache factory, which controls from where and how the cache configuration settings are loaded.
<code><cache-factory-builder-config></code>	Required	Contains the configuration information for a cache factory builder, which allows building and managing multiple cache factory configurations across multiple class loaders.
<code><management-config></code>	Required	Contains the configuration information for the coherence Management Framework. See Chapter 34, "How to Manage Coherence Using JMX" for more information.
<code><security-config></code>	Optional	Contains the configuration information for the Coherence Security Framework.
<code><license-config></code>	Optional	Contains the edition and operational mode configuration.

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See ["Element Attributes"](#) on page A-88 for more information on this attribute

configurable-cache-factory-config

Used in: [coherence](#)

Description

The `configurable-cache-factory-config` element contains the configuration information for constructing an instance of the `com.tangosol.net.ConfigurableCacheFactory` interface. The default implementation is the `com.tangosol.net.DefaultConfigurableCacheFactory` class.

Using a custom `ConfigurableCacheFactory` implementation is an advanced use case and is typically used to allow applications that are scoped by different class loaders to use separate cache configuration files (as is the case with JavaEE and OSGI). Typically, the `DefaultConfigurableCacheFactory` class is extended for such use cases.

The following example loads two configuration files which contain different cache definitions and use different `ClassLoaders`.

```
//load the first configuration and use a cache

ConfigurableCacheFactory dccf= new
    DefaultConfigurableCacheFactory("example-config.xml", loader0);
NamedCache cache = dccf.ensureCache("dist-example", loader0);
cache.put(key, value);

//load the second cache configuration and use a cache

ConfigurableCacheFactory dccf1= new
    DefaultConfigurableCacheFactory("example-config1.xml", loader1);
NamedCache cache1 = dccf1.ensureCache("dist-example1", loader1);
cache1.put(key, value);
```

Note: This example requires each cache definition to use a different service name; otherwise, an exception is thrown indicating that the service was started by a factory with a different configuration descriptor.

Elements

[Table A-10](#) describes the elements you can define within the `configurable-cache-factory-config` element.

Table A-10 *configurable-cache-factory-config Subelements*

Element	Required/ Optional	Description
<class-name>	Required	Specifies the name of a Java class that implements the <code>com.tangosol.net.ConfigurableCacheFactory</code> interface. Default value is <code>com.tangosol.net.DefaultConfigurableCacheFactory</code> .
<init-params>	Optional	<p>Contains initialization parameters for the cache configuration factory implementation. For the default cache configuration factory class, a single parameter is used as follows:</p> <pre> <init-param> <param-type>java.lang.String</param-type> <param-value>coherence-cache-config.xml</param-value> </init-param> </pre> <p>Unless an absolute or relative path is specified, such as with <code>./path/to/config.xml</code>, the application's classpath will be used to find the specified descriptor. Preconfigured is <code>tangosol.coherence.cacheconfig</code>. See Appendix C, "Command Line Overrides" for more information on overrides.</p>

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See ["Element Attributes"](#) on page A-88 for more information on this attribute.

filters

Used in: [cluster-config](#).

Description

Data transformation filters can be used by [services](#) to apply a custom transformation on data being transferred between cluster nodes. This can be used for instance to compress or encrypt Coherence network traffic. See the `<filter-class>` element for more information.

Implementation

Data transformation filters are implementations of the `com.tangosol.util WrapperStreamFactory` interface.

Note: Data transformation filters are not related to `com.tangosol.util.Filter`, which is part of the Coherence API for querying caches.

Elements

[Table A-11](#) describes the elements you can define within each `filters` element.

Table A-11 *filters Subelements*

Element	Required/ Optional	Description
<code><filter-name></code>	Required	Specifies the canonical name of the filter. This name is unique within the cluster. For example: <code>gzip</code> . The content override attribute <code>id</code> can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document.
<code><filter-class></code>	Required	Specifies the class name of the filter implementation. This class must have a zero-parameter public constructor and must implement the <code>com.tangosol.util WrapperStreamFactory</code> interface.
<code><init-params></code>	Optional	<p>Specifies initialization parameters, for configuring filters which implement the <code>com.tangosol.run.xml.XmlConfigurable</code> interface. For example when using a <code>com.tangosol.net.CompressionFilter</code> the parameters are specified as follows:</p> <pre> <init-param> <param-name>strategy</param-name> <param-value>gzip</param-value> </init-param> <init-param> <param-name>level</param-name> <param-value>default</param-value> </init-param> </pre> <p>For more information on the parameter values for the standard filters refer to, refer to Chapter 9, "Using Network Filters."</p>

The content override attributes `id` and `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See ["Element Attributes"](#) on page A-88 for more information on these attributes.

Compression Filter Parameters

The compression `filters` (`com.tangosol.net.CompressionFilter`), supports the parameters described in [Table A-12](#) (see `java.util.zip.Deflater` for details).

Table A-12 *Compression Filter Parameters*

Parameter Name	Value Description
<code>buffer-length</code>	Specifies compression buffer length in bytes. Legal values are positive integers or zero. Default value is 0.
<code>level</code>	Specifies the compression level. Legal values are: <ul style="list-style-type: none">▪ <code>default</code>▪ <code>compression</code>▪ <code>speed</code>▪ <code>none</code> Default value is <code>default</code> .
<code>strategy</code>	Specifies the compressions strategy. Legal values are: <ul style="list-style-type: none">▪ <code>gzip</code>▪ <code>huffman-only</code>▪ <code>filtered</code>▪ <code>default</code> Default value is <code>gzip</code> .

flow-control

Used in: [packet-delivery](#).

Description

The flow-control element contains configuration information related to packet throttling and remote GC detection.

Remote GC Detection

Remote Pause detection allows Coherence to detect and react to a cluster node becoming unresponsive (likely due to a long GC). When a node is marked as paused, packets addressed to it will be sent at a lower rate until the node resumes responding. This remote GC detection is used to avoid flooding a node while it is incapable of responding.

Packet Throttling

Flow control allows Coherence to dynamically adjust the rate at which packets are transmitted to a given cluster node based on point to point transmission statistics.

Elements

[Table A-13](#) describes the elements you can define within the `flow-control` subelement.

Table A-13 *flow-control Subelements*

Element	Required/ Optional	Description
<code><enabled></code>	Optional	Specifies if flow control is enabled. Default is true
<code><pause-detection></code>	Optional	Defines the number of packets that will be resent to an unresponsive cluster node before assuming that the node is paused.
<code><outstanding-packets></code>	Optional	Defines the number of unconfirmed packets that will be sent to a cluster node before packets addressed to that node will be deferred.

host-range

Used in: [authorized-hosts](#).

Description

Specifies a range of unicast addresses of nodes which are allowed to join the cluster.

Elements

[Table A-14](#) describes the elements you can define within each `host-range` element.

Table A-14 *host-range Subelements*

Element	Required/ Optional	Description
<code><from-address></code>	Required	Specifies the starting IP address for a range of host addresses. For example: 198.168.1.1.
<code><to-address></code>	Required	Specifies to-address element specifies the ending IP address (inclusive) for a range of hosts. For example: 198.168.2.255.

The content override attribute `id` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "[Element Attributes](#)" on page A-88 for more information on this attribute.

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 in order to establish a user's identity and is used on a Coherence*Extend proxy server. The identity asserter is used together 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-15](#) describes the elements you can define within the `<identity-asserter>` element.

Table A-15 *identity-asserter Subelements*

Element	Required/ Optional	Description
<code><class-name></code>	Optional	Specifies a class that implements <code>com.tangosol.net.security.IdentityAsserter</code> . This element cannot be used together with the <code><class-factory-name></code> element.
<code><class-factory-name></code>	Optional	Specifies a factory class for creating asserter instances. The instances must implement <code>com.tangosol.net.security.IdentityAsserter</code> . This element cannot be used together with the <code><class-name></code> element. This element can be used together with the <code><method-name></code> element.
<code><method-name></code>	Optional	Specifies the name of a static factory method on the factory class which will perform object instantiation.
<code><init-params></code>	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 is unable to present authentication credentials.

Elements

[Table A-16](#) describes the elements you can define within the `identity-manager` element.

Table A-16 *identity-manager Subelements*

Element	Required/ Optional	Description
<code><algorithm></code>	Optional	Specifies the algorithm used by the identity manager. The default value is <code>SunX509</code> .
<code><provider></code>	Optional	Specifies the configuration for a security provider instance.
<code><key-store></code>	Optional	Specifies the configuration for a key store implementation.
<code><password></code>	Required	Specifies the private key password.

identity-transformer

Used in: [security-config](#)

Description

The `<identity-transformer>` element contains the configuration information for a class that implements the `com.tangosol.net.security.IdentityTransformer` interface. The class is called to transform a Subject (Principal in .NET) to a token that asserts identity and is used on a Coherence*Extend client. The identity transformer is used together with an identityasserter (used on a Coherence*Extend proxy server) to ensure that only valid clients are allowed to connect to an extend proxy.

Elements

[Table A-17](#) describes the elements you can define within the `<identity-transformer>` element.

Table A-17 *identity-transformer Subelements*

Element	Required/ Optional	Description
<code><class-name></code>	Optional	Specifies a class that implements <code>com.tangosol.net.security.IdentityTransformer</code> . This element cannot be used together with the <code><class-factory-name></code> element.
<code><class-factory-name></code>	Optional	Specifies a factory class for creating asserter instances. The instances must implement <code>com.tangosol.net.security.IdentityTransformer</code> . This element cannot be used together with the <code><class-name></code> element. This element can be used together with the <code><method-name></code> element.
<code><method-name></code>	Optional	Specifies the name of a static factory method on the factory class which will perform object instantiation.
<code><init-params></code>	Optional	Contains class initialization parameters for the transformer implementation.

incoming-message-handler

Used in: [cluster-config](#).

Description

The `incoming-message-handler` assembles UDP packets into logical messages and dispatches them to the appropriate Coherence service for processing.

Elements

[Table A-18](#) describes the subelements you can define within the `incoming-message-handler` element.

Table A-18 *incoming-message-handler Subelements*

Element	Required/ Optional	Description
<code><maximum-time-variance></code>	Required	Specifies the maximum time variance between sending and receiving broadcast Messages when trying to determine the difference between a new cluster Member's system time and the cluster time. The smaller the variance, the more certain one can be that the cluster time will be closer between multiple systems running in the cluster; however, the process of joining the cluster will be extended until an exchange of Messages can occur within the specified variance. Normally, a value as small as 20 milliseconds is sufficient, but with heavily loaded clusters and multiple network hops it is possible that a larger value would be necessary. Default value is 16.
<code><use-nack-packets></code>	Required	Specifies whether the packet receiver will use negative acknowledgments (packet requests) to pro-actively respond to known missing packets. See " notification-queueing " on page A-45 for additional details and configuration. Legal values are <code>true</code> or <code>false</code> . Default value is <code>true</code> .
<code><priority></code>	Required	Specifies a priority of the incoming message handler execution thread. Legal values are from 1 to 10. Default value is 7.
<code><packet-pool></code>	Required	Specifies how many incoming packets Coherence will buffer before blocking.

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "[Element Attributes](#)" on page A-88 for more information on this attribute.

init-param

Used in: [init-params](#).

Description

Defines an individual initialization parameter.

Elements

[Table A-19](#) describes the elements you can define within the `init-param` element.

Table A-19 *init-param* Subelement

Element	Required/ Optional	Description
<code><param-name></code>	Optional	Specifies the name of the parameter passed to the class. The <code>param-type</code> or <code>param-name</code> must be specified. For example: <code>thread-count</code> . For more information on the pre-defined parameter values available for the specific elements, refer to Initialization Parameter Settings .
<code><param-type></code>	Optional	Specifies the data type of the parameter passed to the class. The <code>param-type</code> or <code>param-name</code> must be specified. For example: <code>int</code>
<code><param-value></code>	Required	Specifies the value passed in the parameter. For example: <code>8</code> . For more information on the pre-defined parameter values available for the specific elements, refer to Initialization Parameter Settings .

The content override attribute `id` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "[Element Attributes](#)" on page A-88 for more information no this attribute.

init-params

Used in: [address-provider](#), [filters](#), [services](#), [configurable-cache-factory-config](#), [access-controller](#), and [callback-handler](#).

Description

Defines a series of initialization parameters.

Elements

[Table A-20](#) describes the elements you can define within the `init-params` element.

Table A-20 *init-params* Subelement

Element	Required/ Optional	Description
<code><init-param></code>	Optional	Defines an individual initialization parameter.

instance

Used in: [socket-provider](#), [service-failure-policy](#)

Description

The `<instance>` element contains the configuration of an implementation class or class factory that is used to plug in custom functionality.

Elements

[Table A-21](#) describes the elements you can define within the `instance` element.

Table A-21 *instance Subelements*

Element	Required/ Optional	Description
<code><class-name></code>	Optional	Specifies the fully qualified name of an implementation class. This element cannot be used together with the <code><class-factory-name></code> element.
<code><class-factory-name></code>	Optional	Specifies the fully qualified name of a factory class for creating implementation class instances. This element cannot be used together with the <code><class-name></code> element and is used together with the <code><method-name></code> element.
<code><method-name></code>	Optional	Specifies the name of a static factory method on the factory class which will perform object instantiation.
<code><init-params></code>	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 A-22](#) describes the elements you can define within the `key-store` element.

Table A-22 *key-store Subelements*

Element	Required/ Optional	Description
<code><url></code>	Required	Specifies the Uniform Resource Locator (URL) to a key store.
<code><password></code>	Optional	Specifies the password for the key store.
<code><type></code>	Optional	Specifies the type of a <code>java.security.KeyStore</code> instance. The default value is <code>JKS</code> .

license-config

Used in: [coherence](#).

[Table A-23](#) describes the elements you can define within the `license-config` element.

Table A-23 *license-config Subelements*

Element	Required/ Optional	Description
<code><edition-name></code>	Optional	Specifies the product edition that the member will use. This allows multiple product editions to be used within the same cluster, with each member specifying the edition that it will be using. Valid values are: GE (Grid Edition), EE (Enterprise Edition), SE (Standard Edition), RTC (Real-Time Client), DC (Data Client). Default value is GE.
<code><license-mode></code>	Optional	Specifies whether the product is being used in an development or production mode. Valid values are <code>prod</code> (Production), and <code>dev</code> (Development). Note: This value cannot be overridden in <code>tangosol-coherence-override.xml</code> . It must be specified in <code>tangosol-coherence.xml</code> or (preferably) supplied as system property <code>tangosol.coherence.mode</code> on the Java command line. Default value is <code>dev</code> .

logging-config

Used in: [coherence](#).

Elements

The following table describes the elements you can define within the logging-config element.

Table A–24 *logging-config Subelements*

Element	Required/ Optional	Description
<destination>	Required	<p>Specifies the output device used by the logging system. Legal values are:</p> <ul style="list-style-type: none"> ▪ <code>stdout</code> ▪ <code>stderr</code> (default) ▪ <code>jdk</code> ▪ <code>log4j</code> ▪ <i>a file name</i> <p>If <code>jdk</code> is specified as the destination, Coherence must be run using JDK 1.5 or later. If <code>log4j</code> is specified, the Log4j libraries must be in the classpath. In both cases, the appropriate logging configuration mechanism (system properties, property files, and so on) are necessary to configure the JDK/Log4j logging libraries. Preconfigured override is <code>tangosol.coherence.log</code>. See Appendix C, "Command Line Overrides" for more information.</p>
<logger-name>	Optional	<p>Specifies a logger name within chosen logging system that should be used to log Coherence related messages. This value is only used by the JDK and log4j logging systems.</p> <p>Default value is <code>Coherence</code>.</p> <p>Preconfigured override is <code>tangosol.coherence.log.logger</code>. See Appendix C, "Command Line Overrides" for more information.</p>

Table A–24 (Cont.) logging-config Subelements

Element	Required/ Optional	Description
<severity-level>	Required	<p>Specifies which logged messages will be output to the log destination. Legal values are:</p> <ul style="list-style-type: none"> 0—only output without a logging severity level specified will be logged 1—all the above plus errors 2—all the above plus warnings 3—all the above plus informational messages 4-9—all the above plus internal debugging messages (the higher the number, the more the messages) -1—no messages <p>Default value is 3. Preconfigured override is <code>tangosol.coherence.log.level</code>. See Appendix C, "Command Line Overrides" for more information.</p>
<message-format>	Required	<p>Specifies how messages that have a logging level specified will be formatted before passing them to the log destination. The value of the message-format element is static text with the following replaceable parameters:</p> <ul style="list-style-type: none"> {date}—the date/time format (to a millisecond) at which the message was logged {version}—the Oracle Coherence exact version and build details {level}—the logging severity level of the message {thread}—the thread name that logged the message {member}—the cluster member id (if the cluster is currently running) {location}—the fully qualified cluster member id: cluster-name, site-name, rack-name, machine-name, process-name and member-name (if the cluster is currently running) {role}—the specified role of the cluster member {text}—the text of the message <p>Default value is:</p> <pre>{date} Oracle Coherence {version} <{level}> (thread={thread}, member={member}): {text}</pre>

Table A–24 (Cont.) logging-config Subelements

Element	Required/ Optional	Description
<character-limit>	Required	<p>Specifies the maximum number of characters that the logger daemon will process from the message queue before discarding all remaining messages in the queue. Note that the message that caused the total number of characters to exceed the maximum will NOT be truncated, and all messages that are discarded will be summarized by the logging system with a single log entry detailing the number of messages that were discarded and their total size. The truncation of the logging is only temporary, since when the queue is processed (emptied), the logger is reset so that subsequent messages will be logged.</p> <p>The purpose of this setting is to avoid a situation where logging can itself prevent recovery from a failing condition. For example, with tight timings, logging can actually change the timings, causing more failures and probably more logging, which becomes a vicious cycle. A limit on the logging being done at any one point in time is a "pressure valve" that prevents such a vicious cycle from occurring. Note that logging occurs on a dedicated low-priority thread to even further reduce its impact on the critical portions of the system.</p> <p>Legal values are positive integers or zero. Zero implies no limit.</p> <p>Default value in production mode is 4096 and 2147483647 in development mode. Preconfigured override is <code>tangosol.coherence.log.limit</code>. For more information, see Appendix C, "Command Line Overrides"</p>

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See ["Element Attributes"](#) on page A-88 for more information on this attribute.

management-config

Used in: [coherence](#).

Elements

[Table A-25](#) describes the elements you can define within the management-config element.

Table A-25 *management-config Subelements*

Element	Optional/ Required	Description
<managed-nodes>	Required	<p>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:</p> <ul style="list-style-type: none"> ■ none—No MBean server is instantiated. ■ local-only—Manage only MBeans which are local to the cluster node (that is, within the same JVM). ■ remote-only—Manage MBeans on other remotely manageable cluster nodes. See <allowed-remote-management> subelement. Requires Coherence Enterprise Edition or higher ■ all—Manage both local and remotely manageable cluster nodes. See <allowed-remote-management> subelement. Requires Coherence Enterprise Edition or higher. <p>Default value is none. Preconfigured override is <code>tangosol.coherence.management</code>. See Appendix C, "Command Line Overrides" for more information.</p>
<allow-remote-management>	Required	<p>Specifies whether this cluster node exposes its managed objects to remote MBean server(s). Legal values are: <code>true</code> or <code>false</code>. Default value is <code>true</code>. Preconfigured override is <code>tangosol.coherence.management.remote</code>. See Appendix C, "Command Line Overrides" for more information.</p>
<refresh-policy>	Optional	<p>Specifies the method which is used to refresh remote management information.</p> <p>Legal values are: <code>refresh-ahead</code>, <code>refresh-behind</code> or <code>refresh-expired</code>.</p> <p>Default value is <code>refresh-ahead</code>.</p> <p>Preconfigured override is <code>tangosol.coherence.management.refresh.policy</code></p>
<refresh-expiry>	Optional	<p>Specifies the time interval (in milliseconds) after which a remote MBean information will be invalidated on the management node.</p> <p>Legal values are strings representing time intervals.</p> <p>Default value is <code>1s</code>.</p> <p>Preconfigured override is <code>tangosol.coherence.management.refresh.expiry</code></p>

Table A–25 (Cont.) management-config Subelements

Element	Optional/ Required	Description
<code><refresh-timeout></code>	Optional	<p>Specifies the duration which the management node will wait for a response from a remote node when refreshing MBean information. This value must be less than the refresh-expiry interval.</p> <p>Legal values are strings representing time intervals.</p> <p>Default value is 250ms.</p> <p>Preconfigured override is <code>tangosol.coherence.management.refresh.timeout</code></p>
<code><read-only></code>	Optional	<p>Specifies whether the managed objects exposed by this cluster node allow operations that modify run-time attributes. Legal values are: true or false. Default value is false. Preconfigured override is <code>tangosol.coherence.management.readonly</code>. See Appendix C, "Command Line Overrides"</p>
<code><default-domain-name></code>	Optional	<p>Specifies the default domain name for the MBean server that is used to register MBeans exposed by the Coherence management framework. This value is only used by the cluster nodes that have an in-process MBean server and allow management of local or other node's managed objects. If this value is not specified, the first existing MBean server is used.</p> <p>This element is also used when implementing the <code>MBeanServerFinder</code> interface. See the <code><service-factory></code> element below.</p>
<code><service-name></code>	Optional	<p>Specifies the name of the Invocation Service used for remote management. This element is used only if <code>allow-remote-management</code> is set to true.</p>
<code><service-factory></code>	Optional	<p>Contains the configuration information for the <code>MBeanServer</code> factory that implements the <code>com.tangosol.net.management.MBeanServerFinder</code> interface, which is used to find an MBean server that should be used by the Coherence JMX framework to register new or locate existing MBeans. The class name is entered using the <code><class-name></code> subelement and supports initialization parameters using the <code><init-params></code> element.</p>
<code><mbeans></code>	Optional	<p>Contains a list of MBeans to be registered when a node joins the cluster.</p>
<code><mbean-filter></code>	Optional	<p>Contains the configuration information of a filter class that is used to filter MBeans before they are registered.</p>
<code><reporter></code>	Optional	<p>Contains the Reporter's configuration.</p>

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See ["Element Attributes"](#) on page A-88 for more information..

mbean

Used in: [mbeans](#)

Description

The `mbean` element contains a list of elements to be instantiated and registered with the Coherence Management infrastructure.

Elements

[Table A-26](#) describes the subelements you can define within the `mbean` element.

Table A-26 Subelements of `mbean`

Element	Required/ Optional	Description
<code><mbean-class></code>	Optional	<p>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.</p> <p>This element cannot be used together with the <code><mbean-factory></code> element or the <code><mbean-query></code> element.</p>
<code><mbean-factory></code>	Optional	<p>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 together with the <code><mbean-accessor></code> element.</p> <p>This element cannot be used together with the <code><mbean-class></code> element or the <code><mbean-query></code> element.</p>
<code><mbean-query></code>	Optional	<p>Specifies a JMX <code>ObjectName</code> 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 <code>java.lang</code> domain in the Coherence management infrastructure.</p> <pre><mbean-query>java.lang:*/</mbean-query></pre> <p>This element cannot be used together with the <code><mbean-class></code> element or the <code><mbean-factory></code> element.</p>
<code><mbean-accessor></code>	Optional	<p>Specifies the method name on the factory class (specified by the <code><mbean-factory></code> element) that is used to instantiate the MBean.</p>
<code><mbean-name></code>	Required	<p>Specifies the JMX <code>ObjectName</code> prefix for the MBean as it will be registered with the Coherence management framework. The prefix should be a comma-separated <i>Key=Value</i> pair. The Coherence MBean naming convention stipulates that the name should begin with a type/value pair (for example, <code>type=Platform</code>).</p>

Table A–26 (Cont.) Subelements of *mbean*

Element	Required/ Optional	Description
<local-only>	Optional	<p>Specifies whether or not the MBean is visible across the cluster. Valid values are <code>true</code> or <code>false</code>. If set to <code>true</code>, the MBean is registered only with a local MBeanServer and is not accessible by other cluster nodes. If set to <code>false</code>, the <code>nodeId=...</code> key attribute is added to its name and the MBean will be visible from any of the managing nodes (nodes that set the <managed-nodes> element to values of <code>all</code> or <code>remote-only</code>).</p> <p>Default value is <code>false</code>.</p>
<enabled>	Optional	<p>Specifies whether or not the MBean should be instantiated and registered on this instance. Valid values are <code>true</code> or <code>false</code>.</p> <p>Default value is <code>false</code>.</p>
<extend-lifecycle>	Optional	<p>Specifies whether or not the MBean should extend beyond the node connection life cycle. Valid values are <code>true</code> or <code>false</code>. If <code>true</code>, the MBean maintains the statistics and values across connections (coincides with the JVM life cycle). If <code>false</code>, the MBean is destroyed and re-created when a node is disconnected from the grid.</p> <p>Default value is <code>false</code>.</p>

mbeans

Used in: [management-config](#)

Description

The `mbeans` element is the root element for defining custom mbeans and is typically the root element of a custom mbean configuration file. It contains a list of mbean elements to be instantiated and registered with the Coherence management framework.

Elements

[Table A–27](#) describes the elements you can define within the `mbeans` element.

Table A–27 *Subelement of mbeans*

Element	Required/ Optional	Description
<code><mbean></code>	Required	Specifies the MBean type, implementation, and <code>ObjectName</code> that will be 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:

```
<mbean-filter>
  <class-name>com.tangosol.net.management.ObjectNameExcludeFilter</class-name>
  <init-params>
    <init-param>
      <param-type>string</param-type>
      <param-value system-property="tangosol.coherence.management.exclude">
        .*type=Service,name=Management,.*
        .*type=Platform,Domain=java.lang,subType=ClassLoading,.*
        .*type=Platform,Domain=java.lang,subType=Compilation,.*
        .*type=Platform,Domain=java.lang,subType=MemoryManager,.*
        .*type=Platform,Domain=java.lang,subType=Threading,.*
      </param-value>
    </init-param>
  </init-params>
</mbean-filter>
```

Elements

[Table A-43](#) describes the subelements you can define within the `mbean-filter` element.

Table A-28 *mbean-filter Subelements*

Element	Required/ Optional	Description
<code><class-name></code>	Optional	Specifies the name of a filter class for filtering mbeans. This element cannot be used together with the <code><class-factory-name></code> element.
<code><class-factory-name></code>	Optional	Specifies a factory class for creating filter instances. This element cannot be used together with the <code><name></code> element or the <code><class-name></code> element. This element can be used together with the <code><method-name></code> element.
<code><method-name></code>	Optional	Specifies the name of a static factory method on the factory class which will perform object instantiation.
<code><init-params></code>	Optional	Contains class initialization parameters for the filter implementation.

member-identity

Used in: [cluster-config](#).

The `member-identity` element contains detailed identity information that is useful for defining the location and role of the cluster member.

Elements

[Table A-29](#) describes the elements you can define within the `member-identity` element.

Table A-29 *member-identity Subelements*

Element	Required/ Optional	Description
<code><cluster-name></code>	Optional	The <code>cluster-name</code> element contains the name of the cluster. To join the cluster all members must specify the same cluster name. It is strongly suggested that <code>cluster-name</code> be specified for production systems, thus preventing accidental cluster discovery among applications. Preconfigured override is <code>tangosol.coherence.cluster</code> . See Appendix C, "Command Line Overrides" for more information.
<code><site-name></code>	Optional	The <code>site-name</code> element contains the name of the geographic site that the member is hosted at. For WAN clustering, this value identifies the datacenter within which the member is located, and can be used as the basis for intelligent routing, load balancing and disaster recovery planning (that is, the explicit backing up of data on separate geographic sites). The name is also useful for displaying management information (for example, JMX) and interpreting log entries. This element is not currently used to make decisions about data backup location. Preconfigured override is <code>tangosol.coherence.site</code> . See Appendix C, "Command Line Overrides" for more information.
<code><rack-name></code>	Optional	The <code>rack-name</code> element contains the name of the location within a geographic site that the member is hosted at. This is often a cage, rack or blade frame identifier, and can be used as the basis for intelligent routing, load balancing and disaster recovery planning (that is, the explicit backing up of data on separate blade frames). The name is also useful for displaying management information (for example, JMX) and interpreting log entries. This element is not currently used to make decisions about data backup location. Preconfigured override is <code>tangosol.coherence.rack</code> . See Appendix C, "Command Line Overrides" for more information.
<code><machine-name></code>	Optional	The <code>machine-name</code> 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 <code>HOSTNAME</code> , or its name as it appears in a DNS entry). If provided, the <code>machine-name</code> is used as the basis for creating a <code>machine-id</code> , which in turn is used to guarantee that data are backed up on different physical machines to prevent single points of failure (SPOFs). The name is also useful for displaying management information (for example, JMX) and interpreting log entries. It is optional to provide a value for this element. However, it is strongly encouraged that a name always be provided. Preconfigured override is <code>tangosol.coherence.machine</code> . See Appendix C, "Command Line Overrides" for more information.

Table A–29 (Cont.) member-identity Subelements

Element	Required/ Optional	Description
<code><process-name></code>	Optional	The <code>process-name</code> element contains the name of the process (JVM) that the member is hosted on. This name makes it possible to easily differentiate among multiple JVMs running on the same machine. The name is also useful for displaying management information (for example, JMX) and interpreting log entries. It is optional to provide a value for this element. Often, a single member will exist per JVM, and in that situation this name would be redundant. Preconfigured override is <code>tangosol.coherence.process</code> . See Appendix C, "Command Line Overrides" for more information.
<code><member-name></code>	Optional	The <code>member-name</code> element contains the name of the member itself. This name makes it possible to easily differentiate among members, such as when multiple members run on the same machine (or even within the same JVM). The name is also useful for displaying management information (for example, JMX) and interpreting log entries. It is optional to provide a value for this element. However, it is strongly encouraged that a name always be provided. Preconfigured override is <code>tangosol.coherence.member</code> . see Appendix C, "Command Line Overrides" for more information.
<code><role-name></code>	Optional	The <code>role-name</code> element contains the name of the member role. This name allows an application to organize members into specialized roles, such as cache servers and cache clients. The name is also useful for displaying management information (for example, JMX) and interpreting log entries. It is optional to provide a value for this element. However, it is strongly encouraged that a name always be provided. Preconfigured override is <code>tangosol.coherence.role</code> . See Appendix C, "Command Line Overrides" for more information.
<code><priority></code>	Optional	The <code>priority</code> element specifies a priority of the corresponding member. The priority is used as the basis for determining tie-breakers between members. If a condition occurs in which one of two members will be ejected from the cluster, and in the rare case that it is not possible to objectively determine which of the two is at fault and should be ejected, then the member with the lower priority will be ejected. Valid values are from 1 to 10. Preconfigured override is <code>tangosol.coherence.priority</code> . See Appendix C, "Command Line Overrides" for more information.

message-pool

Used in: [outgoing-message-handler](#)

Description

The `<message-pool>` element is used to control how many message buffers are pooled for message transmission. Pooling message buffers relieves the pressure on the JVM garbage collector by pooling the memory resources needed for messaging.

The message pool is comprised of any number of segments of a specified size. For example, a pool with 4 segments and a segment size of 10MB can hold, at most, 40 MB of space for serialization. The number of segments and the segment size are defined using the `<segment>` and `<segment-size>` elements, respectively.

Each pool segment stores message buffers of a specific size. The smallest size buffer is defined by the `<min-buffer-size>` element. The next buffer size for the next segment is then calculated using bitwise left shift using the `<growth-factor>` value (`'min-buffer-size' << growth-factor`). A left shift by n is equivalent to multiplying by 2^n ; where n is the growth factor value. For a growth factor of 2, multiply the minimum buffer size by 4. For a growth factory of 3, multiply the minimum buffer size by 8, and so on.

For example, the default pool values that are shown in [Table A-30](#) results in a message pool were: the first pool segment contains message buffers of 1KB; the second pool segment contains message buffers of 4KB; the third pool segment contains message buffers of 16KB; and the fourth pool segment contains message buffers of 64KB. Using the same default values but increasing the growth factor to 3, results in buffer sizes of 1KB, 8KB, 64KB, and 512KB.

Space that is claimed for network buffers (in and out) and serialization buffers is periodically reclaimed when the capacity is higher than the actual usage.

Elements

[Table A-30](#) describes the elements you can define within the `message-pool` element.

Table A-30 *message-pool Subelements*

Element	Required/ Optional	Description
<code><segments></code>	Optional	Specifies the number of segments used by the message pool to store buffers. Each segment stores buffers of a specific size. The buffer size difference between segments is calculated using the <code><growth-factor></code> element value. Default value is 4.

Table A–30 (Cont.) message-pool Subelements

Element	Required/ Optional	Description
<code><segment-size></code>	Optional	Specifies the maximum size of a single pool segment. The maximum size of the entire pool is the total number of segments times the maximum size of a segment. Default value is 16MB.
<code><min-buffer-size></code>	Optional	Specifies the smallest available buffer size to be stored in a segment. This value must be a multiple of 1024. Therefore, the smallest possible buffer is 1024 bytes. Default value is 1KB.
<code><growth-factor></code>	Optional	Specifies the rate of growth (as bitwise left shift) between successive segments. Default value is 2.

multicast-listener

Used in: [cluster-config](#).

Description

Specifies the configuration information for the Multicast listener. This element is used to specify the address (see <address> subelement) and port (see <port> subelement) that a cluster will use for cluster wide and point-to-multipoint communications. All nodes in a cluster must use the same multicast address and port, whereas distinct clusters on the same network should use different multicast addresses.

Multicast-Free Clustering

By default, Coherence uses a multicast protocol to discover other nodes when forming a cluster. If multicast networking is undesirable, or unavailable in your environment, the [well-known-addresses](#) feature may be used to eliminate the need for multicast traffic. If you are having difficulties in establishing a cluster by using multicast, see [Chapter 43, "Performing a Multicast Connectivity Test."](#)

Elements

[Table A-31](#) describes the elements you can define within the `multicast-listener` element.

Table A–31 *multicast-listener Subelements*

Element	Required /Optional	Description
<address>	Required	Specifies the multicast IP address that a Socket will listen or publish on. Legal values are from 224.0.0.0 to 239.255.255.255. Default value depends on the release and build level and typically follows the convention of {build}. {major version}. {minor version}. {patch}. For example, for Coherence Release 2.2 build 255 it is 225.2.2.0. Preconfigured is <code>tangosol.coherence.clusteraddress</code> . See Appendix C, "Command Line Overrides" for more information.
<port>	Required	Specifies the port that the Socket will listen or publish on. Legal values are from 1 to 65535. Default value depends on the release and build level and typically follows the convention of {version}+{{ {build}}. For example, for Coherence Release 2.2 build 255 it is 22255. Preconfigured override is <code>tangosol.coherence.clusterport</code> . See Appendix C, "Command Line Overrides" for more information.
<interface>	Optional	<p>Specifies the IP address that a multicast socket will be bound to. By default, the interface (NIC) of the unicast-listener IP address is used for the multicast socket; this option allows the interface to be specified. Setting this address to 0.0.0.0 allows the OS to use the unicast routing table to select the interface automatically.</p> <p>WARNING: With rare exception, use of this particular option is strongly discouraged, as it can lead to a condition known as "partial failure." Partial failure occurs when some portion of the cluster communication is working and other cluster communication has failed. Partial failure can occur when using this option, because the interface (and thus network) used for multicast traffic can be different from the interface (and thus network) used for unicast (UDP/IP) and TCP-ring (TCP/IP) traffic. If one interface (or network) fails, some communication can continue to succeed, while other communication fails, which may cause failover to take longer to occur. Since clustering handles node (and thus interface) failure, it is preferable to have all communication fail together, and thus the use of this option is strongly discouraged.</p>
<time-to-live>	Required	<p>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.</p> <p>Default value is 4. Preconfigured override is <code>tangosol.coherence.ttl</code>. See Appendix C, "Command Line Overrides" for more information.</p>
<packet-buffer>	Required	Specifies how many incoming packets the operating system will be requested to buffer. The value may be expressed either in terms of packets or bytes.

Table A-31 (Cont.) multicast-listener Subelements

Element	Required /Optional	Description
<priority>	Required	Specifies a priority of the multicast listener execution thread. Legal values are from 1 to 10. Default value is 8.
<join-timeout-millis econds>	Required	<p>Specifies the number of milliseconds that a new member will wait without finding any evidence of a cluster before starting its own cluster and electing itself as the senior cluster member. Legal values are from 1 to 1000000.</p> <p>Note: For production use, the recommended value is 30000. Default value is 6000.</p>
<multicast-threshold -percent>	Required	<p>Specifies the threshold percentage value used to determine whether a packet will be sent by using unicast or multicast. It is a percentage value and is in the range of 1% to 100%. In a cluster of "n" nodes, a particular node sending a packet to a set of other (that is, not counting self) destination nodes of size "d" (in the range of 0 to n-1), the packet will be sent multicast if and only if the following both hold true:</p> <ol style="list-style-type: none"> 1. The packet is being sent over the network to more than one other node, that is, (d > 1). 2. The number of nodes is greater than the threshold, that is, (d > (n-1) * (threshold/100)). <p>Setting this value to 1 will allow the implementation to use multicast for basically all multi-point traffic.</p> <p>Setting it to 100 will force the implementation to use unicast for all multi-point traffic except for explicit broadcast traffic (for example, cluster heartbeat and discovery) because the 100% threshold will never be exceeded. With the setting of 25 the implementation will send the packet using unicast if it is destined for less than one-fourth of all nodes, and send it using multicast if it is destined for the one-fourth or more of all nodes.</p> <p>Note: This element is only used if the well-known-addresses element is empty. Legal values are from 1 to 100. Default value is 25.</p>

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "[Element Attributes](#)" on page A-88 for more information on this attribute.

notification-queueing

Used in: [packet-publisher](#).

Description

The `notification-queueing` element is used to specify the timing of notifications packets sent to other cluster nodes. Notification packets are used to acknowledge the receipt of packets which require confirmation.

Batched Acknowledgments

Rather than sending an individual ACK for each received packet which requires confirmation, Coherence will batch a series of acknowledgments for a given sender into a single ACK. The `<ack-delay-milliseconds>` specifies the maximum amount of time that an acknowledgment will be delayed before an ACK notification is sent. By batching the acknowledgments Coherence avoids wasting network bandwidth with many small ACK packets.

Negative Acknowledgments

When enabled cluster nodes will use packet ordering to perform early packet loss detection (see the `<use-nack-packets>` subelement of `<incoming-message-handler>`). This allows Coherence to identify a packet as likely being lost and retransmit it well before the packets scheduled (see the `<resend-milliseconds>` subelement of `<packet-delivery>`).

Elements

The following table describes the elements you can define within the `notification-queueing` element.

Table A-32 *notification-queueing Subelements*

Element	Required/ Optional	Description
<code><ack-delay-milliseconds></code>	Required	Specifies the maximum number of milliseconds that the packet publisher will delay before sending an ACK packet. The ACK packet may be transmitted earlier if number of batched acknowledgments fills the ACK packet. This value should be substantially lower than the remote node's packet-delivery resend timeout, to allow ample time for the ACK to be received and processed by the remote node before the resend timeout expires. Default value is 16.
<code><nack-delay-milliseconds></code>	Required	Specifies the number of milliseconds that the packet publisher will delay before sending a NACK packet. Default value is 1.

outgoing-message-handler

Used in: [cluster-config](#)

Description

The outgoing-message-handler element contains the outgoing message handler (also known as a dispatcher) related configuration information.

Elements

[Table A-33](#) describes the elements you can define within the outgoing-message-handler element.

Table A-33 *outgoing-message-handler Subelement*

Element	Required/ Optional	Description
<use-filters>	Optional	Specifies a list of <filter-name> elements to be used by this handler. See the <filters> element for detailed information on defining a filter.
<message-pool>	Optional	Specifies the size of the message buffer pool.

outstanding-packets

Used in: [flow-control](#).

Description

Defines the number of unconfirmed packets that will be sent to a cluster node before packets addressed to that node will be deferred. This helps to prevent the sender from flooding the recipient's network buffers.

Auto Tuning

The value may be specified as either an explicit number by using the `maximum-packets` element, or as a range by using both the `maximum-packets` and `minimum-packets` elements. When a range is specified, this setting will be dynamically adjusted based on network statistics.

Elements

[Table A-34](#) describes the elements you can define within the `outstanding-packets` element.

Table A-34 *outstanding-packets Subelements*

Element	Required/ Optional	Description
<code><maximum-packets></code>	Optional	The maximum number of unconfirmed packets that will be sent to a cluster node before packets addressed to that node will be deferred. It is recommended that this value not be set below 256. Default is 4096.
<code><minimum-packets></code>	Optional	The lower bound on the range for the number of unconfirmed packets that will be sent to a cluster node before packets addressed to that node will be deferred. It is recommended that this value not be set below 16. Default is 64.

packet-buffer

Used in: [unicast-listener](#), [multicast-listener](#), [packet-publisher](#).

Description

Specifies the size (in packets or bytes) of the operating system buffer for datagram sockets.

Performance Impact

Large inbound buffers help insulate the Coherence network layer from JVM pauses caused by the Java Garbage Collector. While the JVM is paused, Coherence is unable to dequeue packets from any inbound socket. If the pause is long enough to cause the packet buffer to overflow, the packet reception will be delayed as the originating node will need to detect the packet loss and retransmit the packet(s).

It's just a hint

The operating system will only treat the specified value as a hint, and is not required to allocate the specified amount. In the event that less space is allocated than requested Coherence will issue a warning and continue to operate with the constrained buffer, which may degrade performance. See [Chapter 45, "Performance Tuning,"](#) for details on configuring your operating system to allow larger buffers.

Latency Versus Throughput

When setting this for transmit (that is, within [packet-publisher](#)), higher values may allow for increased throughput, while lower values may allow for decreased latency. If you make any changes to this value, it is recommended that you evaluate how it effects performance in all of these dimensions.

Elements

[Table A-35](#) describes the elements you can define within the `packet-buffer` element.

Table A-35 *packet-buffer Subelements*

Element	Required/ Optional	Description
<code><maximum-packets></code>	Optional	For unicast-listener , multicast-listener and packet-publisher : Specifies the number of packets of packet-size that the datagram socket will be asked to size itself to buffer. See <code>SO_SNDBUF</code> and <code>SO_RCVBUF</code> . Actual buffer sizes may be smaller if the underlying socket implementation cannot support more than a certain size. Defaults are 32 for publishing, 64 for multicast listening, and 1428 for unicast listening. The <code><maximum-packets></code> element cannot be specified if the <code><size></code> element is specified.
<code><size></code>	Optional	Specifies the requested size of the underlying socket buffer in bytes rather than the number of packets. The <code><size></code> element cannot be specified if the <code><maximum-packets></code> 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.

Default Configuration

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

Elements

[Table A-36](#) describes the elements you can define within the `packet-bundling` element.

Table A-36 *packet-bundling Subelements*

Element	Required/Optional	Description
<code><maximum-deferral-time></code>	Optional	<p>The maximum amount of time to defer a packet while waiting for additional packets to bundle. A value of zero will result in the algorithm not waiting, and only bundling the readily accessible packets. A value greater than zero will cause some transmission deferral while waiting for additional packets to become available. This value is typically set below 250 microseconds to avoid a detrimental throughput impact. If the units are not specified, nanoseconds are assumed.</p> <p>Default value is 1us (microsecond).</p>
<code><aggression-factor></code>	Optional	<p>Specifies the aggressiveness of the packet deferral algorithm. Where as the <code>maximum-deferral-time</code> element defines the upper limit on the deferral time, the <code>aggression-factor</code> influences the average deferral time. The higher the aggression value, the longer the Publisher may wait for additional packets. The factor may be expressed as a real number, and often times values between 0.0 and 1.0 will be allow for high packet utilization while keeping latency to a minimum.</p> <p>Default value is zero.</p>

packet-delivery

Used in: [packet-publisher](#).

Description

Specifies timing and transmission rate parameters related to packet delivery.

Death Detection

The `<timeout-milliseconds>` and `<heartbeat-milliseconds>` subelements are used in detecting the death of other cluster nodes.

Elements

[Table A-37](#) describes the elements you can define within the `packet-delivery` element.

Table A-37 *packet-delivery Subelements*

Element	Required/ Optional	Description
<code><resend-milliseconds></code>	Required	For packets which require confirmation, specifies the minimum amount of time in milliseconds to wait for a corresponding ACK packet, before resending a packet. Default value is 200.
<code><timeout-milliseconds></code>	Required	For packets which require confirmation, specifies the maximum amount of time, in milliseconds, that a packet will be resent. After this timeout expires Coherence will make a determination if the recipient is to be considered "dead". This determination takes additional data into account, such as if other nodes are still able to communicate with the recipient. Default value is 300000. For production use, the recommended value is the greater of 300000 and two times the maximum expected full GC duration.
<code><heartbeat-milliseconds></code>	Required	Specifies the interval between heartbeats. Each member issues a unicast heartbeat, and the most senior member issues the cluster heartbeat, which is a broadcast message. The heartbeat is used by the tcp-ring-listener as part of fast death detection. Default value is 1000.
<code><flow-control></code>	Optional	Configures per-node packet throttling and remote GC detection.
<code><packet-bundling></code>	Optional	Configures how aggressively Coherence will attempt to maximize packet utilization.

packet-pool

Used in: [incoming-message-handler](#), [packet-publisher](#).

Description

A pool of packets that Coherence internally maintains for use in transmitting and receiving UDP packets. Unlike the [packet-buffer](#), these buffers are managed by Coherence rather than the operating system, and allocated on the JVM's heap.

Performance Impact

The packet pools are used as a reusable buffer between Coherence network services. For packet transmission, this defines the maximum number of packets which can be queued on the [packet-speaker](#) before the [packet-publisher](#) must block. For packet reception, this defines the number of packets which can be queued on the [incoming-message-handler](#) before the [unicast-listener](#), and [multicast-listener](#) must block.

Elements

[Table A-38](#) describes the subelements you can define within the `packet-pool` element.

Table A-38 *packet-pool Subelements*

Element	Required/ Optional	Description
<code><size></code>	Required	Specifies the maximum size of the pool of reusable packets to be utilized by the services responsible for publishing and receiving described in bytes. The pools are initially empty, and will grow on demand up to the specified limits. The default value is 16MB for both transmitting and receiving.

packet-publisher

Used in: [cluster-config](#).

Description

Specifies configuration information for the Packet publisher, which manages network data transmission.

Reliable packet delivery

The Packet publisher is responsible for ensuring that transmitted packets reach the destination cluster node. The publisher maintains a set of packets which are waiting to be acknowledged, and if the ACK does not arrive by the `packet-delivery` resend timeout, the packet will be retransmitted (see `<packet-delivery>` subelement). The recipient node will delay the ACK, to batch a series of ACKs into a single response (see `<notification-queuing>` subelement).

Throttling

The rate at which the publisher will accept and transmit packet may be controlled by using the [traffic-jam](#) and [flow-control](#) settings. Throttling may be necessary when dealing with slow networks, or small [packet-buffer](#).

Elements

[Table A-39](#) describes the elements you can define within the `packet-publisher` element.

Table A-39 *packet-publisher Subelements*

Element	Required/ Optional	Description
<code><enabled></code>	Optional	Specifies if TCMP clustering is enabled. For Coherence editions which support both Coherence Extend and Coherence TCMP based clustering, this feature allows TCMP to be disabled to ensure that a node only connects by using the Extend protocol. Default value is <code>true</code> . Preconfigured override is <code>tangosol.coherence.tcmp.enabled</code> . See Appendix C, "Command Line Overrides" for more information.
<code><packet-size></code>	Optional	Specifies the UDP packet sizes to use.
<code><packet-delivery></code>	Required	Specifies timing parameters related to reliable packet delivery.
<code><notification-queuing></code>	Required	Contains the notification queue related configuration info.
<code><traffic-jam></code>	Required	Specifies the maximum number of packets which can be enqueued on the publisher before client threads block.
<code><packet-buffer></code>	Required	Specifies how many outgoing packets the operating system will be requested to buffer. The value may be expressed either in terms of packets or bytes.
<code><packet-pool></code>	Required	Specifies how many outgoing packets Coherence will buffer before blocking.
<code><priority></code>	Required	Specifies a priority of the packet publisher execution thread. Legal values are from 1 to 10. Default value is 6.

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See ["Element Attributes"](#) on page A-88 for more information.

packet-size

Used in: [packet-publisher](#).

Description

The packet-size element specifies the maximum and preferred UDP packet sizes (see the <maximum-length> and <preferred-length> subelements). All cluster nodes must use identical maximum packet sizes. For optimal network utilization this value should be 32 bytes less then the network MTU.

Note: When specifying a UDP packet size larger then 1024 bytes on Microsoft Windows a registry setting must be adjusted to allow for optimal transmission rates. See "[Datagram size \(Microsoft Windows\)](#)" on page 45-3 for details.

Elements

[Table A-40](#) describes the subelements you can define within the packet-size element.

Table A-40 packet-size Subelement

Element	Required/ Optional	Description
<maximum-length>	Required	Specifies the packet size in bytes which all cluster members can safely support. This value must be the same for all members in the cluster. A low value can artificially limit the maximum size of the cluster. This value should be at least 512, and defaults to 64KB.
<preferred-length>	Required	<p>Specifies the preferred size, in bytes, of the DatagramPacket objects that will be sent and received on the unicast and multicast sockets.</p> <p>This value can be larger or smaller then the <maximum-length> value, and need not be the same for all cluster members. The ideal value is one which will fit within the network MTU, leaving enough space for either the UDP or TCP packet headers, which are 32, and 52 bytes respectively.</p> <p>This value should be at least 512, and defaults to a value based on the local nodes MTU. An MTU of 1500 is assumed if the MTU cannot be obtained.</p>

packet-speaker

Used in: [cluster-config](#).

Description

Specifies configuration information for the Packet speaker, used for network data transmission.

Offloaded Transmission

The Packet speaker is responsible for sending packets on the network. The speaker is used when the [packet-publisher](#) detects that a network send operation is likely to block. This allows the Packet publisher to avoid blocking on IO and continue to prepare outgoing packets. The Publisher will dynamically choose whether to use the speaker as the packet load changes.

Elements

[Table A-41](#) describes the subelements you can define within the `packet-speaker` element.

Table A-41 *packet-speaker Subelements*

Element	Required/ Optional	Description
<code><volume-threshold></code>	Optional	Specifies the packet load which must be present for the speaker to be activated.
<code><priority></code>	Required	Specifies a priority of the packet speaker execution thread. Legal values are from 1 to 10. Default value is 8.

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See "[Element Attributes](#)" on page A-88 for more information on this attribute.

pause-detection

Used in: [flow-control](#).

Description

Remote Pause detection allows Coherence to detect and react to a cluster node becoming unresponsive (likely due to a long GC). When a node is marked as paused, packets addressed to it will be sent at a lower rate until the node resumes responding. This remote GC detection is used to avoid flooding a node while it is incapable of responding.

Elements

[Table A-42](#) describes the subelements you can define within the `pause-detection` element.

Table A-42 *pause-detection Subelements*

Element	Required/ Optional	Description
<code><maximum-packets></code>	Optional	The maximum number of packets that will be resent to an unresponsive cluster node before assuming that the node is paused. Specifying a value of 0 will disable pause detection. Default is 16.

provider

Used in: [ssl](#), [identity-manager](#), [trust-manager](#).

Description

The provider element contains the configuration information for a security provider that extends the `java.security.Provider` class.

Elements

[Table A-43](#) describes the subelements you can define within the provider element.

Table A-43 *provider Subelements*

Element	Required/ Optional	Description
<code><name></code>	Optional	Specifies the name of a security provider that extends the <code>java.security.Provider</code> class. The class name can be entered using either this element or by using the <code><class-name></code> element or by using the <code><class-factory-name></code> element.
<code><class-name></code>	Optional	Specifies the name of a security provider that extends the <code>java.security.Provider</code> class. This element cannot be used together with the <code><name></code> element or the <code><class-factory-name></code> element.
<code><class-factory-name></code>	Optional	Specifies a factory class for creating <code>Provider</code> instances. The instances must implement the <code>java.security.Provider</code> class. This element cannot be used together with the <code><name></code> element or the <code><class-name></code> element. This element can be used together with the <code><method-name></code> element.
<code><method-name></code>	Optional	Specifies the name of a static factory method on the factory class which will perform object instantiation.
<code><init-params></code>	Optional	Contains class initialization parameters for the provider implementation. This element cannot be used together with the <code><name></code> element.

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-44](#) describes the subelements you can define within the `reporter` element.

Table A-44 *reporter Subelements*

Element	Required/ Optional	Description
<code><configuration></code>	Required	Specifies the location for the Reporter Batch XML. The default file is <code>reports/report-group.xml</code> and is located in the <code>coherence.jar</code> library.
<code><autostart></code>	Required	Specifies whether or not the Reporter automatically starts when the node starts. Valid values are <code>true</code> and <code>false</code> . Default value is <code>false</code> .
<code><distributed></code>	Required	Specifies whether or not the reporter runs on multiple management nodes. Valid values are <code>true</code> and <code>false</code> . Default value is <code>false</code> .

security-config

Used in: [coherence](#).

Elements

[Table A-45](#) describes the subelements you can define within the `security-config` element.

Table A-45 *security-config Subelements*

Element	Required/ Optional	Description
<code><enabled></code>	Required	Specifies whether the security features are enabled. All other configuration elements in the <code>security-config</code> group will be verified for validity and used if and only if the value of this element is <code>true</code> . Legal values are <code>true</code> or <code>false</code> . Default value is <code>false</code> . Preconfigured override is <code>tangosol.coherence.security</code> . See Appendix C, "Command Line Overrides" for more information.
<code><login-module-name></code>	Required	Specifies the name of the JAAS LoginModule that should be used to authenticate the caller. This name should match a module in a configuration file will be used by the JAAS (for example specified by using the <code>-Djava.security.auth.login.config</code> Java command line attribute). For details please refer to the Sun Login Module Developer's Guide.
<code><access-controller></code>	Required	Contains the configuration information for the class that implements <code>com.tangosol.net.security.AccessController</code> interface, which will be used by the security framework to check access rights for clustered resources and encrypt/decrypt node-to-node communications regarding those rights.
<code><callback-handler></code>	Optional	Contains the configuration information for the class that implements <code>javax.security.auth.callback.CallbackHandler</code> interlace which will be called if an attempt is made to access a protected clustered resource when there is no identity associated with the caller.
<code><identity-asserter></code>	Optional	Contains the configuration information for a class that implements the <code>com.tangosol.net.security.IdentityAsserter</code> interface which is called to validate an identity token in order to establish a user's identity. An identity asserter is used together with an identity transformer to protect connections between Coherence*Extend clients and proxies.
<code><identity-transformer></code>	Optional	Contains the configuration information for the class that implements <code>com.tangosol.net.security.IdentityTransformer</code> interface which is called to transform a <code>Subject</code> (<code>Principal</code> for <code>.NET</code>) to a token that asserts identity. An identity transformer is used together with an identity asserter to protect connections between Coherence*Extend clients and proxies.
<code><subject-scope></code>	Optional	Specifies whether or not the remote cache or service reference is shared by subject. Valid values are <code>true</code> or <code>false</code> . Setting the value to <code>true</code> means that remote references are not globally shared; each subject will get a different reference. Default value is <code>false</code> .

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See ["Element Attributes"](#) on page A-88 for more information.

serializers

Used in: [cluster-config](#)

Description

The `serializers` element contains any number of serializer class configurations. Serializer classes must implement `com.tangosol.io.Serializer`. Each serializer class is defined within a `serializer` subelement. The operational configuration file contains a default Java and POF serializer class configuration:

```
<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>
```

Serializers that are defined in the operational configuration can be referenced by individual cache scheme definitions (see ["serializer"](#) on page B-95) and can be referenced by the default serializer for services that do not explicitly define a serializer (see ["defaults"](#) on page B-25).

Additional serializers can be defined in an operational override file as required.

Elements

[Table A-46](#) describes the subelements you can define within the `serializer` element.

Table A-46 *serializer Subelements*

Element	Required/ Optional	Description
<code><class-name></code>	Optional	Specifies a class that implements <code>com.tangosol.io.Serializer</code> . This element cannot be used together with the <code><class-factory-name></code> element.
<code><class-factory-name></code>	Optional	Specifies a factory class for creating custom serializer instances. The instances must implement <code>com.tangosol.io.Serializer</code> . This element cannot be used together with the <code><class-name></code> element. This element can be used together with the <code><method-name></code> element.
<code><method-name></code>	Optional	Specifies the name of a static factory method on the factory class which will perform object instantiation.
<code><init-params></code>	Optional	Contains class initialization parameters for the serializer implementation.

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-47](#) describes the subelements you can define within the `service-guardian` element.

Table A-47 *service-guardian Subelements*

Element	Required/ Optional	Description
<code><timeout-milliseconds></code>	Optional	<p>The timeout value used to guard against deadlocked or unresponsive services. It is recommended that <code>service-guardian/timeout-milliseconds</code> be set equal to or greater than the <code>packet-delivery/timeout-milliseconds</code> value. A timeout of 0 will disable service guardians.</p> <p>Default value is 305000.</p> <p>Preconfigured override is <code>tangosol.coherence.guard.timeout</code></p>
<code><service-failure-policy></code>	Optional	<p>Specifies the action to take when an abnormally behaving service thread cannot be terminated gracefully by the service guardian.</p> <p>Legal values are:</p> <ul style="list-style-type: none"> ■ <code>exit-cluster</code> – 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. ■ <code>exit-process</code> – 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. ■ <code>logging</code> – 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 <code>com.tangosol.net.ServiceFailurePolicy</code> implementation. <p>Default value is <code>exit-cluster</code>.</p>

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See ["Element Attributes"](#) on page A-88 for more information.

services

Used in: [cluster-config](#).

Description

Specifies the configuration for Coherence services.

Service Components

The types of services which can be configured includes:

- `ReplicatedCache`—A cache service which maintains copies of all cache entries on all cluster nodes which run the service.
- `ReplicatedCache.Optimistic`—A version of the `ReplicatedCache` which uses optimistic locking.
- `DistributedCache`—A cache service which evenly partitions cache entries across the cluster nodes which run the service.
- `SimpleCache` —A version of the `ReplicatedCache` which lacks 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.

Elements

[Table A-48](#) describes the subelements you can define for each `services` element.

Table A–48 *services Subelements*

Element	Required/ Optional	Description
<service-type>	Required	Specifies the canonical name for a service, allowing the service to be referenced from the <code>service-name</code> element in cache configuration caching schemes. See " caching-schemes " on page B-21 for more information.
<service-component>	Required	Specifies either the fully qualified class name of the service or the relocatable component name relative to the base Service component. Legal values are: <ul style="list-style-type: none"> ■ <code>ReplicatedCache</code> ■ <code>ReplicatedCache.Optimistic</code> ■ <code>DistributedCache</code> ■ <code>SimpleCache</code> ■ <code>LocalCache</code> ■ <code>InvocationService</code>
<use-filters>	Optional	Contains the list of filters names to be used by this service. For example, specify <code>use-filter</code> as follows <pre><use-filters> <filter-name>gzip</filter-name> </use-filters></pre> will activate <code>gzip</code> 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 <code>service-component</code> . For more service specific parameter information see: <ul style="list-style-type: none"> ■ "DistributedCache Service Parameters" on page A-65 ■ "ReplicatedCache Service Parameters" on page A-69 ■ "InvocationService Parameters" on page A-70

The content override attributes `xml-override` and `id` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document.

Initialization Parameter Settings

The `<init-param>` element in the Coherence operational configuration deployment descriptor defines initialization parameters for a service or filter. The parameters that appear under `init-param` will be different, depending on the service or filter you are working with.

The following sections describe the parameters that can be configured for these services and filters:

- [DistributedCache Service Parameters](#)
- [ReplicatedCache Service Parameters](#)
- [InvocationService Parameters](#)
- [ProxyService Parameters](#)

The tables in each section describe the specific `<param-name>` — `<param-value>` pairs that can be configured for various elements. The **Parameter Name** column refers

to the value of the param-name element and **Value Description** column refers to the possible values for the corresponding param-value element.

For example, the sample entry in [Table A-49](#) means that the init-params element may look like the configuration in [Example A-2](#) or [Example A-3](#).

Table A-49 Sample Table Entry

Parameter Value	Value Description
local-storage	Specifies whether this member of the DistributedCache service enables the local storage. Legal values are true or false. Default value is true. Preconfigured override is tangosol.coherence.distributed.localstorage. See Appendix C, "Command Line Overrides" for more information.

Example A-2 Sample init-param Configuration

```
...
<init-params>
  <init-param>
    <param-name>local-storage</param-name>
    <param-value>>false</param-value>
  </init-param>
</init-params>
...
```

or as follows:

Example A-3 Another Sample init-param Configuration

```
...
<init-params>
  <init-param>
    <param-name>local-storage</param-name>
    <param-value>true</param-value>
  </init-param>
</init-params>
...
```

DistributedCache Service Parameters

DistributedCache [<services>](#) elements support the parameters described in [Table A-50](#). These settings may also be specified as part of the [<distributed-scheme>](#) element in the coherence-cache-config.xml descriptor.

Table A–50 DistributedCache Service Parameters

Parameter Name	Value, Description
backup-count	<p>Specifies the number of members of the DistributedCache service that hold the backup data for each unit of storage in the cache. Value of 0 means that in the case of abnormal termination, some portion of the data in the cache will be lost. Value of N means that if up to N cluster nodes terminate immediately, the cache data will be preserved. To maintain the distributed cache of size M, the total memory usage in the cluster does not depend on the number of cluster nodes and will be in the order of $M*(N+1)$.</p> <p>Recommended values are 0, 1 or 2.</p> <p>Default value is 1.</p>
backup-storage/ class-name	<p>Only applicable with the custom type. Specifies a class name for the custom storage implementation. If the class implements <code>com.tangosol.run.xml.XmlConfigurable</code> interface then upon construction the <code>setConfig</code> method is called passing the entire backup-storage element.</p>
backup-storage/ directory	<p>Only applicable with the file-mapped type. Specifies the path name for the directory that the disk persistence manager (<code>com.tangosol.util.nio.MappedBufferManager</code>) will use as "root" to store files in. If not specified or specifies a non-existent directory, a temporary file in the default location is used.</p> <p>Default value is the default temporary directory designated by the Java runtime.</p>
backup-storage/ initial-size	<p>Only applicable with the off-heap and file-mapped types. Specifies the initial buffer size in bytes. The value of this element must be in the following format: <code>[\d]+[.][\d]?[K k M m G g]?[B b]?</code> where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied:</p> <ul style="list-style-type: none"> ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) <p>If the value does not contain a factor, a factor of mega is assumed.</p> <p>Legal values are positive integers between 1 and <code>Integer.MAX_VALUE - 1023</code> (that is, 2,147,482,624 bytes).</p> <p>Default value is 1MB.</p>
backup-storage/ maximum-size	<p>Only applicable with the off-heap and file-mapped types. Specifies the maximum buffer size in bytes. The value of this element must be in the following format: <code>[\d]+[.][\d]?[K k M m G g]?[B b]?</code> where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied:</p> <ul style="list-style-type: none"> ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) <p>If the value does not contain a factor, a factor of mega is assumed.</p> <p>Legal values are positive integers between 1 and <code>Integer.MAX_VALUE - 1023</code> (that is, 2,147,482,624 bytes).</p> <p>Default value is 1024MB.</p>
backup-storage/ scheme-name	<p>Only applicable with the scheme type. Specifies a scheme name for the <code>ConfigurableCacheFactory</code>.</p>

Table A–50 (Cont.) DistributedCache Service Parameters

Parameter Name	Value, Description
backup-storage/ type	<p>Specifies the type of the storage used to hold the backup data. Legal values are:</p> <ul style="list-style-type: none"> ■ on-heap—The corresponding implementations class is <code>java.util.HashMap</code>. ■ off-heap—The corresponding implementations class is <code>com.tangosol.util.nio.BinaryMap</code> using <code>com.tangosol.util.nio.DirectBufferManager</code>. Only available with JDK 1.5 and later. ■ file-mapped—The corresponding implementations class is <code>com.tangosol.util.nio.BinaryMap</code> using <code>com.tangosol.util.nio.MappedBufferManager</code>. Only available with JDK 1.5 and later. ■ custom—The corresponding implementations class is the class specified by the <code>backup-storage/class</code> element. ■ scheme—The corresponding implementations class is the map returned by the <code>ConfigurableCacheFactory</code> for the scheme referred to by the <code>backup-storage/scheme-name</code> element. <p>Default value is on-heap.</p> <p>Preconfigured override is <code>tangosol.coherence.distributed.backup</code>. See Appendix C, "Command Line Overrides" for more information.</p>
key-associator/ class-name	<p>Specifies the name of a class that implements the <code>com.tangosol.net.partition.KeyAssociator</code> interface. This implementation must have a zero-parameter public constructor.</p>
key-partitioning/ class-name	<p>Specifies the name of a class that implements the <code>com.tangosol.net.partition.KeyPartitioningStrategy</code> interface. This implementation must have a zero-parameter public constructor.</p>
lease-granularity	<p>Specifies the lease ownership granularity. Available since release 2.3. Legal values are:</p> <ul style="list-style-type: none"> ■ thread ■ member <p>A value of <code>thread</code> means that locks are held by a thread that obtained them and can only be released by that thread. A value of <code>member</code> means that locks are held by a cluster node and any thread running on the cluster node that obtained the lock can release it.</p> <p>Default value is <code>thread</code>.</p>
local-storage	<p>Specifies whether this member of the <code>DistributedCache</code> service enables local storage.</p> <p>Normally this value should be left unspecified within the configuration file, and instead set on a per-process basis using the <code>tangosol.coherence.distributed.localstorage</code> system property. This allows cache clients and servers to use the same configuration descriptor.</p> <p>Legal values are <code>true</code> or <code>false</code>. Default value is <code>true</code>.</p> <p>Preconfigured override is <code>tangosol.coherence.distributed.localstorage</code>. See Appendix C, "Command Line Overrides" for more information.</p>

Table A–50 (Cont.) DistributedCache Service Parameters

Parameter Name	Value, Description												
partition-count	<p>Specifies the number of partitions that a partitioned (distributed) cache will be "chopped up" into. Each member running the partitioned cache service that has the local-storage (<local-storage> subelement) option set to true will manage a "fair" (balanced) number of partitions.</p> <p>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 in size.</p> <p>The following are good defaults based on service storage sizes:</p> <table> <tr> <th>service storage</th><th>partition-count</th></tr> <tr> <td>100M</td><td>257</td></tr> <tr> <td>1G</td><td>509</td></tr> <tr> <td>10G</td><td>2039</td></tr> <tr> <td>50G</td><td>4093</td></tr> <tr> <td>100G</td><td>8191</td></tr> </table> <p>A list of first 1,000 primes can be found at: http://primes.utm.edu/lists/</p> <p>Valid values are positive integers. The default value is 257.</p>	service storage	partition-count	100M	257	1G	509	10G	2039	50G	4093	100G	8191
service storage	partition-count												
100M	257												
1G	509												
10G	2039												
50G	4093												
100G	8191												
partition-listener/ class-name	Specifies the name of a class that implements the <code>com.tangosol.net.partition.PartitionListener</code> interface. This implementation must have a zero-parameter public constructor.												
request-timeout	<p>Specifies the maximum amount of time a client will wait for a response before abandoning the original request. The request time is measured on the client side as the time elapsed from the moment a request is sent for execution to the corresponding server node(s) and includes the following:</p> <ul style="list-style-type: none"> 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 <p>The value of this element must be in the following format: <code>[\d] + [[.] [\d] +] ? [MS ms S s M m H h D d] ?</code> where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> MS or ms (milliseconds) S or s (seconds) M or m (minutes) H or h (hours) D or d (days) <p>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.</p>												
task-hung-threshold	<p>Specifies the amount of time in milliseconds that a task can execute before it is considered "hung".</p> <p>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 <code>thread-count</code> value is positive).</p> <p>Legal values are positive integers or zero (indicating no default timeout).</p>												

Table A–50 (Cont.) DistributedCache Service Parameters

Parameter Name	Value, Description
<code>task-timeout</code>	<p>Specifies the default timeout value in milliseconds for tasks that can be timed-out (for example, implement the <code>com.tangosol.net.PriorityTask</code> interface), but don't explicitly specify the task execution timeout value. The task execution time is measured on the server side and does not include the time spent waiting in a service backlog queue before being started. This attribute is applied only if the thread pool is used (the <code>thread-count</code> value is positive).</p> <p>Legal values are positive integers or zero (indicating no default timeout).</p>
<code>thread-count</code>	<p>Specifies the number of daemon threads used by the distributed cache service. If zero, all relevant tasks are performed on the service thread.</p> <p>Legal values are from positive integers or zero.</p> <p>Default value is 0. Preconfigured override is <code>tangosol.coherence.distributed.threads</code>. See Appendix C, "Command Line Overrides" for more information.</p> <p>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 IO 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.</p>
<code>transfer-threshold</code>	<p>Specifies the threshold for the primary buckets distribution in kilobytes. When a new node joins the distributed cache service or when a member of the service leaves, the remaining nodes perform a task of bucket ownership re-distribution. During this process, the existing data gets rebalanced along with the ownership information. This parameter indicates a preferred message size for data transfer communications. Setting this value lower will make the distribution process take longer, but will reduce network bandwidth utilization during this activity.</p> <p>Legal values are integers greater than zero.</p> <p>Default value is 512 (0.5MB). Preconfigured override is <code>tangosol.coherence.distributed.transfer</code>. See Appendix C, "Command Line Overrides" for more information.</p>

ReplicatedCache Service Parameters

ReplicatedCache [services](#) elements support the parameters described in [Table A–51](#). These settings may also be specified as part of the `replicated-scheme` element in the `coherence-cache-config.xml` descriptor.

Table A–51 ReplicatedCache Service Parameters

Parameter Name	Value Description
lease-granularity	<p>Specifies the lease ownership granularity. Available since release 2.3. Legal values are:</p> <ul style="list-style-type: none"> ■ thread ■ member <p>A value of <code>thread</code> means that locks are held by a thread that obtained them and can only be released by that thread. A value of <code>member</code> means that locks are held by a cluster node and any thread running on the cluster node that obtained the lock can release it.</p> <p>Default value is <code>thread</code>.</p>
mobile-issues	<p>Specifies whether lease issues should be transferred to the most recent lock holders.</p> <p>Legal values are <code>true</code> or <code>false</code>.</p> <p>Default value is <code>false</code>.</p>
standard-lease-milliseconds	<p>Specifies the duration of the standard lease in milliseconds. When a lease has aged past this number of milliseconds, the lock will automatically be released. Set this value to zero to specify a lease that never expires. The purpose of this setting is to avoid deadlocks or blocks caused by stuck threads; the value should be set higher than the longest expected lock duration (for example, higher than a transaction timeout). It's also recommended to set this value higher than <code>packet-delivery/timeout-milliseconds</code> value.</p> <p>Legal values are from positive long numbers or zero.</p> <p>Default value is 0.</p>

InvocationService Parameters

`InvocationService` [services](#) elements support the following parameters listed in [Table A–52](#). These settings may also be specified as part of the `invocation-scheme` element in the `coherence-cache-config.xml` descriptor.

Table A–52 InvocationService Parameters

Parameter Name	Value, Description
<code>request-timeout</code>	<p>Specifies the default timeout value in milliseconds for requests that can time-out (for example, implement the <code>com.tangosol.net.PriorityTask</code> interface), but don't explicitly specify the request timeout value. The request time is measured on the client side as the time elapsed from the moment a request is sent for execution to the corresponding server node(s) and includes the following:</p> <ul style="list-style-type: none"> ■ 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 <p>Legal values are positive integers or zero (indicating no default timeout).</p>
<code>task-hung-threshold</code>	<p>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 <code>thread-count</code> value is positive).</p>
<code>task-timeout</code>	<p>Specifies the default timeout value in milliseconds for tasks that can be timed-out (for example, implement the <code>com.tangosol.net.PriorityTask</code> interface), but don't explicitly specify the task execution timeout value. The task execution time is measured on the server side and does not include the time spent waiting in a service backlog queue before being started. This attribute is applied only if the thread pool is used (the <code>thread-count</code> value is positive).</p> <p>Legal values are positive integers or zero (indicating no default timeout).</p>
<code>thread-count</code>	<p>Specifies the number of daemon threads to be used by the invocation service. If zero, all relevant tasks are performed on the service thread.</p> <p>Legal values are from positive integers or zero. Preconfigured override is <code>tangosol.coherence.invocation.threads</code>. See Appendix C, "Command Line Overrides" for more information.</p> <p>Default value is 0.</p> <p>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 IO 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.</p>

ProxyService Parameters

ProxyService [services](#) elements support the parameters described in [Table A–53](#). These settings may also be specified as part of the [proxy-scheme](#) element in the `coherence-cache-config.xml` descriptor.

Table A–53 *ProxyService Parameters*

Parameter Name	Value Description
thread-count	<p>Specifies the number of daemon threads to be used by the proxy service. If zero, all relevant tasks are performed on the service thread.</p> <p>Legal values are from positive integers or zero.</p> <p>Default value is 0.</p> <p>Proxy service threads perform operations on behalf of the calling application. Therefore, set the value to as many threads as there are concurrent operations that are occurring.</p>
task-hung-threshold	<p>Specifies the amount of time in milliseconds that a task can execute before it is considered "hung".</p> <p>Note: a posted task that has not yet started is never considered as hung. This attribute is applied only if the Thread pool is used (the thread-count value is positive).</p> <p>Legal values are positive integers or zero (indicating no default timeout).</p>
task-timeout	<p>Specifies the default timeout value in milliseconds for tasks that can be timed-out (for example, implement the <code>com.tangosol.net.PriorityTask</code> interface), but don't explicitly specify the task execution timeout value. The task execution time is measured on the server side and does not include the time spent waiting in a service backlog queue before being started. This attribute is applied only if the thread pool is used (the thread-count value is positive).</p> <p>Legal values are positive integers or zero (indicating no default timeout).</p>

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-54](#) describes the elements you can define within the `shutdown-listener` element.

Table A-54 *shutdown-listener Subelements*

Element	Required/ Optional	Description
<code><enabled></code>	Required	<p>Specifies the type of action to take upon an external JVM shutdown. Legal values:</p> <ul style="list-style-type: none"> ■ <code>none</code>—perform no explicit shutdown actions ■ <code>force</code>—perform "hard-stop" the node by calling <code>Cluster.stop()</code> ■ <code>graceful</code>—perform a "normal" shutdown by calling <code>Cluster.shutdown()</code> ■ <code>true</code>—same as <code>force</code> ■ <code>false</code>—same as <code>none</code> <p>Note: For production use, the suggested value is <code>none</code> unless testing has verified that the behavior on external shutdown is exactly what is desired. Default value is <code>force</code>. Preconfigured override is <code>tangosol.coherence.shutdownhook</code>. See Appendix C, "Command Line Overrides" for more information.</p>

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See ["Element Attributes"](#) on page A-88 for more information.

socket-address

Used in: [well-known-addresses](#).

Elements

[Table A-55](#) describes the subelements you can define within the `socket-address` element.

Table A-55 *socket-address Subelements*

Element	Required/ Optional	Description
<address>	Required	Specifies the IP address that a Socket will listen or publish on. Note: The localhost setting may not work on systems that define localhost as the loopback address; in that case, specify the machine name or the specific IP address.
<port>	Required	Specifies the port that the Socket will listen or publish on. Legal values are from 1 to 65535.

socket-provider

Used in: [socket-providers](#), [unicast-listener](#).

Description

The `<socket-provider>` element contains the configuration information for a socket and channel factory that implements the `com.tangosol.net.SocketProvider` interface. A socket provider configured within the `<unicast-listener>` element is for use with TCMP. Socket providers for Coherence*Extend are configured in a cache configuration file within the `<tcp-acceptor>` and `<tcp-initiator>` elements.

Socket providers that are defined within the `<socket-providers>` element can be referenced by a unicast listener configuration (see "[unicast-listener](#)" on page A-83), individual cache scheme definitions (see "[socket-provider](#)" on page B-97) and by the default socket provider for services that do not explicitly define a socket provider (see "[defaults](#)" on page B-25).

Out-of-box, the following pre-defined socket provider configurations are included. Additional socket providers can be defined in an operational override file as required.

- `system` (default)– The system socket provider returns socket instances based upon the JVM's default socket implementations.
- `tcp` – The TCP socket provider is a socket provider which, whenever possible, produces TCP-based sockets. This socket provider creates instances of `DatagramSocket` which are backed by TCP. When coupled with `well-known-address`, this allows Coherence TCMP to function entirely over TCP without the need UDP.

Note: if this socket provider is used without `well-known-addresses`, TCP is used for all unicast communications; while, multicast is utilized for group based communications.

- `ssl` – The ssl socket provider is a socket provider which only produces SSL protected sockets. Just as with the TCP socket provider, this includes a SSL/TCP based `DatagramSocket`. Unlike the TCP socket-provider, multicast sockets are not supported, and thus `well-known-addresses` must be enabled for TCMP to function with this provider.

This basic configuration allows for easy configuration of two-way SSL connections, based on peer trust where every trusted peer resides within a single JKS keystore. More elaborate configuration can be defined with alternate identity and trust managers to allow for CA based trust validation.

Elements

[Table A-56](#) describes the subelements you can define within the `socket-provider` element.

Table A–56 *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.
<tcp>	Optional	Specifies a socket provider that produces TCP-based sockets and channel implementations.
<ssl>	Optional	Specifies a socket provider that produces socket and channel implementations which utilize SSL.
<instance>	Optional	Contains the class configuration information for a <code>com.tangosol.net.SocketProvider</code> implementation.

socket-providers

Used in [cluster-config](#)

Description

The `socket-providers` element contains the declarative data for each socket provider implementation. Coherence includes the following pre-defined socket providers: `system`, `tcp`, and `ssl`.

Elements

[Table A-57](#) describes the subelements you can define within the `socket-providers` element.

Table A-57 *socket-providers Subelements*

Element	Required/ Optional	Description
<code><socket-provider></code>	Optional	Specifies the configuration information for a socket and channel factory that implements the <code>com.tangosol.net.SocketProvider</code> interface.

ssl

Used in: [socket-provider](#).

Description

The `<ssl>` element contains the configuration information for a socket provider that produces socket and channel implementations which utilize SSL. If SSL is configured for the unicast listener, the listener must be configured to use well known addresses.

Elements

[Table A-58](#) describes the elements you can define within the `ssl` element.

Table A-58 *ssl Subelements*

Element	Required/ Optional	Description
<code><protocol></code>	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.
<code><provider></code>	Optional	Specifies the configuration for a security provider instance.
<code><executor></code>	Optional	Specifies the configuration information for an implementation of the <code>java.util.concurrent.Executor</code> interface. A <code><class-name></code> subelement is used to provide the name of a class that implements the <code>Executor</code> interface. As an alternative, a <code><class-factory-name></code> subelement can be used to specify a factory class for creating <code>Executor</code> instances and a <code><method-name></code> subelement that specifies the name of a static factory method on the factory class which will perform object instantiation. Either approach can specify initialization parameters using the <code><init-params></code> element.
<code><identity-manager></code>	Optional	Specifies the configuration information for initializing an identity manager instance.
<code><trust-manager></code>	Optional	Specifies the configuration information for initializing a trust manager instance.
<code><hostname-verifier></code>	Optional	Specifies the configuration information for an implementation of the <code>javax.net.ssl.HostnameVerifier</code> interface. During the SSL handshake, if the URL's hostname and the server's identification hostname mismatch, the verification mechanism will call back to this instance to determine if the connection should be allowed. A <code><class-name></code> subelement is used to provide the name of a class that implements the <code>HostnameVerifier</code> interface. As an alternative, a <code><class-factory-name></code> subelement can be used to specify a factory class for creating <code>HostnameVerifier</code> instances and a <code><method-name></code> subelement that specifies the name of a static factory method on the factory class which will perform object instantiation. Either approach can specify initialization parameters using the <code><init-params></code> element.

tcp-ring-listener

Used in: [cluster-config](#).

Description

The TCP-ring provides a means for fast death detection of another node within the cluster. When enabled the cluster nodes form a single "ring" of TCP connections spanning the entire cluster. A cluster node is able to use the TCP connection to detect the death of another node within a heartbeat interval (default is one second; see the `<heartbeat-milliseconds>` subelement of [packet-delivery](#)). If disabled, the cluster node must rely on detecting that another node has stopped responding to UDP packets for a considerably 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-59](#) describes the subelements you can define within the `tcp-ring-listener` element.

Table A-59 *tcp-ring-listener Subelements*

Element	Required/ Optional	Description
<code><enabled></code>	Optional	Specifies whether the tcp ring listener should be enabled to detect node failures faster. Legal values are <code>true</code> and <code>false</code> . Default value is <code>true</code> . Preconfigured override is <code>tangosol.coherence.tcpring</code> . see Appendix C, "Command Line Overrides" for more information.
<code><ip-timeout></code>	Optional	<p>Specifies the timeout to use for determining that a machine hosting cluster members has become unreachable. A number of connection attempts may be made before determining that the unreachable members should be removed.</p> <p>The values of the <code><ip-timeout></code> and <code><ip-attempts></code> elements should be high enough to insulate against allowable temporary network outages.</p> <p>This feature relies upon the <code>java.net.InetAddress.isReachable</code> mechanism, see http://java.sun.com/j2se/1.5.0/docs/api/java/net/InetAddress.html#isReachable(int) for a description of how it will identify reachability.</p> <p>Legal values are strings representing time intervals. A timeout of 0 disables machine-level monitoring and is not recommended.</p> <p>The default value is 5s.</p>

Table A–59 (Cont.) tcp-ring-listener Subelements

Element	Required/ Optional	Description
<ip-attempts>	Optional	<p>specifies the number of connection attempts to make before determining that a machine hosting cluster members has become unreachable, and that those cluster members should be removed.</p> <p>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.</p> <p>The default value is 3.</p>
<listen-backlog>	Optional	<p>Specifies the size of the TCP/IP server socket backlog queue. Valid values are positive integers.</p> <p>Default value is O/S dependent.</p>
<priority>	Required	<p>Specifies a priority of the tcp ring listener execution thread. Legal values are from 1 to 10. Default value is 6.</p>

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See ["Element Attributes"](#) on page A-88 for more information on this attribute.

traffic-jam

Used in: [packet-publisher](#).

Description

The `traffic-jam` element is used to control the rate at which client threads enqueue packets for the Packet publisher to transmit on the network. When the limit is exceeded any client thread will be forced to pause until the number of outstanding packets drops below the specified limit. To limit the rate at which the Publisher transmits packets see the [flow-control](#) element.

Tuning

Specifying a limit which is too low, or a pause which is too long may result in the publisher transmitting all pending packets, and being left without packets to send. An ideal value will ensure that the publisher is never left without work to do, but at the same time prevent the queue from growing uncontrollably. It is therefore recommended that the pause remain quite short (10ms or under), and that the limit on the number of packets be kept high (that is, greater than 5000). As of Coherence 3.2 a warning will be periodically logged if this condition is detected.

Traffic Jam and Flow Control

When [flow-control](#) is enabled the `traffic-jam` operates in a point-to-point mode, only blocking a send if the recipient has too many packets outstanding. It is recommended that the `traffic-jam/maximum-packets` value be greater than the value (see the `<maximum-packets>` subelement of [outstanding-packets](#)). When `flow-control` is disabled, the `traffic-jam` will take all outstanding packets into account.

Elements

[Table A-60](#) describes the subelements you can define within the `traffic-jam` element.

Table A-60 *traffic-jam Subelements*

Element	Required/ Optional	Description
<code><maximum-packets></code>	Required	Specifies the maximum number of pending packets that the Publisher will tolerate before determining that it is clogged and must slow down client requests (requests from local non-system threads). Zero means no limit. This property prevents most unexpected out-of-memory conditions by limiting the size of the resend queue. Default value is 8192.
<code><pause-milliseconds></code>	Required	Number of milliseconds that the Publisher will pause a client thread that is trying to send a message when the Publisher is clogged. The Publisher will not allow the message to go through until the clog is gone, and will repeatedly sleep the thread for the duration specified by this property. Default value is 10.

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 will contain at least one child element.

Elements

[Table A-61](#) describes the elements you can define within the `trust-manager` element.

Table A-61 *trust-manager Subelements*

Element	Required/ Optional	Description
<code><algorithm></code>	Optional	Specifies the algorithm used by the trust manager. The default value is <code>SunX509</code> .
<code><provider></code>	Optional	Specifies the configuration for a security provider instance.
<code><key-store></code>	Optional	Specifies the configuration for a key store implementation.

unicast-listener

Used in: [cluster-config](#).

Description

Specifies the configuration information for the Unicast listener. This element is used to specify the address and port that a cluster node will bind to, to listen for point-to-point cluster communications.

Automatic Address Settings

By default Coherence will attempt to obtain the IP to bind to using the `java.net.InetAddress.getLocalHost()` call. On machines with multiple IPs or NICs you may need to explicitly specify the address (see the `<address>` subelement). Additionally if the specified port is already in use, Coherence will by default auto increment the port number until the binding succeeds (see the `<port>` and `<auto>` subelements).

Multicast-Free Clustering

By default Coherence uses a multicast protocol to discover other nodes when forming a cluster. If multicast networking is undesirable, or unavailable in your environment, the `well-known-addresses` feature may be used to eliminate the need for multicast traffic. If you are having difficulties in establishing a cluster by using multicast, see [Chapter 43, "Performing a Multicast Connectivity Test."](#)

Elements

[Table A-62](#) describes the subelements you can define within the `unicast-listener` element.

Table A-62 *unicast-listener Subelements*

Element	Required/ Optional	Description
<code><well-known-addresses></code>	Optional	Contains a list of "well known" addresses (WKA) that are used by the cluster discovery protocol in place of multicast broadcast.
<code><machine-id></code>	Optional	Specifies an identifier that should uniquely identify each server machine. If not specified, a default value is generated from the address of the default network interface. The machine id for each machine in the cluster can be used by cluster services to plan for failover by making sure that each member is backed up by a member running on a different machine.
<code><address></code>	Required	Specifies the IP address that a Socket will listen or publish on. Note: The localhost setting may not work on systems that define localhost as the loopback address; in that case, specify the machine name or the specific IP address. Also, the multicast listener, by default, binds to the same interface as defined by this address. Default value is <code>localhost</code> . Preconfigured override is <code>tangosol.coherence.localhost</code> . See Appendix C, "Command Line Overrides" for more information.

Table A–62 (Cont.) unicast-listener Subelements

Element	Required/ Optional	Description
<code><port></code>	Required	Specifies the ports that the Socket will listen or publish on. A second port is automatically opened and defaults to the next available port. Legal values are from 1 to 65535. Default value is 8088 for the first port and 8089 (if available) for the second port. Preconfigured override is <code>tangosol.coherence.localport</code> . See Appendix C, "Command Line Overrides" for more information.
<code><port-auto-adjust></code>	Required	Specifies whether the unicast port will be automatically incremented if the specified port cannot be bound to because it is already in use. Legal values are <code>true</code> or <code>false</code> . Default value is <code>true</code> . Preconfigured override is <code>tangosol.coherence.localport.adjust</code> . See Appendix C, "Command Line Overrides" for more information.
<code><packet-buffer></code>	Required	Specifies how many incoming packets the operating system will be requested to buffer. The value may be expressed either in terms of packets or bytes.
<code><priority></code>	Required	Specifies a priority of the unicast listener execution thread. Legal values are from 1 to 10. Default value is 8.
<code><socket-provider></code>	Optional	<p>Specifies the configuration information for a socket and channel factory that implements the <code>com.tangosol.net.SocketProvider</code> interface. The socket provider is used for TCMP communication. This element may also be used to refer to a socket provider configuration that is already defined within the <code><socket-providers></code> element. For example:</p> <pre><socket-provider>ssl</socket-provider></pre> <p>Preconfigured override is <code>tangosol.coherence.cluster.socketprovider</code>. See Appendix C, "Command Line Overrides" for more information.</p>

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See ["Element Attributes"](#) on page A-88 for more information on this attribute.

volume-threshold

Used in: [packet-speaker](#)

Description

Specifies the minimum outgoing packet volume which must exist for the speaker daemon to be activated.

Performance Impact

When the packet load is relatively low it may be more efficient for the speaker's operations to be performed on the publisher's thread. When the packet load is high using the speaker allows the publisher to continue preparing packets while the speaker transmits them on the network.

Elements

[Table A-63](#) describes the elements you can define within the `packet-speaker` element.

Table A-63 *packet-speaker Subelements*

Element	Required/ Optional	Description
<code><minimum-packets></code>	Required	Specifies the minimum number of packets which must be ready to be sent for the speaker daemon to be activated. A value of 0 will force the speaker to always be used, while a very high value will cause it to never be used. If unspecified, it will be set to match the packet-buffer , this is the default.

well-known-addresses

Used in: [unicast-listener](#).

Note: This is not a security-related feature, and does **not** limit the addresses which are allowed to join the cluster. See the [authorized-hosts](#) element for details on limiting cluster membership.

Use of the Well Known Addresses (WKA) feature is not supported by Standard Edition. If you are having difficulties in establishing a cluster by using multicast, see [Chapter 43, "Performing a Multicast Connectivity Test"](#).

Description

By default, Coherence uses a multicast protocol to discover other nodes when forming a cluster. If multicast networking is undesirable, or unavailable in your environment, the Well Known Addresses feature may be used to eliminate the need for multicast traffic. When in use the cluster is configured with a relatively small list of nodes which are allowed to start the cluster, and which are likely to remain available over the cluster lifetime. There is no requirement for all WKA nodes to be simultaneously active at any point in time. This list is used by all other nodes to find their way into the cluster without the use of multicast, thus at least one node that is configured as a well-known node must be running for other nodes to be able to join.

Example

[Example A-4](#) illustrates a configuration for two well-known-addresses with the default port.

Example A-4 Configuration for Two Well-Known-Addresses

```
<well-known-addresses>
  <socket-address id="1">
    <address>192.168.0.100</address>
    <port>8088</port>
  </socket-address>
  <socket-address id="2">
    <address>192.168.0.101</address>
    <port>8088</port>
  </socket-address>
</well-known-addresses>
```

Elements

[Table A-64](#) describes the subelements you can define within the `well-known-addresses` element.

Table A–64 *well-known-addresses Subelements*

Element	Required/ Optional	Description
<code><socket-address></code>	Optional	Specifies a list of WKA that are used by the cluster discovery protocol in place of multicast broadcast. If one or more WKA is specified, for a member to join the cluster it will either have to be a WKA or there will have to be at least one WKA member running. Additionally, all cluster communication will be performed using unicast. If empty or unspecified multicast communications will be used. Preconfigured overrides are <code>tangosol.coherence.wka</code> and <code>tangosol.coherence.wka.port</code> . See Appendix C, "Command Line Overrides" for more information.
<code><address-provider></code>	Optional	Contains the configuration for a <code>com.tangosol.util.AddressProvider</code> implementation that supplies the WKAs. The calling component will attempt to obtain the full list upon node startup, the provider must return a terminating <code>null</code> address to indicate that all available addresses have been returned.

The content override attribute `xml-override` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See ["Element Attributes"](#) on page A-88 for more information about this attribute.

Element Attributes

The optional `id` and `xml-override` attributes can be used to override the contents of an element. These attributes can appear, either individually or together, within the following elements:

Table A-65 lists the elements that can use `id` or `xml-override`, or both.

Table A-65 Elements that can use `id` or `xml-override`, or Both

<code>authorized-hosts</code>	<code>cluster-config</code>	<code>coherence</code>	<code>configurable-cache-factory-config</code>
<code>filter-name</code>	<code>filters</code>	<code>host-range</code>	<code>incoming-message-handler</code>
<code>init-param</code>	<code>logging-config</code>	<code>multicast-listener</code>	<code>packet-publisher</code>
<code>services</code>	<code>shutdown-listener</code>	<code>tcp-ring-listener</code>	<code>unicast-listener</code>

Table A-66 describes the functionality of the `id` and `xml-override` attributes.

Table A-66 `id` and `xml-override` Attribute Descriptions

Attribute	Required/ Optional	Description
<code>xml-override</code>	Optional	<p>Allows the content of this element to be fully or partially overridden with XML documents that are external to the base document. Legal value of this attribute is the resource name of such an override document that should be accessible using the <code>ClassLoader.getResourceAsStream(String name)</code> by the classes contained in <code>coherence.jar</code> library. In general that means that resource name should be prefixed with <code>'/'</code> and located in the classpath.</p> <p>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, <code><services></code>) the <code>id</code> attribute should be used to identify the base element that will be overridden and the override element itself. The elements of the override document that do not have a match in the base document are just appended to the base.</p>
<code>id</code>	Optional	<p>Used in conjunction with the <code>xml-override</code> attribute in cases where there are multiple elements with the same name (for example, <code><services></code>) to identify the base element that will be overridden and the override element itself. The elements of the override document that do not have a match in the base document are just appended to the base.</p>

Cache Configuration Elements

This appendix provides a reference of the elements that can be used in a cache configuration deployment descriptor and includes a brief overview of the descriptor. See [Chapter 11, "Configuring Caches,"](#) for details on how to configure caches and complete usage instructions.

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 (packaged in `coherence.jar`) is used unless a custom `coherence-cache-config.xml` file is found within the application's classpath. It is recommended that all nodes within a cluster use identical cache configuration descriptors.

The cache configuration deployment descriptor is defined in the `cache-config.dtd` file, which is located in the root of the `coherence.jar` library. The descriptor should begin with the following DOCTYPE declaration:

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

The root element of the configuration descriptor is the `<cache-config>` element. All caches are defined within the root element.

Note: When deploying Coherence into environments where the default character set is EBCDIC rather than ASCII, make sure that this descriptor file is in ASCII format and is deployed into its runtime environment in the binary format.

Element Reference

The following table lists all non-terminal elements which may be used within a cache configuration deployment descriptor.

Table B-1 *Cache Configuration Elements*

Element	Used In:
<code>acceptor-config</code>	<code>proxy-scheme</code>
<code>address-provider</code>	<code>tcp-acceptor</code> , <code>remote-addresses</code>
<code>async-store-manager</code>	<code>external-scheme</code> , <code>paged-external-scheme</code>
<code>authorized-hosts</code>	<code>tcp-acceptor</code>
<code>backing-map-scheme</code>	<code>distributed-scheme</code> , <code>optimistic-scheme</code> , <code>replicated-scheme</code>
<code>backup-storage</code>	<code>distributed-scheme</code>
<code>bdb-store-manager</code>	<code>external-scheme</code> , <code>paged-external-scheme</code> , <code>async-store-manager</code>
<code>cache-config</code>	<i>root element</i>
<code>cache-mapping</code>	<code>caching-scheme-mapping</code>
<code>cache-service-proxy</code>	<code>proxy-config</code>
<code>caching-scheme-mapping</code>	<code>cache-config</code>
<code>caching-schemes</code>	<code>cache-config</code>
<code>class-scheme</code>	<code>caching-schemes</code> , <code>local-scheme</code> , <code>distributed-scheme</code> , <code>replicated-scheme</code> , <code>optimistic-scheme</code> , <code>near-scheme</code> , <code>overflow-scheme</code> , <code>read-write-backing-map-scheme</code> , <code>cachestore-scheme</code> , <code>listener</code>
<code>cachestore-scheme</code>	<code>local-scheme</code> , <code>read-write-backing-map-scheme</code>
<code>custom-store-manager</code>	<code>external-scheme</code> , <code>paged-external-scheme</code> , <code>async-store-manager</code>
<code>defaults</code>	<code>cache-config</code>
<code>disk-scheme</code>	<code>caching-schemes</code>
<code>distributed-scheme</code>	<code>caching-schemes</code> , <code>near-scheme</code> , <code>overflow-scheme</code>
<code>external-scheme</code>	<code>caching-schemes</code> , <code>distributed-scheme</code> , <code>replicated-scheme</code> , <code>optimistic-scheme</code> , <code>near-scheme</code> , <code>overflow-scheme</code> , <code>read-write-backing-map-scheme</code>
<code>identity-manager</code>	<code>ssl</code>
<code>init-param</code>	<code>init-params</code>
<code>init-params</code>	<code>class-scheme</code>
<code>initiator-config</code>	<code>remote-cache-scheme</code> , <code>remote-invocation-scheme</code>
<code>instance</code>	<code>serializer</code> , <code>socket-provider</code> , <code>service-failure-policy</code>
<code>invocation-scheme</code>	<code>caching-schemes</code>
<code>jms-acceptor</code>	<code>acceptor-config</code>

Table B-1 (Cont.) Cache Configuration Elements

Element	Used In:
json-initiator	initiator-config
key-associator	distributed-scheme
key-partitioning	distributed-scheme
key-store	identity-manager, trust-manager
local-file-manager	external-scheme, paged-external-scheme, async-store-manager
listener	disk-scheme, local-scheme, external-scheme, paged-external-scheme, distributed-scheme, replicated-scheme, optimistic-scheme, near-scheme, overflow-scheme, read-write-backing-map-scheme
local-address	tcp-acceptor, tcp-initiator
local-scheme	caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme, near-scheme, overflow-scheme, read-write-backing-map-scheme
near-scheme	caching-schemes
nio-file-manager	external-scheme, paged-external-scheme, async-store-manager
nio-memory-manager	external-scheme, paged-external-scheme, async-store-manager
operation-bundling	cache-store-scheme, distributed-scheme, remote-cache-scheme
optimistic-scheme	caching-schemes, near-scheme, overflow-scheme
outgoing-message-handler	acceptor-config, initiator-config
overflow-scheme	caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme, read-write-backing-map-scheme
paged-external-scheme	caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme, near-scheme, overflow-scheme, read-write-backing-map-scheme
partitioned	backing-map-scheme
partitioned-quorum-policy-scheme	distributed-scheme
provider	identity-manager, ssl, trust-manager
proxy-config	proxy-scheme
proxy-scheme	caching-schemes
proxy-quorum-policy-scheme	proxy-scheme
read-write-backing-map-scheme	caching-schemes, distributed-scheme, replicated-scheme, optimistic-scheme
remote-addresses	tcp-initiator
remote-cache-scheme	cache-store-scheme, caching-schemes, near-scheme
remote-invocation-scheme	caching-schemes

Table B-1 (Cont.) Cache Configuration Elements

Element	Used In:
<code>replicated-scheme</code>	<code>caching-schemes</code> , <code>near-scheme</code> , <code>overflow-scheme</code>
<code>serializer</code>	<code>acceptor-config</code> , <code>defaults</code> , <code>distributed-scheme</code> , <code>initiator-config</code> , <code>invocation-scheme</code> , <code>optimistic-scheme</code> , <code>replicated-scheme</code> , <code>transactional-scheme</code>
<code>socket-address</code>	<code>remote-addresses</code>
<code>socket-provider</code>	<code>tcp-acceptor</code> , <code>tcp-initiator</code>
<code>ssl</code>	<code>socket-provider</code>
<code>tcp-acceptor</code>	<code>acceptor-config</code>
<code>tcp-initiator</code>	<code>initiator-config</code>
<code>transactional-scheme</code>	<code>caching-schemes</code>
<code>trust-manager</code>	<code>ssl</code>

acceptor-config

Used in: [proxy-scheme](#)

Description

The `acceptor-config` element specifies the configuration information for a TCP/IP connection acceptor. The connection acceptor is used by a proxy service to enable Coherence*Extend clients to connect to the cluster and use the services offered by the cluster without having to join the cluster.

Elements

[Table B-2](#) describes the elements you can define within the `acceptor-config` element.

Table B-2 *acceptor-config Subelements*

Element	Required/Optional	Description
<code><connection-limit></code>	Optional	The maximum number of simultaneous connections allowed by this connection acceptor. Valid values are positive integers and zero. A value of zero implies no limit. Default value is zero.
<code><outgoing-message-handler></code>	Optional	Specifies the configuration information used by the connection acceptor to detect dropped client-to-cluster connections.
<code><serializer></code>	Optional	Specifies the class configuration information for a <code>com.tangosol.io.Serializer</code> implementation used by the connection acceptor to serialize and deserialize user types. For example, the following configures a <code>ConfigurablePofContext</code> that uses the <code>my-pof-types.xml</code> POF type configuration file to deserialize user types to and from a POF stream: <pre> <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> </pre>
<code><tcp-acceptor></code>	Optional	Specifies the configuration information for a connection acceptor that enables Coherence*Extend clients to connect to the cluster over TCP/IP.
<code><use-filters></code>	Optional	Contains the list of filter names to be used by this connection acceptor. For example, specifying <code><use-filter></code> as follows will activate gzip compression for all network messages, which can help substantially with WAN and low-bandwidth networks. <pre> <use-filters> <filter-name>gzip</filter-name> </use-filters> </pre>

address-provider

Used in: [tcp-acceptor](#), [remote-addresses](#)

Description

Contains the configuration information for an address factory that implements the `com.tangosol.net.AddressProvider` interface.

Elements

[Table B-3](#) describes the subelements you can define within the `address-provider` element.

Table B-3 *address-provider Subelements*

Element	Required/ Optional	Description
<code><class-name></code>	Optional	<p>Specifies the fully qualified name of a class that implements the <code>com.tangosol.net.AddressProvider</code> interface.</p> <p>This element cannot be used together with the <code><class-factory-name></code> element.</p>
<code><class-factory-name></code>	Optional	<p>Specifies the fully qualified name of a factory class for creating address provider instances. The instances must implement the <code>com.tangosol.net.AddressProvider</code> interface.</p> <p>This element cannot be used together with the <code><class-name></code> element and is used together with the <code><method-name></code> element.</p>
<code><method-name></code>	Optional	<p>Specifies the name of a static factory method on the factory class which will perform object instantiation.</p>
<code><init-params></code>	Optional	<p>Specifies initialization parameters which are accessible by implementations which support the <code>com.tangosol.run.xml.XmlConfigurable</code> interface, or which include a public constructor with a matching signature. Initialization parameters can be specified for both the <code><class-name></code> element and the <code><class-factory-name></code> element.</p>

async-store-manager

Used in: [external-scheme](#), [paged-external-scheme](#).

Description

The `async-store-manager` element adds asynchronous write capabilities to other store manager implementations. Supported store managers include:

- [custom-store-manager](#)—allows definition of custom implementations of store managers
- [bdb-store-manager](#)—uses Berkeley Database JE to implement an on disk cache
- [lh-file-manager](#)—uses a Coherence LH on disk database cache
- [nio-file-manager](#)—uses NIO to implement memory-mapped file based cache
- [nio-memory-manager](#)—uses NIO to implement an off JVM heap, in-memory cache

Implementation

This store manager is implemented by the `com.tangosol.io.AsyncBinaryStoreManager` class.

Elements

[Table B-4](#) describes the subelements you can define within the `async-store-manager` element.

Table B-4 *async-store-manager Subelements*

Element	Required/Optional	Description
<code><async-limit></code>	Optional	Specifies the maximum number of bytes that will be queued to be written asynchronously. Setting the value to zero does not disable the asynchronous writes; instead, it indicates that the implementation default for the maximum number of bytes are necessary value of this element must be in the following format: $[\backslash d]^+ [[.] [\backslash d]^+] ? [K k M m] ? [B b] ?$ where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied: <ul style="list-style-type: none"> ■ K (kilo, 210) ■ M (mega, 220) If the value does not contain a factor, a factor of one is assumed. Valid values are any positive memory sizes and zero. Default value is 4MB.
<code><bdb-store-manager></code>	Optional	Configures the external cache to use Berkeley Database JE on disk databases for cache storage.
<code><class-name></code>	Optional	Specifies a custom implementation of the <code>async-store-manager</code> . Any custom implementation must extend the <code>com.tangosol.io.AsyncBinaryStoreManager</code> class and declare the exact same set of public constructors.
<code><custom-store-manager></code>	Optional	Configures the external cache to use a custom storage manager implementation.

Table B–4 (Cont.) *async-store-manager* Subelements

Element	Required/ Optional	Description
<code><init-params></code>	Optional	Specifies initialization parameters, for use in custom <i>async-store-manager</i> implementations which implement the <code>com.tangosol.run.xml.XmlConfigurable</code> interface.
<code><lh-file-manager></code>	Optional	Configures the external cache to use a Coherence LH on disk database for cache storage.
<code><nio-file-manager></code>	Optional	Configures the external cache to use a memory-mapped file for cache storage.
<code><nio-memory-manager></code>	Optional	Configures the external cache to use an off JVM heap, memory region for cache storage.

authorized-hosts

Used in: [tcp-acceptor](#).

Description

This element contains the collection of IP addresses of TCP/IP initiator hosts that are allowed to connect to the cluster using a TCP/IP acceptor. If this collection is empty no constraints are imposed. Any number of `host-address` and `host-range` elements may be specified.

Elements

[Table B-5](#) describes the subelements you can define within the `authorized-hosts` element.

Table B-5 *authorized-hosts Subelements*

Element	Required/ Optional	Description
<code><host-address></code>	Optional	Specifies an IP address or hostname. If any are specified, only hosts with specified host-addresses or within the specified host-ranges will be allowed to join the cluster. The content override attributes <code>id</code> can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document.
<code><host-range></code>	Optional	Specifies a range of IP addresses. If any are specified, only hosts with specified host-addresses or within the specified host-ranges will be allowed to join the cluster.
<code><host-filter></code>	Optional	Specifies class configuration information for a <code>com.tangosol.util.Filter</code> implementation that is used by a TCP/IP acceptor to determine whether to accept a particular TCP/IP initiator. The <code>evaluate()</code> method will be passed to the <code>java.net.InetAddress</code> of the client. Implementations should return <code>true</code> to allow the client to connect. Classes are specified using the <code><class-name></code> subelement. Any initialization parameters can be defined within an <code><init-params></code> subelement.

The content override attributes `xml-override` and `id` can be optionally used to fully or partially override the contents of this element with XML document that is external to the base document. See ["Element Attributes"](#) on page A-88.

backing-map-scheme

Used in: [distributed-scheme](#), [optimistic-scheme](#), [replicated-scheme](#)

Description

Specifies what type of cache will be used within the cache server to store the entries.

When using an overflow-based backing map, it is important that the corresponding `backup-storage` be configured for overflow (potentially using the same scheme as the `backing-map`). See ["Partitioned Cache with Overflow"](#) on page 15-6 for an example configuration.

Note: The `partitioned` subelement is used if and only if the parent element is the `distributed-scheme`.

Elements

[Table B-6](#) describes the subelements you can define within the `backing-map-scheme` element. :

Table B-6 *backing-map-scheme Subelements*

Element	Required/ Optional	Description
<partitioned>	Optional	Specifies whether the backing map itself is partitioned. It is respected only within a <code>distributed-scheme</code> . See Chapter 12, "Implementing Storage and Backing Maps."
<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 will dictate what class or interface(s) must be extended.
<external-scheme>	Optional	External schemes define caches which are not JVM heap based, allowing for greater storage capacity.
<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.
<paged-external-scheme>	Optional	As with external-scheme , <code>paged-external-schemes</code> define caches which are not JVM heap based, allowing for greater storage capacity.
<overflow-scheme>	Optional	The <code>overflow-scheme</code> defines a two-tier cache consisting of a fast, size limited front-tier, and slower but much higher capacity back-tier cache.
<read-write-backing-map-scheme>	Optional	The <code>read-write-backing-map-scheme</code> defines a backing map which provides a size limited cache of a persistent store.
<versioned-backing-map-scheme>	Optional	The <code>versioned-backing-map-scheme</code> is an extension of a read-write-backing-map-scheme , defining a size limited cache of a persistent store. It uses object versioning to determine what updates need to be written to the persistent store.

backup-storage

Used in: [distributed-scheme](#).

Description

The `backup-storage` element specifies the type and configuration of backup storage for a partitioned cache.

Elements

The following table describes the elements you can define within the `backup-storage` element.

Table B-7 *backup-storage Subelements*

Element	Required/Optional	Description
<code><type></code>	Required	<p>Specifies the type of the storage used to hold the backup data. Legal values are:</p> <ul style="list-style-type: none"> on-heap—The corresponding implementations class is <code>java.util.HashMap</code>. off-heap—The corresponding implementations class is <code>com.tangosol.io.nio.BinaryMap</code> using the <code>com.tangosol.io.nio.DirectBufferManager</code>. file-mapped—The corresponding implementations class is <code>com.tangosol.io.nio.BinaryMap</code> using the <code>com.tangosol.io.nio.MappedBufferManager</code>. custom—The corresponding implementations class is the class specified by the <code>class-name</code> element. scheme—The corresponding implementations class is specified as a caching-scheme by the <code>scheme-name</code> element. <p>Default value is the value specified in the <code>tangosol-coherence.xml</code> descriptor. For more information, see the <code><backup-storage/type></code> parameter in "DistributedCache Service Parameters" on page A-65.</p>
<code><class-name></code>	Optional	<p>Only applicable with the custom type. Specifies a class name for the custom storage implementation. If the class implements <code>com.tangosol.run.xml.XmlConfigurable</code> interface then upon construction, the <code>setConfig</code> method is called passing the entire <code>backup-storage</code> element. Default value is the <code>backup-storage/class-name</code> value specified in the <code>tangosol-coherence.xml</code> descriptor. See "DistributedCache Service Parameters" on page A-65 for more information.</p>
<code><directory></code>	Optional	<p>Only applicable with the file-mapped type. Specifies the path name for the directory that the disk persistence manager (<code>com.tangosol.util.nio.MappedBufferManager</code>) will use as "root" to store files in. If not specified or specifies a non-existent directory, a temporary file in the default location is used. Default value is the <code>backup-storage/directory</code> value specified in the <code>tangosol-coherence.xml</code> descriptor. See "DistributedCache Service Parameters" on page A-65 for more information.</p>

Table B–7 (Cont.) backup-storage Subelements

Element	Required/ Optional	Description
<initial-size>	Optional	<p>Only applicable with the off-heap and file-mapped types. Specifies the initial buffer size in bytes. The value of this element must be in the following format:</p> <pre>[\\d]+[\\.][\\d]]?[K k M m G g]?[B b]?</pre> <p>where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied:</p> <ul style="list-style-type: none"> ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) <p>If the value does not contain a factor, a factor of mega is assumed. Legal values are positive integers between 1 and <code>Integer.MAX_VALUE - 1023</code> (that is, 2,147,482,624 bytes). Default value is the <code>backup-storage/initial-size</code> value specified in the <code>tangosol-coherence.xml</code> descriptor. See "DistributedCache Service Parameters" on page A-65 for more information.</p>
<maximum-size>	Optional	<p>Only applicable with the off-heap and file-mapped types. Specifies the initial buffer size in bytes. The value of this element must be in the following format:</p> <pre>[\\d]+[\\.][\\d]]?[K k M m G g]?[B b]?</pre> <p>where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied:</p> <ul style="list-style-type: none"> ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) <p>If the value does not contain a factor, a factor of mega is assumed. Legal values are positive integers between 1 and <code>Integer.MAX_VALUE - 1023</code> (that is, 2,147,482,624 bytes). Default value is the <code>backup-storage/maximum-size</code> value specified in the <code>tangosol-coherence.xml</code> descriptor. See "DistributedCache Service Parameters" on page A-65 for more information.</p>
<scheme-name>	Optional	<p>Only applicable with the scheme type. Specifies a scheme name for the <code>ConfigurableCacheFactory</code>. Default value is the <code>backup-storage/scheme-name</code> value specified in the <code>tangosol-coherence.xml</code> descriptor. See "DistributedCache Service Parameters" on page A-65 for more information.</p>

bdb-store-manager

Used in: [external-scheme](#), [paged-external-scheme](#), [async-store-manager](#).

Note: Berkeley Database JE Java class libraries are required to use a bdb-store-manager, see the Berkeley Database JE product page for additional information.

<http://www.oracle.com/technology/documentation/berkeley-db/je/index.html>

Description

The BDB store manager is used to define external caches which will use Berkeley Database JE on disk embedded databases for storage. See the examples of Berkeley-based store configurations in "[Persistent Cache on Disk](#)" on page 15-3 and "[In-memory Cache with Disk Based Overflow](#)" on page 15-4.

Implementation

This store manager is implemented by the `com.tangosol.io.bdb.BerkeleyDBBinaryStoreManager` class, and produces `BinaryStore` objects implemented by the `com.tangosol.io.bdb.BerkeleyDBBinaryStore` class.

Elements

[Table B-8](#) describes the elements you can define within the bdb-store-manager element.

Table B-8 *bdb-store-manager Subelements*

Element	Required/Optional	Description
<code><class-name></code>	Optional	Specifies a custom implementation of the Berkeley Database <code>BinaryStoreManager</code> . Any custom implementation must extend the <code>com.tangosol.io.bdb.BerkeleyDBBinaryStoreManager</code> class and declare the exact same set of public constructors.

Table B–8 (Cont.) bdb-store-manager Subelements

Element	Required/ Optional	Description
<directory>	Optional	Specifies the path name to the root directory where the Berkeley Database JE store manager will store files. If not specified or specified with a non-existent directory, a temporary directory in the default location will be used.
<init-params>	Optional	<p>Specifies additional Berkeley DB configuration settings. See the Berkeley DB Configuration instructions:</p> <p>http://www.oracle.com/technology/documentation/berkeley-db/jc/GettingStartedGuide/administration.html#propertyfile</p> <p>Also used to specify initialization parameters, for use in custom implementations which implement the <code>com.tangosol.run.xml.XmlConfigurable</code> interface.</p>
<store-name>	Optional	<p>Specifies the name for a database table that the Berkeley Database JE store manager will use to store data in. Specifying this parameter will cause the <code>bdb-store-manager</code> 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.</p> <p>When specifying this property, it is recommended to use the <code>{cache-name}</code> macro. See "Using Parameter Macros" on page 11-12 for more information on the <code>{cache-name}</code> macro.</p>

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 you can define within the `bundle-config` element.

Table B-9 *bundle-config Subelements*

Element	Required/ Optional	Description
<code><auto-adjust></code>	Optional	Specifies whether the auto adjustment of the preferred-size value (based on the run-time statistics) is allowed. Valid values are <code>true</code> or <code>false</code> . Default value is <code>false</code> .
<code><delay-millis></code>	Optional	Specifies the maximum amount of time in milliseconds that individual execution requests are allowed to be deferred for a purpose of "bundling" them together and passing into a corresponding bulk operation. If the preferred-size threshold is reached before the specified delay, the bundle is processed immediately. Valid values are positive numbers. Default value is 1.
<code><operation-name></code>	Required	Specifies the operation name for which calls performed concurrently on multiple threads will be "bundled" into a functionally analogous "bulk" operation that takes a collection of arguments instead of a single one. Valid values depend on the bundle configuration context. For the <code><cachestore-scheme></code> the valid operations are: <ul style="list-style-type: none"> ▪ <code>load</code> ▪ <code>store</code> ▪ <code>erase</code> For the <code><distributed-scheme></code> and <code><remote-cache-scheme></code> the valid operations are: <ul style="list-style-type: none"> ▪ <code>get</code> ▪ <code>put</code> ▪ <code>remove</code> In all cases there is a pseudo operation named <code>all</code> , referring to all valid operations. Default value is <code>all</code> .
<code><preferred-size></code>	Optional	Specifies the bundle size threshold. When a bundle size reaches this value, the corresponding "bulk" operation will be invoked immediately. This value is measured in context-specific units. Valid values are zero (disabled bundling) or positive values. Default value is zero.
<code><thread-threshold></code>	Optional	Specifies the minimum number of threads that must be concurrently executing individual (non-bundled) requests for the bundler to switch from a pass-through to a bundling mode. Valid values are positive numbers. Default value is 4.

cache-config

Root Element

Description

The `cache-config` element is the root element of the cache configuration descriptor, `coherence-cache-config.xml`. For more information on this document, see ["Cache Configuration Deployment Descriptor"](#) on page B-1.

At a high level, a cache configuration consists of cache schemes and cache scheme mappings. Cache schemes describe a type of cache, for instance a database backed, distributed cache. Cache mappings define what scheme to use for a given cache name.

Elements

[Table B-10](#) describes the subelements you can define within the `cache-config` element.

Table B-10 *cache-config Subelements*

Element	Required/ Optional	Description
<code><caching-scheme-mapping></code>	Required	Specifies the caching-scheme that will be used for caches, based on the cache's name.
<code><caching-schemes></code>	Required	Defines the available caching-schemes for use in the cluster.
<code><defaults></code>	Optional	Defines factory wide default settings.

cache-mapping

Used in: [caching-scheme-mapping](#)

Description

Each cache-mapping element specifies the [caching-schemes](#) which are to be used for a given cache name or pattern.

Elements

[Table B-11](#) describes the subelements you can define within the cache-mapping element.

Table B-11 *cache-mapping Subelements*

Element	Required/ Optional	Description
<code><cache-name></code>	Required	<p>Specifies a cache name or name pattern. The name is unique within a cache factory. The following cache name patterns are supported:</p> <ul style="list-style-type: none"> ■ exact match, for example, <code>MyCache</code> ■ prefix match, for example, <code>My*</code> that matches to any cache name starting with <code>My</code> ■ any match <code>"*"</code>, that matches to any cache name <p>The patterns get matched in the order of specificity (more specific definition is selected whenever possible). For example, if both <code>MyCache</code> and <code>My*</code> mappings are specified, the scheme from the <code>MyCache</code> mapping will be used to configure a cache named <code>MyCache</code>.</p>
<code><scheme-name></code>	Required	<p>Contains the caching scheme name. The name is unique within a configuration file. Caching schemes are configured in the caching-schemes element.</p>
<code><init-params></code>	Optional	<p>Allows specifying replaceable cache scheme parameters. During cache scheme parsing, any occurrence of any replaceable parameter in format <code>param-name</code> is replaced with the corresponding parameter value. Consider the following cache mapping example:</p> <pre> <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> </pre> <p>For any cache name match <code>My*</code>, any occurrence of the literal <code>cache-loader</code> in any part of the corresponding <code>cache-scheme</code> element will be replaced with the string <code>com.acme.MyCacheLoader</code> and any occurrence of the literal <code>size-limit</code> will be replaced with the value of <code>1000</code>.</p>

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 elements you can define within the `cache-service-proxy` element.

Table B-12 *cache-service-proxy Subelements*

Element	Required/ Optional	Description
<code><enabled></code>	Optional	Specifies whether the cache service proxy is enabled. If disabled, clients will not be able to access any proxied caches. Legal values are <code>true</code> or <code>false</code> . Default value is <code>true</code> .
<code><lock-enabled></code>	Optional	Specifies whether lock requests from remote clients are permitted on a proxied cache. Legal values are <code>true</code> or <code>false</code> . Default value is <code>false</code> .
<code><read-only></code>	Optional	Specifies whether requests from remote clients that update a cache are prohibited on a proxied cache. Legal values are <code>true</code> or <code>false</code> . Default value is <code>false</code> .
<code><class-name></code>	Optional	Specifies the fully qualified name of a class that implements the <code>com.tangosol.net.CacheService</code> interface. The class acts as an interceptor between a client and a proxied cache service in order 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.
<code><init-params></code>	Optional	Contains initialization parameters for the <code>CacheService</code> implementation.

cachestore-scheme

Used in: [local-scheme](#), [read-write-backing-map-scheme](#), [versioned-backing-map-scheme](#).

Description

Cache store schemes define a mechanism for connecting a cache to a back-end data store. The cache store scheme may use any class implementing either the `com.tangosol.net.cache.CacheStore` or `com.tangosol.net.cache.CacheLoader` interfaces, where the former offers read-write capabilities, where the latter is read-only. Custom implementations of these interfaces may be produced to connect Coherence to various data stores. See ["Cache of a Database"](#) on page 15-4 for an example of using a `cachestore-scheme`.

Elements

[Table B-13](#) describes the elements you can define within the `cachestore-scheme` element.

Table B-13 *cachestore-scheme Subelements*

Element	Required/Optional	Description
<code><scheme-name></code>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<code><scheme-ref></code>	Optional	Specifies the name of another scheme to inherit from. See "Using Scheme Inheritance" on page 11-9.
<code><class-scheme></code>	Optional	Specifies the implementation of the cache store. The specified class must implement one of the following two interfaces. <ul style="list-style-type: none"> ■ <code>com.tangosol.net.cache.CacheStore</code>—for read-write support ■ <code>com.tangosol.net.cache.CacheLoader</code>—for read-only support
<code><remote-cache-scheme></code>	Optional	Configures the <code>cachestore-scheme</code> to use Coherence*Extend as its cache store implementation.
<code><operation-bundling></code>	Optional	Specifies the configuration information for a bundling strategy.

caching-scheme-mapping

Used in: [cache-config](#)

Description

Defines mappings between cache names, or name patterns, and [caching-schemes](#). For instance you may define that caches whose names start with `accounts-` will use a distributed ([distributed-scheme](#)) caching scheme, while caches starting with the name `rates-` will use 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 will use.

caching-schemes

Used in: [cache-config](#)

Description

The `caching-schemes` element defines a series of cache scheme elements. Each cache scheme defines a type of cache, for instance a database backed partitioned cache, or a local cache with an LRU eviction policy. Scheme types are bound to actual caches using 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
<code><local-scheme></code>	Optional	Defines a cache scheme which provides on-heap cache storage.
<code><external-scheme></code>	Optional	Defines a cache scheme which provides off-heap cache storage, for instance on disk.
<code><paged-external-scheme></code>	Optional	Defines a cache scheme which provides off-heap cache storage, that is size-limited by using time based paging.
<code><distributed-scheme></code>	Optional	Defines a cache scheme where storage of cache entries is partitioned across the cluster nodes.
<code><transactional-scheme></code>	Optional	Defines a cache scheme where storage of cache entries is partitioned across the cluster nodes with transactional guarantees.
<code><replicated-scheme></code>	Optional	Defines a cache scheme where each cache entry is stored on all cluster nodes.
<code><optimistic-scheme></code>	Optional	Defines a replicated cache scheme which uses optimistic rather than pessimistic locking.
<code><near-scheme></code>	Optional	Defines a two tier cache scheme which consists of a fast local front-tier cache of a much larger back-tier cache.
<code><versioned-near-scheme></code>	Optional	Defines a near-scheme which uses object versioning to ensure coherence between the front and back tiers.
<code><overflow-scheme></code>	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.
<code><invocation-scheme></code>	Optional	Defines an invocation service which can be used for performing custom operations in parallel across cluster nodes.
<code><read-write-backing-map-scheme></code>	Optional	Defines a backing map scheme which provides a cache of a persistent store.
<code><versioned-backing-map-scheme></code>	Optional	Defines a backing map scheme which uses object versioning to determine what updates need to be written to the persistent store.

Table B–15 (Cont.) caching-schemes Subelements

Element	Required/ Optional	Description
<code><remote-cache-scheme></code>	Optional	Defines a cache scheme that enables caches to be accessed from outside a Coherence cluster by using Coherence*Extend.
<code><class-scheme></code>	Optional	Defines a cache scheme using a custom cache implementation. Any custom implementation must implement the <code>java.util.Map</code> interface, and include a zero-parameter public constructor. Additionally if the contents of the Map can be modified by anything other than the <code>CacheService</code> itself (for example, if the Map automatically expires its entries periodically or size-limits its contents), then the returned object must implement the <code>com.tangosol.util.ObservableMap</code> interface.
<code><disk-scheme></code>	Optional	Note: As of Coherence 3.0, the disk-scheme configuration element has been deprecated and replaced by the external-scheme and paged-external-scheme configuration elements.

class-scheme

Used in: [caching-schemes](#), [local-scheme](#), [distributed-scheme](#), [replicated-scheme](#), [optimistic-scheme](#), [near-scheme](#), [versioned-near-scheme](#), [overflow-scheme](#), [read-write-backing-map-scheme](#), [versioned-backing-map-scheme](#), [cachestore-scheme](#), [listener](#), [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 will dictate what class or interface(s) must be extended. See ["Cache of a Database"](#) on page 15-4 for an example of using a `class-scheme`.

The `class-scheme` may be configured to either instantiate objects directly by using their `class-name`, or indirectly by using a `class-factory-name` and `method-name`. The `class-scheme` must be configured with either a `class-name` or `class-factory-name` and `method-name`.

Elements

[Table B-16](#) describes the elements you can define within the `class-scheme` element.

Table B-16 *class-scheme Subelements*

Element	Required/ Optional	Description
<code><scheme-name></code>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<code><scheme-ref></code>	Optional	Specifies the name of another scheme to inherit from. See "Using Scheme Inheritance" on page 11-9 for more information.
<code><class-name></code>	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.
<code><class-factory-name></code>	Optional	Specifies a fully specified name of a Java class that will be used as a factory for object instantiation.
<code><method-name></code>	Optional	Specifies the name of a static factory method on the factory class which will perform object instantiation.
<code><init-params></code>	Optional	Specifies initialization parameters which are accessible by implementations which support the <code>com.tangosol.run.xml.XmlConfigurable</code> interface, or which include a public constructor with a matching signature.

custom-store-manager

Used in: [external-scheme](#), [paged-external-scheme](#), [async-store-manager](#).

Description

Used to create and configure custom implementations of a store manager for use in external caches.

Elements

[Table B-17](#) describes the elements you can define within the `custom-store-manager` element.

Table B-17 *custom-store-manager Subelements*

Element	Required/ Optional	Description
<code><class-name></code>	Required	Specifies the implementation of the store manager. The specified class must implement the <code>com.tangosol.io.BinaryStoreManager</code> interface.
<code><init-params></code>	Optional	Specifies initialization parameters, for use in custom store manager implementations which implement the <code>com.tangosol.run.xml.XmlConfigurable</code> interface.

defaults

Used in: [cache-config](#)

Description

The `defaults` element defines factory wide default settings. This feature enables global configuration of serializers and socket providers used by all services which have not explicitly defined these settings.

Elements

[Table B-18](#) describes the elements you can define within the `defaults` element.

Table B-18 *defaults Subelements*

Element	Required/ Optional	Description
<code><serializer></code>	Optional	Specifies either: the class configuration information for a <code>com.tangosol.io.Serializer</code> implementation used to serialize and deserialize user types, or it references a serializer class configuration that is defined in the operational configuration file (see " serializers " on page A-61).
<code><socket-provider></code>		<p>Specifies either: the configuration information for a socket provider, or it references a pre-defined socket provider within the <code><socket-providers></code> element of the operational deployment descriptor. Pre-defined socket providers are referred to using their <code>id</code> attribute name: <code>system</code> or <code>ssl</code>. For example:</p> <pre><socket-provider>ssl</socket-provider></pre> <p>This setting only specifies the default socket provider for Coherence*Extend services. The TCMP socket provider is specified within the <code><unicast-listener></code> element in the operational configuration.</p>

disk-scheme

Note: As of Coherence 3.0, the disk-scheme configuration element has been deprecated and replaced with by the <[external-scheme](#)> and <[paged-external-scheme](#)> configuration elements.

distributed-scheme

Used in: [caching-schemes](#), [near-scheme](#), [versioned-near-scheme](#), [overflow-scheme](#), [versioned-backing-map-scheme](#)

Description

The `distributed-scheme` defines caches where the storage for entries is partitioned across cluster nodes. See "[Distributed Cache](#)" on page 10-1 for a more detailed description of partitioned caches. See "[Partitioned Cache](#)" on page 15-6 for examples of various `distributed-scheme` configurations.

Clustered Concurrency Control

Partitioned caches support cluster wide key-based locking so that data can be modified in a cluster without encountering the classic missing update problem. Note that any operation made without holding an explicit lock is still atomic but there is no guarantee that the value stored in the cache does not change between atomic operations.

Cache Clients

The partitioned cache service supports the concept of cluster nodes which do not contribute to the overall storage of the cluster. Nodes which are not storage enabled (see `<local-storage>` subelement) are considered "cache clients".

Cache Partitions

The cache entries are evenly segmented into several logical partitions (see `<partition-count>` subelement), and each storage enabled (see `<local-storage>` subelement) cluster node running the specified partitioned service (see `<service-name>` subelement) will be responsible for maintain a fair-share of these partitions.

Key Association

By default the specific set of entries assigned to each partition is transparent to the application. In some cases it may be advantageous to keep certain related entries within the same cluster node. A key-associator (see `<key-associator>` subelement) may be used to indicate related entries, the partitioned cache service will ensure that associated entries reside on the same partition, and thus on the same cluster node. Alternatively, key association may be specified from within the application code by using keys which implement the `com.tangosol.net.cache.KeyAssociation` interface.

Cache Storage (Backing Map)

Storage for the cache is specified by using the `backing-map-scheme` (see `<backing-map-scheme>` subelement). For instance a partitioned cache which uses a [local-scheme](#) for its backing map will result in cache entries being stored in-memory on the storage enabled cluster nodes.

Failover

For the purposes of failover, a 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 will not reside on the same physical machine as the primary partition. If a cluster node abruptly leaves the cluster, responsibility for its partitions will automatically be reassigned to the existing backups, and new backups of those partitions will be created (on remote nodes) to maintain the configured backup count.

Partition Redistribution

When a node joins or leaves the cluster, a background redistribution of partitions occurs to ensure that all cluster nodes manage a fair-share of the total number of partitions. The amount of bandwidth consumed by the background transfer of partitions is governed by the `transfer-threshold` (see `<transfer-threshold>` subelement).

Elements

[Table B-19](#) describes the elements you can define within the `distributed-scheme` element.

Table B-19 *distributed-scheme Subelements*

Element	Required/ Optional	Description
<code><scheme-name></code>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<code><scheme-ref></code>	Optional	Specifies the name of another scheme to inherit from. See "Using Scheme Inheritance" on page 11-9 for more information.
<code><service-name></code>	Optional	Specifies the name of the service which will manage caches created from this scheme. Services are configured in the <code><services></code> element in the <code>tangosol-coherence.xml</code> descriptor. See Appendix A, "Operational Configuration Elements" for more information.
<code><serializer></code>	Optional	Specifies either: the class configuration information for a <code>com.tangosol.io.Serializer</code> implementation used to serialize and deserialize user types, or it references a serializer class configuration that is defined in the operational configuration file (see "serializers" on page A-61).
<code><listener></code>	Optional	Specifies an implementation of a <code>com.tangosol.MapListener</code> which will be notified of events occurring on the cache.

Table B–19 (Cont.) distributed-scheme Subelements

Element	Required/ Optional	Description												
<backing-map-scheme>	Optional	<p>Specifies what type of cache will be used within the cache server to store the entries.</p> <p>Legal schemes are:</p> <ul style="list-style-type: none">▪ local-scheme▪ external-scheme▪ paged-external-scheme▪ class-scheme▪ overflow-scheme▪ read-write-backing-map-scheme▪ versioned-backing-map-scheme <p>Note that when using an off-heap backing map it is important that the corresponding <backup-storage> be configured for off-heap (potentially using the same scheme as the backing-map). Here off-heap refers to any storage where some or all entries are stored outside of the JVMs garbage collected heap space. Examples include: <overflow-scheme> and <external-scheme>. See "Partitioned Cache with Overflow" on page 15-6 for an example configuration.</p>												
<partition-count>	Optional	<p>Specifies the number of partitions that a partitioned (distributed) cache will be "chopped up" into. Each member running the partitioned cache service that has the local-storage (<local-storage> subelement) option set to true will manage a "fair" (balanced) number of partitions.</p> <p>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 in size.</p> <p>The following are good defaults based on service storage sizes:</p> <table><tr><th>service storage</th><th>partition-count</th></tr><tr><td>100M</td><td>257</td></tr><tr><td>1G</td><td>509</td></tr><tr><td>10G</td><td>2039</td></tr><tr><td>50G</td><td>4093</td></tr><tr><td>100G</td><td>8191</td></tr></table> <p>A list of first 1,000 primes can be found at http://primes.utm.edu/lists/</p> <p>Valid values are positive integers. The default value is 257 as specified in the tangosol-coherence.xml descriptor. See the partition-count parameter in "DistributedCache Service Parameters" on page A-65.</p>	service storage	partition-count	100M	257	1G	509	10G	2039	50G	4093	100G	8191
service storage	partition-count													
100M	257													
1G	509													
10G	2039													
50G	4093													
100G	8191													
<key-associator>	Optional	<p>Specifies a class that will be responsible for providing associations between keys and allowing associated keys to reside on the same partition. This implementation must have a zero-parameter public constructor.</p>												

Table B–19 (Cont.) distributed-scheme Subelements

Element	Required/ Optional	Description
<code><key-partitioning></code>	Optional	Specifies a class that implements the <code>com.tangosol.net.partition.KeyPartitioningStrategy</code> interface, which will be responsible for assigning keys to partitions. This implementation must have a zero-parameter public constructor. If unspecified, the default key partitioning algorithm will be used, which ensures that keys are evenly segmented across partitions.
<code><partition-listener></code>	Optional	Specifies a class that implements the <code>com.tangosol.net.partition.PartitionListener</code> interface.
<code><backup-count></code>	Optional	Specifies the number of members of the partitioned cache service that hold the backup data for each unit of storage in the cache. Value of 0 means that in the case of abnormal termination, some portion of the data in the cache will be lost. Value of N means that if up to N cluster nodes terminate immediately, the cache data will be preserved. To maintain the partitioned cache of size M, the total memory usage in the cluster does not depend on the number of cluster nodes and will be in the order of $M*(N+1)$. Recommended values are 0 or 1. Default value is the <code>backup-count</code> value specified in the <code>tangosol-coherence.xml</code> descriptor. See the <code>backup-count</code> parameter in value specified in the <code>tangosol-coherence.xml</code> descriptor. See "DistributedCache Service Parameters" on page A-65 for more information.
<code><backup-count-after-writebehind></code>	Optional	<p>Specifies the number of members of the partitioned cache service that will hold the backup data for each unit of storage in the cache that does <i>not</i> require write-behind, that is, data that is not vulnerable to being lost even if the entire cluster were shut down. Specifically, if a unit of storage is marked as requiring write-behind, then it will be backed up on the number of members specified by the <code><backup-count></code> subelement, and if the unit of storage is not marked as requiring write-behind, then it will be backed up by the number of members specified by the <code><backup-count-after-writebehind></code> element.</p> <p>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 will be discarded. However, until write-behind occurs, the data will be backed up in accordance with the <code><backup-count></code> setting.</p> <p>Recommended value is 0 or this element should be omitted altogether.</p>
<code><backup-storage></code>	Optional	Specifies the type and configuration for the partitioned cache backup storage.
<code><thread-count></code>	Optional	Specifies the number of daemon threads used by the partitioned cache service. If zero, all relevant tasks are performed on the service thread. Legal values are positive integers or zero. Default value is the <code>thread-count</code> value specified in the <code>tangosol-coherence.xml</code> descriptor. See the <code>thread-count</code> parameter in "DistributedCache Service Parameters" on page A-65 for more information.

Table B–19 (Cont.) distributed-scheme Subelements

Element	Required/ Optional	Description
<lease-granularity>	Optional	<p>Specifies the lease ownership granularity. Available since release 2.3. Legal values are:</p> <ul style="list-style-type: none"> ■ thread ■ member <p>A value of thread means that locks are held by a thread that obtained them and can only be released by that thread. A value of member means that locks are held by a cluster node and any thread running on the cluster node that obtained the lock can release it. Default value is the lease-granularity value specified in the tangosol-coherence.xml descriptor. See the lease-granularity parameter in "DistributedCache Service Parameters" on page A-65 for more information.</p>
<transfer-threshold>	Optional	<p>Specifies the threshold for the primary buckets distribution in kilo-bytes. When a new node joins the partitioned cache service or when a member of the service leaves, the remaining nodes perform a task of bucket ownership re-distribution. During this process, the existing data gets re-balanced along with the ownership information. This parameter indicates a preferred message size for data transfer communications. Setting this value lower will make the distribution process take longer, but will reduce network bandwidth utilization during this activity. Legal values are integers greater than zero. Default value is the transfer-threshold value specified in the tangosol-coherence.xml descriptor. See the transfer-threshold parameter in "DistributedCache Service Parameters" on page A-65 for more information.</p>
<local-storage>	Optional	<p>Specifies whether a cluster node will contribute storage to the cluster, that is, maintain partitions. When disabled the node is considered a cache client.</p> <p>Normally this value should be left unspecified within the configuration file, and instead set on a per-process basis using the tangosol.coherence.distributed.localstorage system property. This allows cache clients and servers to use the same configuration descriptor.</p> <p>Legal values are true or false. Default value is the local-storage value specified in the tangosol-coherence.xml descriptor. See the local-storage parameter in "DistributedCache Service Parameters" on page A-65 for more information.</p>
<autostart>	Optional	<p>The autostart element is intended to be used by cache servers (that is, com.tangosol.net.DefaultCacheServer). It specifies whether the cache services associated with this cache scheme should be automatically started at a cluster node. Legal values are true or false. Default value is false.</p>
<task-hung-threshold>	Optional	<p>Specifies the amount of time in milliseconds that a task can execute before it is considered "hung". Note: a posted task that has not yet started is never considered as hung. This attribute is applied only if the Thread pool is used (the thread-count value is positive). Legal values are positive integers or zero (indicating no default timeout). Default value is the task-hung-threshold value specified in the tangosol-coherence.xml descriptor. See the task-hung-threshold parameter in "DistributedCache Service Parameters" on page A-65 for more information.</p>

Table B–19 (Cont.) distributed-scheme Subelements

Element	Required/ Optional	Description
<task-timeout>	Optional	Specifies the timeout value in milliseconds for requests executing on the service worker threads. This attribute is applied only if the thread pool is used (the <code>thread-count</code> value is positive). If zero is specified, the default service-guardian <timeout-milliseconds> value is used. Legal values are non-negative integers. Default value is the value specified in the <code>tangosol-coherence.xml</code> descriptor. See the <code>task-timeout</code> parameter in " DistributedCache Service Parameters " on page A-65.
<request-timeout>	Optional	<p>Specifies the maximum amount of time a client will wait for a response before abandoning the original request. The request time is measured on the client side as the time elapsed from the moment a request is sent for execution to the corresponding server node(s) and includes the following:</p> <ul style="list-style-type: none"> 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 <p>Legal values are positive integers or zero (indicating no default timeout). Default value is the value specified in the <code>tangosol-coherence.xml</code> descriptor. See the <code>request-timeout</code> parameter in "DistributedCache Service Parameters" on page A-65 for more information.</p>
<guardian-timeout>	Optional	<p>Specifies the guardian timeout value to use for guarding the service and any dependant 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>.</p> <p>The value of this element must be in the following format:</p> <pre>[\d] + [[.] [\d] +] ? [MS ms S s M m H h D d] ?</pre> <p>where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> MS or ms (milliseconds) S or s (seconds) M or m (minutes) H or h (hours) D or d (days) <p>If the value does not contain a unit, a unit of milliseconds is assumed.</p>

Table B–19 (Cont.) distributed-scheme Subelements

Element	Required/ Optional	Description
<code><service-failure-policy></code>	Optional	<p>Specifies the action to take when an abnormally behaving service thread cannot be terminated gracefully by the service guardian.</p> <p>Legal values are:</p> <ul style="list-style-type: none"> ■ <code>exit-cluster</code> – 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. ■ <code>exit-process</code> – 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. ■ <code>logging</code> – 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 <code>com.tangosol.net.ServiceFailurePolicy</code> implementation. <p>Default value is <code>exit-cluster</code>.</p>
<code><member-listener></code>	Optional	<p>Specifies the configuration information for a class that implements the <code>com.tangosol.net.MemberListener</code> interface. The implementation must have a public default constructor.</p> <p>The <code>MemberListener</code> implementation receives cache service lifecycle events. The <code><member-listener></code> element is used as an alternative to programmatically adding a <code>MapListener</code> on a service.</p>
<code><operation-bundling></code>	Optional	Specifies the configuration information for a bundling strategy.
<code><partitioned-quorum-policy-scheme></code>	Optional	Specifies quorum policy settings for the partitioned cache service.

external-scheme

Used in: [caching-schemes](#), [distributed-scheme](#), [replicated-scheme](#), [optimistic-scheme](#), [near-scheme](#), [versioned-near-scheme](#), [overflow-scheme](#), [read-write-backing-map-scheme](#), [versioned-backing-map-scheme](#)

Description

External schemes define caches which are not JVM heap based, allowing for greater storage capacity. See "[Local Caches \(accessible from a single JVM\)](#)" on page 15-1 for examples of various external cache configurations.

Implementation

This scheme is implemented by:

- `com.tangosol.net.cache.SerializationMap`—for unlimited size caches
- `com.tangosol.net.cache.SerializationCache`—for size limited caches

The implementation type is chosen based on the following rule:

- if the `<high-units>` subelement is specified and not zero then `SerializationCache` is used;
- otherwise `SerializationMap` is used.

Pluggable Storage Manager

External schemes use a pluggable store manager to store and retrieve binary key value pairs. Supported store managers include:

- a wrapper providing asynchronous write capabilities for other store manager implementations
- allows definition of custom implementations of store managers
- uses Berkeley Database JE to implement an on disk cache
- uses a Coherence LH on disk database cache
- uses NIO to implement memory-mapped file based cache
- uses NIO to implement an off JVM heap, in-memory cache

Size Limited Cache

The cache may be configured as size-limited, which means that when it reaches its maximum allowable size (that is, the `<high-units>` subelement) it prunes itself.

Note: Eviction against disk-based caches can be expensive, consider using a [paged-external-scheme](#) for such cases.

Entry Expiration

External schemes support automatic expiration of entries based on the age of the value, as configured by the `<expiry-delay>` subelement.

Persistence (long-term storage)

External caches are generally used for temporary storage of large data sets, for example as the back-tier of an [overflow-scheme](#). Certain implementations do however support persistence for non-clustered caches, see the `<store-name>` subelement of [bdb-store-manager](#) and the `<manager-filename>` subelement of [lh-file-manager](#) for details. Clustered persistence should be configured by using a [read-write-backing-map-scheme](#) on a [distributed-scheme](#).

Elements

[Table B-20](#) describes the elements you can define within the `external-scheme` element.

Table B-20 *external-scheme Subelements*

Element	Required/ Optional	Description
<code><scheme-name></code>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<code><scheme-ref></code>	Optional	Specifies the name of another scheme to inherit from. See "Using Scheme Inheritance" on page 11-9 for more information
<code><class-name></code>	Optional	<p>Specifies a custom implementation of the external cache. Any custom implementation must extend one of the following classes:</p> <ul style="list-style-type: none"> ■ <code>com.tangosol.net.cache.SerializationCache</code>—for size limited caches ■ <code>com.tangosol.net.cache.SerializationMap</code>—for unlimited size caches ■ <code>com.tangosol.net.cache.SimpleSerializationMap</code>—for unlimited size caches <p>and declare the exact same set of public constructors as the superclass.</p>
<code><init-params></code>	Optional	Specifies initialization parameters, for use in custom external cache implementations which implement the <code>com.tangosol.run.xml.XmlConfigurable</code> interface.
<code><listener></code>	Optional	Specifies an implementation of a <code>com.tangosol.util.MapListener</code> which will be notified of events occurring on the cache.
<code><high-units></code>	Optional	Used to limit the size of the cache. Contains the maximum number of units that can be placed in the cache before pruning occurs. An entry is the unit of measurement. When this limit is exceeded, the cache will begin the pruning process, evicting the least recently used entries until the number of units is brought below this limit. The scheme's <code>class-name</code> element may be used to provide custom extensions to <code>SerializationCache</code> , which implement alternative eviction policies. Legal values are positive integers or zero. Zero implies no limit. Default value is zero.

Table B-20 (Cont.) external-scheme Subelements

Element	Required/ Optional	Description
<unit-calculator>	Optional	<p>Specifies the type of unit calculator to use. A unit calculator is used to determine the cost (in "units") of a given object. Legal values are:</p> <ul style="list-style-type: none"> ■ FIXED—A unit calculator that assigns an equal weight of 1 to all cached objects. ■ BINARY—A unit calculator that assigns an object a weight equal to the number of bytes of memory required to cache the object. This requires that the objects are <code>com.tangosol.util.Binary</code> instances, as in a Partitioned cache. See <code>com.tangosol.net.cache.BinaryMemoryCalculator</code> for additional details. ■ <class-scheme>—A custom unit calculator, specified as a class-scheme. The class specified within this scheme must implement the <code>com.tangosol.net.cache.ConfigurableCacheMap.UnitCalculator</code> interface. <p>This element is used only if the <code>high-units</code> element is set to a positive number. Default value is FIXED.</p>
<unit-factor>	Optional	<p>The unit-factor element specifies the factor by which the units, low-units and high-units properties are adjusted. Using a BINARY unit calculator, for example, the factor of 1048576 could be used to count megabytes instead of bytes.</p> <p>Using a BINARY unit calculator, for example, the factor of 1048576 could be used to count megabytes instead of bytes.</p> <p>Note: This element was introduced only to avoid changing the type of the units, low units and high units properties from 32-bit values to 64-bit values and is used only if the <code>high-units</code> element is set to a positive number. It is expected that this element will be dropped in a future release.</p> <p>Valid values are positive integer numbers. Default value is 1.</p>
<expiry-delay>	Optional	<p>Specifies the amount of time since the last update that entries are kept by the cache before being expired. Entries that have expired are not be accessible and are evicted the next time a client accesses the cache.</p> <p>The value of this element must be in the following format:</p> <pre>[\d] + [[.] [\d] +] ? [MS ms S s M m H h D d] ?</pre> <p>where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) <p>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.</p> <p>Note: The expiry delay parameter (<code>cExpiryMillis</code>) is defined as an integer and is expressed in milliseconds. Therefore, the maximum amount of time can never exceed <code>Integer.MAX_VALUE</code> (2147483647) milliseconds or approximately 24 days.</p>

Table B–20 (Cont.) external-scheme Subelements

Element	Required/ Optional	Description
<code><async-store-manager></code>	Optional	Configures the external cache to use an asynchronous storage manager wrapper for any other storage manager. See "Pluggable Storage Manager" on page B-34
<code><custom-store-manager></code>	Optional	Configures the external cache to use a custom storage manager implementation.
<code><bdb-store-manager></code>	Optional	Configures the external cache to use Berkeley Database JE on disk databases for cache storage.
<code><lh-file-manager></code>	Optional	Configures the external cache to use a Coherence LH on disk database for cache storage.
<code><nio-file-manager></code>	Optional	Configures the external cache to use a memory-mapped file for cache storage.
<code><nio-memory-manager></code>	Optional	Configures the external cache to use an off JVM heap, memory region for cache storage.

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 is unable to present authentication credentials.

Elements

[Table B-21](#) describes the elements you can define within the `identity-manager` element.

Table B-21 *identity-manager Subelements*

Element	Required/ Optional	Description
<code><algorithm></code>	Optional	Specifies the algorithm used by the identity manager. The default value is <code>SunX509</code> .
<code><provider></code>	Optional	Specifies the configuration for a security provider instance.
<code><key-store></code>	Optional	Specifies the configuration for a key store implementation.
<code><password></code>	Required	Specifies the private key password.

initiator-config

Used in: [remote-cache-scheme](#), [remote-invocation-scheme](#).

Description

The `initiator-config` element specifies the configuration information for a TCP/IP connection initiator. A connection initiator allows a Coherence*Extend client to connect to a cluster (by using a connection acceptor) and use the clustered services offered by the cluster without having to first join the cluster.

Elements

[Table B-22](#) describes the elements you can define within the `initiator-config` element.

Table B-22 *initiator-config Subelements*

Element	Required/ Optional	Description
<code><outgoing-message-handler></code>	Optional	Specifies the configuration information used by the connection initiator to detect dropped client-to-cluster connections.
<code><serializer></code>	Optional	Specifies either: the class configuration information for a <code>com.tangosol.io.Serializer</code> implementation used to serialize and deserialize user types, or it references a serializer class configuration that is defined in the operational configuration file (see " serializers " on page A-61).
<code><tcp-initiator></code>	Optional	Specifies the configuration information for a connection initiator that connects to the cluster over TCP/IP.
<code><use-filters></code>	Optional	Contains the list of filter names to be used by this connection initiator. In the following example, specifying <code><use-filter></code> will activate gzip compression for all network messages, which can help substantially with WAN and low-bandwidth networks. <pre> <use-filters> <filter-name>gzip</filter-name> </use-filters> </pre>

init-param

Used in: [init-params](#).

Defines an individual initialization parameter.

Elements

[Table B–23](#) describes the elements you can define within the `init-param` element.

Table B–23 *init-param* Subelements

Element	Required/ Optional	Description
<code><param-name></code>	Optional	Contains the name of the initialization parameter. For example: <pre><class-name>com.mycompany.cache.CustomCacheLoader</class-name> <init-params> <init-param> <param-name>sTableName</param-name> <param-value>EmployeeTable</param-value> </init-param> <init-param> <param-name>iMaxSize</param-name> <param-value>2000</param-value> </init-param> </init-params></pre>
<code><param-type></code>	Optional	Contains the Java type of the initialization parameter. The following standard types are supported: <ul style="list-style-type: none">▪ <code>java.lang.String</code> (a.k.a. <code>string</code>)▪ <code>java.lang.Boolean</code> (a.k.a. <code>boolean</code>)▪ <code>java.lang.Integer</code> (a.k.a. <code>int</code>)▪ <code>java.lang.Long</code> (a.k.a. <code>long</code>)▪ <code>java.lang.Double</code> (a.k.a. <code>double</code>)▪ <code>java.math.BigDecimal</code>▪ <code>java.io.File</code>▪ <code>java.sql.Date</code>▪ <code>java.sql.Time</code>▪ <code>java.sql.Timestamp</code> For example: <pre><class-name>com.mycompany.cache.CustomCacheLoader</class-name> <init-params> <init-param> <param-type>java.lang.String</param-type> <param-value>EmployeeTable</param-value> </init-param> <init-param> <param-type>int</param-type> <param-value>2000</param-value> </init-param> </init-params></pre>
<code><param-value></code>	Optional	Contains the value of the initialization parameter. The value is in the format specific to the Java type of the parameter.

init-params

Used in: [class-scheme](#), [cache-mapping](#).

Description

Defines a series of initialization parameters as name-value pairs. See "[Partitioned Cache of a Database](#)" on page 15-6 for an example of using `init-params`.

Elements

[Table B-24](#) describes the elements you can define within the `init-params` element.

Table B-24 *init-params* Subelements

Element	Required/ Optional	Description
<code><init-param></code>	Optional	Defines an individual initialization parameter.

instance

Used in: [serializer](#), [socket-provider](#), [service-failure-policy](#)

Description

The `<instance>` element contains the configuration of an implementation class or class factory that is used to plug in custom functionality.

Elements

[Table B–25](#) describes the elements you can define within the `instance` element.

Table B–25 *instance Subelements*

Element	Required/ Optional	Description
<code><class-name></code>	Optional	Specifies the fully qualified name of an implementation class. This element cannot be used together with the <code><class-factory-name></code> element.
<code><class-factory-name></code>	Optional	Specifies the fully qualified name of a factory class for creating implementation class instances. This element cannot be used together with the <code><class-name></code> element and is used together with the <code><method-name></code> element.
<code><method-name></code>	Optional	Specifies the name of a static factory method on the factory class which will perform object instantiation.
<code><init-params></code>	Optional	Contains class initialization parameters for the implementation class.

invocation-scheme

Used in: [caching-schemes](#).

Description

Defines an Invocation Service. The invocation service may be used to perform custom operations in parallel on any number of cluster nodes. See the `com.tangosol.net.InvocationService` API for additional details.

Elements

The following table describes the elements you can define within the invocation-scheme element.

Table B–26 *invocation-scheme Subelements*

Element	Required/ Optional	Description
<scheme-name>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<scheme-ref>	Optional	Specifies the name of another scheme to inherit from. See "Using Scheme Inheritance" on page 11-9 for more information.
<service-name>	Optional	Specifies the name of the service which will manage invocations from this scheme.
<serializer>	Optional	Specifies either: the class configuration information for a <code>com.tangosol.io.Serializer</code> implementation used to serialize and deserialize user types, or it references a serializer class configuration that is defined in the operational configuration file (see "serializers" on page A-61).
<thread-count>	Optional	Specifies the number of daemon threads used by the invocation service. If zero, all relevant tasks are performed on the service thread. Legal values are positive integers or zero. Default value is the <code>thread-count</code> value specified in the <code>tangosol-coherence.xml</code> descriptor. See the <code>thread-count</code> parameter in "InvocationService Parameters" on page A-70.
<autostart>	Optional	The <code>autostart</code> element is intended to be used by cache servers (that is, <code>com.tangosol.net.DefaultCacheServer</code>). It specifies whether this service should be automatically started at a cluster node. Legal values are <code>true</code> or <code>false</code> . Default value is <code>false</code> .
<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 <code>thread-count</code> value is positive). Legal values are positive integers or zero (indicating no default timeout). Default value is the <code>task-hung-threshold</code> value specified in the <code>tangosol-coherence.xml</code> descriptor. See the <code>task-hung-threshold</code> parameter in "InvocationService Parameters" on page A-70.

Table B–26 (Cont.) invocation-scheme Subelements

Element	Required/ Optional	Description
<task-timeout>	Optional	<p>Specifies the default timeout value in milliseconds for tasks that can be timed-out (for example, implement the <code>com.tangosol.net.PriorityTask</code> interface), but don't explicitly specify the task execution timeout value. The task execution time is measured on the server side and does not include the time spent waiting in a service backlog queue before being started. This attribute is applied only if the thread pool is used (the <code>thread-count</code> value is positive). If zero is specified, the default service-guardian <code><timeout-milliseconds></code> value is used. Legal values are non-negative integers. Default value is the <code>task-timeout</code> value specified in the <code>tangosol-coherence.xml</code> descriptor. See the <code>task-timeout</code> parameter in "InvocationService Parameters" on page A-70.</p>
<guardian-timeout>	Optional	<p>Specifies the guardian timeout value to use for guarding the service and any dependant threads. If the element is not specified for a given service, the default guardian timeout (as specified by the <code><timeout-milliseconds></code> operational configuration element) is used. See <service-guardian>.</p> <p>The value of this element must be in the following format:</p> <pre>[\\d]+[\\.][\\d]+]?[MS ms S s M m H h D d]?</pre> <p>where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) <p>If the value does not contain a unit, a unit of milliseconds is assumed.</p>

Table B–26 (Cont.) invocation-scheme Subelements

Element	Required/ Optional	Description
<code><service-failure-policy></code>	Optional	<p>Specifies the action to take when an abnormally behaving service thread cannot be terminated gracefully by the service guardian.</p> <p>Legal values are:</p> <ul style="list-style-type: none"> ▪ <code>exit-cluster</code> – 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. ▪ <code>exit-process</code> – 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. ▪ <code>logging</code> – 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 <code>com.tangosol.net.ServiceFailurePolicy</code> implementation. <p>Default value is <code>exit-cluster</code>.</p>
<code><member-listener></code>	Optional	<p>Specifies the configuration information for a class that implements the <code>com.tangosol.net.MemberListener</code> interface. The implementation must have a public default constructor.</p> <p>The <code>MemberListener</code> implementation receives service lifecycle events. The <code><member-listener></code> element is used as an alternative to programmatically adding a <code>MapListener</code> on a service.</p>
<code><request-timeout></code>	Optional	<p>Specifies the default timeout value in milliseconds for requests that can time-out (for example, implement the <code>com.tangosol.net.PriorityTask</code> interface), but don't explicitly specify the request timeout value. The request time is measured on the client side as the time elapsed from the moment a request is sent for execution to the corresponding server node(s) and includes the following:</p> <ol style="list-style-type: none"> (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. <p>Legal values are positive integers or zero (indicating no default timeout). Default value is the <code>request-timeout</code> value specified in the <code>tangosol-coherence.xml</code> descriptor. See the <code>request-timeout</code> parameter in "InvocationService Parameters" on page A-70.</p>

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-27](#) describes the elements you can define within the `invocation-service-proxy` element.

Table B-27 *invocation-service-proxy Subelement*

Element	Required/ Optional	Description
<code><enabled></code>	Optional	Specifies whether the invocation service proxy is enabled. If disabled, clients will not be able to execute <code>Invocable</code> objects on the proxy service JVM. Legal values are <code>true</code> or <code>false</code> . Default value is <code>true</code> .
<code><class-name></code>	Optional	Specifies the fully qualified name of a class that implements the <code>com.tangosol.net.InvocationService</code> interface. The class acts as an interceptor between a client and a proxied invocation service in order 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.
<code><init-params></code>	Optional	Contains initialization parameters for the <code>InvocationService</code> implementation.

jms-acceptor

Note: Coherence*Extend-JMS support has been deprecated.

Used in: [acceptor-config](#).

Description

The `jms-acceptor` element specifies the configuration information for a connection acceptor that accepts connections from Coherence*Extend clients over JMS. For additional details and example configurations see *Oracle Coherence Client Guide*.

Elements

[Table B-28](#) describes the elements you can define within the `jms-acceptor` element.

Table B-28 *jms-acceptor Subelements*

Element	Required/ Optional	Description
<queue-connection-factory-name>	Required	Specifies the JNDI name of the JMS <code>QueueConnectionFactory</code> used by the connection acceptor.
<queue-name>	Required	Specifies the JNDI name of the JMS Queue used by the connection acceptor.

jms-initiator

Note: Coherence*Extend-JMS support has been deprecated.

Used in: [initiator-config](#).

Description

The `jms-initiator` element specifies the configuration information for a connection initiator that enables Coherence*Extend clients to connect to a remote cluster by using JMS. For additional details and example configurations see *Oracle Coherence Client Guide*.

Elements

The following table describes the elements you can define within the `jms-initiator` element.

Table B–29 *jms-initiator* Subelements

Element	Required/ Optional	Description
<code><queue-connection-factory-name></code>	Required	Specifies the JNDI name of the JMS <code>QueueConnectionFactory</code> used by the connection initiator.
<code><queue-name></code>	Required	Specifies the JNDI name of the JMS Queue used by the connection initiator.
<code><connect-timeout></code>	Optional	<p>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:</p> <p><code>[\d] + [[.] [\d] +] ? [MS ms S s M m H h D d] ?</code></p> <p>where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) <p>If the value does not contain a unit, a unit of milliseconds is assumed. Default value is an infinite timeout.</p>

key-associator

Used in: [distributed-scheme](#)

Description

Specifies an implementation of a `com.tangosol.net.partition.KeyAssociator` which will be used to determine associations between keys, allowing related keys to reside on the same partition.

Alternatively the cache's keys may manage the association by implementing the `com.tangosol.net.cache.KeyAssociation` interface.

Elements

[Table B-30](#) describes the elements you can define within the key-associator element.

Table B-30 *key-associator Subelements*

Element	Required/ Optional	Description
<class-name>	Required	The name of a class that implements the <code>com.tangosol.net.partition.KeyAssociator</code> interface. This implementation must have a zero-parameter public constructor. Default value is the value of the <code>key-associator/class-name</code> parameter specified in the <code>tangosol.coherence.xml</code> descriptor. See "DistributedCache Service Parameters" on page A-65 for more information.

key-partitioning

Used in: [distributed-scheme](#)

Description

Specifies an implementation of a `com.tangosol.net.partition.KeyPartitioningStrategy` which will be used to determine the partition in which a key will reside.

Elements

[Table B-31](#) describes the elements you can define within the key-partitioning element.

Table B-31 *key-partitioning Subelements*

Element	Required/Optional	Description
<code><class-name></code>	Required	The name of a class that implements the <code>com.tangosol.net.partition.KeyPartitioningStrategy</code> interface. This implementation must have a zero-parameter public constructor. Default value is the value of the <code>key-partitioning/class-name</code> parameter specified in the <code>tangosol-coherence.xml</code> descriptor. See "DistributedCache Service Parameters" on page A-65 for more information.

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-32](#) describes the elements you can define within the `key-store` element.

Table B-32 *key-store Subelements*

Element	Required/ Optional	Description
<code><url></code>	Required	Specifies the Uniform Resource Locator (URL) to a key store.
<code><password></code>	Optional	Specifies the password for the key store.
<code><type></code>	Optional	Specifies the type of a <code>java.security.KeyStore</code> instance. The default value is <code>JKS</code> .

lh-file-manager

Used in: [external-scheme](#), [paged-external-scheme](#), [async-store-manager](#).

Description

Configures a store manager which will use a Coherence LH on disk embedded database for storage. See "[Persistent Cache on Disk](#)" on page 15-3 and "[In-memory Cache with Disk Based Overflow](#)" on page 15-4 for examples of LH-based store configurations.

Implementation

Implemented by the `com.tangosol.io.lh.LHBinaryStoreManager` class. The `BinaryStore` objects created by this class are instances of `javadoc:com.tangosol.io.lh.LHBinaryStore`.

Elements

[Table B-33](#) describes the elements you can define within the `lh-file-manager` element.

Table B-33 *lh-file-manager Subelements*

Element	Required/Optional	Description
<code><class-name></code>	Optional	Specifies a custom implementation of the LH <code>BinaryStoreManager</code> . Any custom implementation must extend the <code>com.tangosol.io.lh.LHBinaryStoreManager</code> class and declare the exact same set of public constructors.
<code><init-params></code>	Optional	Specifies initialization parameters, for use in custom LH file manager implementations which implement the <code>com.tangosol.run.xml.XmlConfigurable</code> interface.
<code><directory></code>	Optional	Specifies the path name for the root directory that the LH file manager will use to store files in. If not specified or specifies a non-existent directory, a temporary file in the default location will be used.
<code><file-name></code>	Optional	Specifies the name for a non-temporary (persistent) file that the LH file manager will use to store data in. Specifying this parameter will cause the <code>lh-file-manager</code> to use non-temporary database instances. Use this parameter only for local caches that are backed by a cache loader from a non-temporary file: this allows the local cache to be pre-populated from the disk file on startup. When specified it is recommended that it use the <code>{cache-name}</code> macro described in " Using Parameter Macros " on page 11-12. Normally this parameter should be left unspecified, indicating that temporary storage is to be used.

listener

Used in: [local-scheme](#), [external-scheme](#), [paged-external-scheme](#), [distributed-scheme](#), [replicated-scheme](#), [optimistic-scheme](#), [near-scheme](#), [versioned-near-scheme](#), [overflow-scheme](#), [read-write-backing-map-scheme](#), [versioned-backing-map-scheme](#).

Description

The Listener element specifies an implementation of a `com.tangosol.util.MapListener` which will be notified of events occurring on a cache.

Elements

The following table describes the elements you can define within the listener element.

Table B–34 *listener Subelement*

Element	Required/ Optional	Description
<code><class-scheme></code>	Required	Specifies the full class name of the listener implementation to use. The specified class must implement the <code>com.tangosol.util.MapListener</code> interface.

local-address

Used in: [tcp-acceptor](#), [tcp-initiator](#)

Description

The `local-address` element specifies the local address (IP or DNS name) and port to which a TCP/IP socket is bound.

The `local-address` element is used within a TCP/IP acceptor definition to specify the address and port on which the TCP/IP server socket (opened by the connection acceptor) is bound. The socket is used by the proxy service to accept connections from Coherence*Extend clients. The following example binds the server socket to 192 . 168 . 0 . 2 : 9099.

```
<local-address>
  <address>192.168.0.2</address>
  <port>9099</port>
</local-address>
```

The `local-address` element is used within a TCP/IP initiator definition to specify the local address and port on which the TCP/IP client socket (opened by the connection initiator) is bound. The socket is used by remote services to connect to a proxy service on the cluster. The following example binds the client socket to 192 . 168 . 0 . 1 on port 9099:

```
<local-address>
  <address>192.168.0.1</address>
  <port>9099</port>
</local-address>
```

Elements

[Table B-57](#) describes the subelements you can define within the `local-address` element.

Table B-35 *local-address Subelements*

Element	Required/ Optional	Description
<address>	Optional	Specifies the address (IP or DNS name) on which a TCP/IP socket listens and publishes.
<port>	Optional	Specifies the port on which a TCP/IP socket listens and publishes. Legal values are from 1 to 65535. When used in the context of a TCP/IP server (that is, for a TCP acceptor), the port child element is required.

local-scheme

Used in: [caching-schemes](#), [distributed-scheme](#), [replicated-scheme](#), [optimistic-scheme](#), [near-scheme](#), [versioned-near-scheme](#), [overflow-scheme](#), [read-write-backing-map-scheme](#), [versioned-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 ["Local Cache of a Partitioned Cache \(Near cache\)"](#) on page 15-7 for examples of various local cache configurations.

Implementation

Local caches are implemented by the `com.tangosol.net.cache.LocalCache` class.

Cache of an External Store

A local cache may be backed by an external cache store (see ["cachestore-scheme"](#) on page B-19). Cache misses will read-through to the back end store to retrieve the data. If a writable store is provided, cache writes will propagate to the cache store as well. For optimizing read/write access against a cache store, see the ["read-write-backing-map-scheme"](#) on page B-85.

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-36](#) describes the elements you can define within the `local-scheme` element.

Table B-36 *local-scheme Subelements*

Element	Required/Optional	Description
<code><scheme-name></code>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<code><scheme-ref></code>	Optional	Specifies the name of another scheme to inherit from. See "Using Scheme Inheritance" on page 11-9 for more information.
<code><service-name></code>	Optional	Specifies the name of the service which will manage caches created from this scheme. Services are configured from within the <code><services></code> element in the <code>tangosol-coherence.xml</code> descriptor. See Appendix A, "Operational Configuration Elements" for more information.

Table B–36 (Cont.) local-scheme Subelements

Element	Required/ Optional	Description
<code><class-name></code>	Optional	Specifies a custom implementation of the local cache. Any custom implementation must extend the <code>com.tangosol.net.cache.LocalCache</code> class and declare the exact same set of public constructors.
<code><init-params></code>	Optional	Specifies initialization parameters, for use in custom local cache implementations which implement the <code>com.tangosol.run.xml.XmlConfigurable</code> interface.
<code><eviction-policy></code>	Optional	<p>Specifies the type of eviction policy to use. Legal values are:</p> <ul style="list-style-type: none"> ■ LRU - Least Recently Used eviction policy chooses which entries to evict based on how recently they were last accessed, evicting those that were not accessed the for the longest period first. ■ LFU - Least Frequently Used eviction policy chooses which entries to evict based on how often they are being accessed, evicting those that are accessed least frequently first. ■ HYBRID - Hybrid eviction policy chooses which entries to evict based 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. ■ <code><class-scheme></code> - A custom eviction policy, specified as a class-scheme. The class specified within this scheme must implement the <code>com.tangosol.net.cache.LocalCache.EvictionPolicy</code> interface. <p>Default value is HYBRID.</p>
<code><high-units></code>	Optional	Used to limit the size of the cache. Contains the maximum number of units that can be placed in the cache before pruning occurs. An entry is the unit of measurement, unless it is overridden by an alternate unit-calculator (see <code><unit-calculator></code> subelement). When this limit is exceeded, the cache will begin the pruning process, evicting entries according to the eviction policy. Legal values are positive integers or zero. Zero implies no limit. Default value is 0.
<code><low-units></code>	Optional	Contains the lowest number of units that a cache will be pruned down to when pruning takes place. A pruning will not necessarily result in a cache containing this number of units, however a pruning will never result 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 <code><unit-calculator></code> subelement). When pruning occurs entries will continue to be evicted according to the eviction policy until this size. Legal values are positive integers or zero. Zero implies the default. Default value is 75% of the high-units setting (that is, for a high-units setting of 1000 the default low-units will be 750).
<code><unit-calculator></code>	Optional	<p>Specifies the type of unit calculator to use. A unit calculator is used to determine the cost (in "units") of a given object. Legal values are:</p> <ul style="list-style-type: none"> ■ <code>FIXED</code>—A unit calculator that assigns an equal weight of 1 to all cached objects. ■ <code>BINARY</code>—A unit calculator that assigns an object a weight equal to the number of bytes of memory required to cache the object. This requires that the objects are <code>com.tangosol.util.Binary</code> instances, as in a Partitioned cache. See <code>com.tangosol.net.cache.BinaryMemoryCalculator</code> for additional details. ■ <code><class-scheme></code>— A custom unit calculator, specified as a class-scheme. The class specified within this scheme must implement the <code>com/tangosol/net/cache/ConfigurableCacheMap.UnitCalculator</code> interface. <p>This element is used only if the <code>high-units</code> element is set to a positive number. Default value is <code>FIXED</code>.</p>

Table B–36 (Cont.) local-scheme Subelements

Element	Required/ Optional	Description
<unit-factor>	Optional	<p>The unit-factor element specifies the factor by which the units, low-units and high-units properties are adjusted. Using a BINARY unit calculator, for example, the factor of 1048576 could be used to count megabytes instead of bytes.</p> <p>Using a BINARY unit calculator, for example, the factor of 1048576 could be used to count megabytes instead of bytes.</p> <p>Note: This element was introduced only to avoid changing the type of the units, low units and high units properties from 32-bit values to 64-bit values and is used only if the high-units element is set to a positive number. It is expected that this element will be dropped in a future release.</p> <p>Valid values are positive integer numbers. Default value is 1.</p>
<expiry-delay>	Optional	<p>Specifies the amount of time since the last update that entries are kept by the cache before being expired. Entries that have expired are not be accessible and are evicted the next time a client accesses the cache. Any attempt to read an expired entry will result in a reloading of the entry from the configured cache store (see <cachestore-scheme>).</p> <p>The value of this element must be in the following format:</p> <pre>[\d] + [[.] [\d] +] ? [MS ms S s M m H h D d] ?</pre> <p>where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) <p>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.</p> <p>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.</p>
<cachestore-scheme>	Optional	Specifies the store which is being cached. If unspecified the cached data will only reside in memory, and only reflect operations performed on the cache itself.
<pre-load>	Optional	Specifies whether or not a cache will pre-load data from its CacheLoader (or CacheStore) object. Valid values are true and false. Default value is false.
<listener>	Optional	Specifies an implementation of a com.tangosol.util.MapListener which will be notified of events occurring on the cache.

near-scheme

Used in: [caching-schemes](#).

Description

The `near-scheme` defines a two-tier cache consisting of a front-tier (see `<front-scheme>` subelement) which caches a subset of a back-tier cache (see `<back-scheme>` subelement). The front-tier is generally a fast, size limited cache, while the back-tier is slower, but much higher capacity. A typical deployment might use a [local-scheme](#) for the front-tier, and a [distributed-scheme](#) for the back-tier. The result is that a portion of a large partitioned cache will be cached locally in-memory allowing for very fast read access. See "Near Cache" on page 10-7 for a more detailed description of near caches, and "Local Cache of a Partitioned Cache (Near cache)" on page 15-7 for an example of near cache configurations.

Implementation

The near scheme is implemented by the `com.tangosol.net.cache.NearCache` class.

Front-tier Invalidation

Specifying an invalidation-strategy (see `<invalidation-strategy>` subelement) defines a strategy that is used to keep the front tier of the near cache in sync with the back tier. Depending on that strategy a near cache is configured to listen to certain events occurring on the back tier and automatically update (or invalidate) the front portion of the near cache.

Elements

[Table B-37](#) describes the elements you can define within the `near-scheme` element.

Table B-37 *near-scheme Subelements*

Element	Required/ Optional	Description
<code><scheme-name></code>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<code><scheme-ref></code>	Optional	Specifies the name of another scheme to inherit from. See "Using Scheme Inheritance" on page 11-9 for more information
<code><class-name></code>	Optional	Specifies a custom implementation of the near cache. Any custom implementation must extend the <code>com.tangosol.net.cache.NearCache</code> class and declare the exact same set of public constructors.
<code><init-params></code>	Optional	Specifies initialization parameters for custom near cache implementations which implement the <code>com.tangosol.run.xml.XmlConfigurable</code> interface.
<code><listener></code>	Optional	Specifies an implementation of a <code>com.tangosol.util.MapListener</code> which will be notified of events occurring on the cache.

Table B–37 (Cont.) near-scheme Subelements

Element	Required/ Optional	Description
<code><front-scheme></code>	Required	<p>Specifies the cache-scheme to use in creating the front-tier cache. Legal values are:</p> <ul style="list-style-type: none">▪ <code>local-scheme</code>▪ <code>external-scheme</code>▪ <code>paged-external-scheme</code>▪ <code>class-scheme</code> <p>The eviction policy of the front-scheme defines which entries will be cached locally. For example:</p> <pre><front-scheme> <local-scheme> <eviction-policy>HYBRID</eviction-policy> <high-units>1000</high-units> </local-scheme> </front-scheme></pre>

Table B–37 (Cont.) near-scheme Subelements

Element	Required/ Optional	Description
<back-scheme>	Required	<p>Specifies the cache-scheme to use in creating the back-tier cache. Legal values are:</p> <ul style="list-style-type: none"> distributed-scheme replicated-scheme optimistic-scheme local-scheme external-scheme paged-external-scheme class-scheme remote-cache-scheme <p>For example:</p> <pre><back-scheme> <distributed-scheme> <scheme-ref>default-distributed</scheme-ref> </distributed-scheme> </back-scheme></pre>
<invalidation-strategy>	Optional	<p>Specifies the strategy used keep the front-tier in-sync with the back-tier. Please see <code>com.tangosol.net.cache.NearCache</code> for more details. Legal values are:</p> <ul style="list-style-type: none"> none - instructs the cache not to listen for invalidation events at all. This is the best choice for raw performance and scalability when business requirements permit the use of data which might not be absolutely current. Freshness of data can be guaranteed by use of a sufficiently brief eviction policy. The worst case performance is identical to a standard Distributed cache. present - instructs the near cache to listen to the back map events related only to the items currently present in the front map. This strategy works best when cluster nodes have sticky data access patterns (for example, HTTP session management with a sticky load balancer). all - instructs the near cache to listen to all back map events. This strategy is optimal for read-heavy access patterns where there is significant overlap between the front caches on each cluster member. auto - instructs the near cache to switch between present and all strategies automatically based on the cache statistics. <p>Default value is auto.</p>
<autostart>	Optional	<p>The autostart element is intended to be used by cache servers (that is, <code>com.tangosol.net.DefaultCacheServer</code>). It specifies whether the cache services associated with this cache scheme should be automatically started at a cluster node. Legal values are true or false. Default value is false.</p>

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–38](#) describes the elements you can define within the `nio-file-manager` element.

Table B–38 *nio-file-manager Subelements*

Element	Required/ Optional	Description
<code><class-name></code>	Optional	Specifies a custom implementation of the local cache. Any custom implementation must extend the <code>com.tangosol.io.nio.MappedStoreManager</code> class and declare the exact same set of public constructors.
<code><init-params></code>	Optional	Specifies initialization parameters, for use in custom <code>nio-file-manager</code> implementations which implement the <code>com.tangosol.run.xml.XmlConfigurable</code> interface.
<code><initial-size></code>	Optional	<p>Specifies the initial buffer size in megabytes. The value of this element must be in the following format:</p> <pre>[\d] + [[.] [\d] +] ? [K k M m G g] ? [B b] ?</pre> <p>where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied:</p> <ul style="list-style-type: none"> ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) <p>If the value does not contain a factor, a factor of mega is assumed. Legal values are positive integers between 1 and <code>Integer.MAX_VALUE - 1023</code> (that is, 2,147,482,624 bytes). Default value is 1MB.</p>
<code><maximum-size></code>	Optional	<p>Specifies the maximum buffer size in bytes. The value of this element must be in the following format:</p> <pre>[\d] + [[.] [\d] +] ? [K k M m G g] ? [B b] ?</pre> <p>where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied:</p> <ul style="list-style-type: none"> ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) <p>If the value does not contain a factor, a factor of mega is assumed. Legal values are positive integers between 1 and <code>Integer.MAX_VALUE - 1023</code> (that is, 2,147,482,624 bytes). Default value is 1024MB.</p>
<code><directory></code>	Optional	Specifies the path name for the root directory that the manager will use to store files in. If not specified or specifies a non-existent directory, a temporary file in the default location will be used.

nio-memory-manager

Used in: [external-scheme](#), [paged-external-scheme](#), [async-store-manager](#).

Description

Configures a store-manager which uses an off JVM heap, memory region for storage, which means that it does not affect the Java heap size and the related JVM garbage-collection performance that can be responsible for application pauses. See "[NIO In-memory Cache](#)" on page 15-2 for an example of an NIO cache configuration.

Note: JVMs require the use of a command line parameter if the total NIO buffers will be greater than 64MB. For example: -
XX:MaxDirectMemorySize=512M

Implementation

Implemented by the `com.tangosol.io.nio.DirectStoreManager` class. The `BinaryStore` objects created by this class are instances of the `com.tangosol.io.nio.BinaryMapStore`.

Elements

[Table B-39](#) describes the elements you can define within the `nio-memory-manager` element.

Table B-39 *nio-memory-manager Subelements*

Element	Required/ Optional	Description
<class-name>	Optional	Specifies a custom implementation of the local cache. Any custom implementation must extend the <code>com.tangosol.io.nio.DirectStoreManager</code> class and declare the exact same set of public constructors.

Table B–39 (Cont.) *nio-memory-manager* Subelements

Element	Required/ Optional	Description
<code><init-params></code>	Optional	Specifies initialization parameters, for use in custom <code>nio-memory-manager</code> implementations which implement the <code>com.tangosol.run.xml.XmlConfigurable</code> interface.
<code><initial-size></code>	Optional	<p>Specifies the initial buffer size in bytes. The value of this element must be in the following format:</p> <pre>[\d] + [[.] [\d] +] ? [K k M m G g] ? [B b] ?</pre> <p>where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied:</p> <ul style="list-style-type: none"> ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) <p>If the value does not contain a factor, a factor of mega is assumed. Legal values are positive integers between 1 and <code>Integer.MAX_VALUE - 1023</code> (that is, 2,147,482,624 bytes). Default value is 1MB.</p>
<code><maximum-size></code>	Optional	<p>Specifies the maximum buffer size in bytes. The value of this element must be in the following format:</p> <pre>[\d] + [[.] [\d] +] ? [K k M m G g] ? [B b] ?</pre> <p>where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied:</p> <ul style="list-style-type: none"> ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) <p>If the value does not contain a factor, a factor of mega is assumed. Legal values are positive integers between 1 and <code>Integer.MAX_VALUE - 1023</code> (that is, 2,147,482,624 bytes). Default value is 1024MB.</p>

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.

See `com.tangosol.net.cache.AbstractBundler` for additional implementation details.

Elements

[Table B–40](#) describes the subelement for the `operation-bundling` element.

Table B–40 *operation-bundling Subelement*

Element	Required/ Optional	Description
<code><bundle-config></code>	Required	Describes one or more bundle-able operations.

optimistic-scheme

Used in: [caching-schemes](#), [near-scheme](#), [versioned-near-scheme](#), [overflow-scheme](#)

The optimistic scheme defines a cache which fully replicates all of its data to all cluster nodes that run the service (see `<service-name>` subelement). See "Optimistic Cache" on page 10-7 for a more detailed description of optimistic caches.

Optimistic Locking

Unlike the [replicated-scheme](#) and [distributed-scheme](#) caches, optimistic caches do not support concurrency control (locking). Individual operations against entries are atomic but there is no guarantee that the value stored in the cache does not change between atomic operations. The lack of concurrency control allows optimistic caches to support very fast write operations.

Cache Storage (Backing Map)

Storage for the cache is specified by using the `backing-map-scheme` (see `<backing-map-scheme>` subelement). For instance an optimistic cache which uses a [local-scheme](#) for its backing map will result in cache entries being stored in-memory.

Elements

[Table B-41](#) describes the elements you can define within the `optimistic-scheme` element.

Table B-41 *optimistic-scheme Subelements*

Element	Required/Optional	Description
<code><scheme-name></code>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<code><scheme-ref></code>	Optional	Specifies the name of another scheme to inherit from. See "Using Scheme Inheritance" on page 11-9 for more information.
<code><service-name></code>	Optional	Specifies the name of the service which will manage caches created from this scheme. Services are configured from within the <code><services></code> parameter in <code>tangosol-coherence.xml</code> . See Appendix A , "Operational Configuration Elements" for more information.
<code><serializer></code>	Optional	Specifies either: the class configuration information for a <code>com.tangosol.io.Serializer</code> implementation used to serialize and deserialize user types, or it references a serializer class configuration that is defined in the operational configuration file (see "serializers" on page A-61).
<code><listener></code>	Optional	Specifies an implementation of a <code>com.tangosol.util.MapListener</code> which will be notified of events occurring on the cache.

Table B–41 (Cont.) optimistic-scheme Subelements

Element	Required/ Optional	Description
<backing-map-scheme>	Optional	<p>Specifies what type of cache will be used within the cache server to store the entries. Legal values are:</p> <ul style="list-style-type: none"> ■ local-scheme ■ external-scheme ■ paged-external-scheme ■ overflow-scheme ■ class-scheme <p>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.</p>
<guardian-timeout>	Optional	<p>Specifies the guardian timeout value to use for guarding the service and any dependant 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>.</p> <p>The value of this element must be in the following format:</p> <pre>[\\d]+[\\.][\\d]+]?[MS ms S s M m H h D d]?</pre> <p>where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) <p>If the value does not contain a unit, a unit of milliseconds is assumed.</p>

Table B–41 (Cont.) optimistic-scheme Subelements

Element	Required/ Optional	Description
<code><service-failure-policy></code>	Optional	<p>Specifies the action to take when an abnormally behaving service thread cannot be terminated gracefully by the service guardian.</p> <p>Legal values are:</p> <ul style="list-style-type: none"> ▪ <code>exit-cluster</code> – 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. ▪ <code>exit-process</code> – 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. ▪ <code>logging</code> – 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 <code>com.tangosol.net.ServiceFailurePolicy</code> implementation. <p>Default value is <code>exit-cluster</code>.</p>
<code><member-listener></code>	Optional	<p>Specifies the configuration information for a class that implements the <code>com.tangosol.net.MemberListener</code> interface. The implementation must have a public default constructor.</p> <p>The <code>MemberListener</code> implementation receives cache service lifecycle events. The <code><member-listener></code> element is used as an alternative to programmatically adding a <code>MapListener</code> on a service.</p>
<code><autostart></code>	Optional	<p>The <code>autostart</code> element is intended to be used by cache servers (that is, <code>com.tangosol.net.DefaultCacheServer</code>). It specifies whether the cache services associated with this cache scheme should be automatically started at a cluster node. Legal values are <code>true</code> or <code>false</code>. Default value is <code>false</code>.</p>

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-42](#) describes the elements you can define within the `outgoing-message-handler` element.

Table B–42 *outgoing-message-handler Subelements*

Element	Required/ Optional	Description
<heartbeat-interval>	Optional	<p>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:</p> <pre>[\d] + [[.] [\d] +] ? [MS ms S s M m H h D d] ?</pre> <p>where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) <p>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.</p>
<heartbeat-timeout>	Optional	<p>Specifies the maximum amount of time to wait for a response to a ping request before declaring the underlying connection unusable. The value of this element must be in the following format:</p> <pre>[\d] + [[.] [\d] +] ? [MS ms S s M m H h D d] ?</pre> <p>where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) <p>If the value does not contain a unit, a unit of milliseconds is assumed. The default value is the value of the <code>request-timeout</code> element.</p>
<request-timeout>	Optional	<p>Specifies the maximum amount of time to wait for a response message before declaring the underlying connection unusable. The value of this element must be in the following format:</p> <pre>[\d] + [[.] [\d] +] ? [MS ms S s M m H h D d] ?</pre> <p>where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) <p>If the value does not contain a unit, a unit of milliseconds is assumed. The default value is an infinite timeout.</p>

overflow-scheme

Used in: [caching-schemes](#), [distributed-scheme](#), [replicated-scheme](#), [optimistic-scheme](#), [read-write-backing-map-scheme](#), [versioned-backing-map-scheme](#).

Description

The `overflow-scheme` defines a two-tier cache consisting of a fast, size limited front-tier, and slower but much higher capacity back-tier cache. When the size limited front fills up, evicted entries are transparently moved to the back. In the event of a cache miss, entries may move from the back to the front. A typical deployment might use a [local-scheme](#) for the front-tier, and a [external-scheme](#) for the back-tier, allowing for fast local caches with capacities larger the JVM heap would allow. In such a deployment the `local-scheme` element's `high-units` and `eviction-policy` will control the transfer (eviction) of entries from the front to back caches.

Note: Relying on overflow for normal cache storage is not recommended. It should only be used to help avoid eviction-related data loss in the case where the storage requirements temporarily exceed the configured capacity. In general, the overflow's on disk storage should remain empty.

Implementation

Implemented by either `com.tangosol.net.cache.OverflowMap` or `com.tangosol.net.cache.SimpleOverflowMap`, see `expiry-enabled` for details.

Entry Expiration

Overflow supports automatic expiration of entries based on the age of the value, as configured by the `<expiry-delay>` subelement.

Elements

[Table B-43](#) describes the elements you can define within the `overflow-scheme` element.

Table B-43 *overflow-scheme Subelements*

Element	Required/Optional	Description
<code><scheme-name></code>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<code><scheme-ref></code>	Optional	Specifies the name of another scheme to inherit from. See "Using Scheme Inheritance" on page 11-9 for more information.
<code><class-name></code>	Optional	Specifies a custom implementation of the overflow cache. Any custom implementation must extend either the <code>com.tangosol.net.cache.OverflowMap</code> or <code>com.tangosol.net.cache.SimpleOverflowMap</code> class, and declare the exact same set of public constructors.
<code><init-params></code>	Optional	Specifies initialization parameters, for use in custom overflow cache implementations which implement the <code>com.tangosol.run.xml.XmlConfigurable</code> interface.

Table B–43 (Cont.) overflow-scheme Subelements

Element	Required/ Optional	Description
<front-scheme>	Required	<p>Specifies the cache-scheme to use in creating the front-tier cache. Legal values are:</p> <ul style="list-style-type: none"> ■ <code>local-scheme</code> ■ <code>external-scheme</code> ■ <code>paged-external-scheme</code> ■ <code>class-scheme</code> <p>The eviction policy of the front-scheme defines which entries which items are in the front versus back tiers. For example:</p> <pre><front-scheme> <local-scheme> <eviction-policy>HYBRID</eviction-policy> <high-units>1000</high-units> </local-scheme> </front-scheme></pre>
<back-scheme>	Required	<p>Specifies the cache-scheme to use in creating the back-tier cache. Legal values are:</p> <ul style="list-style-type: none"> ■ <code>local-scheme</code> ■ <code>external-scheme</code> ■ <code>paged-external-scheme</code> ■ <code>class-scheme</code> <p>For Example:</p> <pre><back-scheme> <external-scheme> <lh-file-manager/> </external-scheme> </back-scheme></pre>
<miss-cache-scheme>	Optional	<p>Specifies a cache-scheme for maintaining information on cache misses. For caches which are not expiry-enabled (see <expiry-enabled> subelement), the miss-cache is used track keys which resulted in both a front and back tier cache miss. The knowledge that a key is not in either tier allows some operations to perform faster, as they can avoid querying the potentially slow back-tier. A size limited scheme may be used to control how many misses are tracked. If unspecified no cache-miss data will be maintained. Legal values are:</p> <ul style="list-style-type: none"> ■ <code>local-scheme</code>

Table B-43 (Cont.) overflow-scheme Subelements

Element	Required/ Optional	Description
<expiry-enabled>	Optional	Turns on support for automatically-expiring data, as provided by the <code>com.tangosol.net.cache.CacheMap</code> API. When enabled the overflow-scheme will be implemented using <code>com.tangosol.net.cache.OverflowMap</code> , rather than <code>com.tangosol.net.cache.SimpleOverflowMap</code> . Legal values are <code>true</code> or <code>false</code> . Default value is <code>false</code> .
<expiry-delay>	Optional	<p>Specifies the amount of time since the last update that entries are kept by the cache before being expired. Entries that have expired are not be accessible and are evicted the next time a client accesses the cache.</p> <p>The value of this element must be in the following format:</p> <pre>[\d] + [[.] [\d] +] ? [MS ms S s M m H h D d] ?</pre> <p>where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) <p>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.</p> <p>Note: The expiry delay parameter (<code>cExpiryMillis</code>) is defined as an integer and is expressed in milliseconds. Therefore, the maximum amount of time can never exceed <code>Integer.MAX_VALUE</code> (2147483647) milliseconds or approximately 24 days.</p>
<listener>	Optional	Specifies an implementation of a <code>com.tangosol.util.MapListener</code> which will be notified of events occurring on the cache.

paged-external-scheme

Used in: [caching-schemes](#), [distributed-scheme](#), [replicated-scheme](#), [optimistic-scheme](#), [near-scheme](#), [versioned-near-scheme](#), [overflow-scheme](#), [read-write-backing-map-scheme](#), [versioned-backing-map-scheme](#)

Description

As with [external-scheme](#), `paged-external-schemes` define caches which are not JVM heap based, allowing for greater storage capacity. The `paged-external-scheme` optimizes LRU eviction by using a paging approach (see `<paging>` subelement). See [Chapter 14, "Serialization Paged Cache,"](#) for a detailed description of the paged cache functionality.

Implementation

This scheme is implemented by the `com.tangosol.net.cache.SerializationPagedCache` class.

Paging

Cache entries are maintained over a series of pages, where each page is a separate `com.tangosol.io.BinaryStore`, obtained from the configured storage manager (see ["Pluggable Storage Manager"](#)). When a page is created it is considered to be the "current" page, and all write operations are performed against this page. On a configured interval (see `<page-duration>` subelement), the current page is closed and a new current page is created. Read operations for a given key are performed against the last page in which the key was stored. When the number of pages exceeds a configured maximum (see `<page-limit>` subelement), the oldest page is destroyed and those items which were not updated since the page was closed are be evicted. For example configuring a cache with a duration of ten minutes per page, and a maximum of six pages, will result in entries being cached for at most an hour. Paging improves performance by avoiding individual delete operations against the storage manager as cache entries are removed or evicted. Instead the cache simply releases its references to those entries, and relies on the eventual destruction of an entire page to free the associated storage of all page entries in a single stroke.

Pluggable Storage Manager

External schemes use a pluggable store manager to create and destroy pages, and to access entries within those pages. Supported store-managers include:

- [async-store-manager](#)—a wrapper providing asynchronous write capabilities for of other store-manager implementations
- [custom-store-manager](#)—allows definition of custom implementations of store-managers
- [bdb-store-manager](#)—uses Berkeley Database JE to implement an on disk cache
- [lh-file-manager](#)—uses a Coherence LH on disk database cache
- [nio-file-manager](#)—uses NIO to implement memory-mapped file based cache
- [nio-memory-manager](#)—uses NIO to implement an off JVM heap, in-memory cache

Persistence (long-term storage)

Paged external caches are used for temporary storage of large data sets, for example as the back-tier of an [overflow-scheme](#). These caches are not usable as for long-term storage (persistence), and will not survive beyond the life of the JVM. Clustered persistence should be configured by using a [read-write-backing-map-scheme](#) on a [distributed-scheme](#). If a non-clustered persistent cache is what is needed, refer to "Persistence (long-term storage)" on page B-35.

Elements

[Table B-44](#) describes the elements you can define within the `paged-external-scheme` element.

Table B-44 *paged-external-scheme Subelements*

Element	Required/Optional	Description
<code><scheme-name></code>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<code><scheme-ref></code>	Optional	Specifies the name of another scheme to inherit from. See "Using Scheme Inheritance" on page 11-9 for more information.
<code><class-name></code>	Optional	Specifies a custom implementation of the external paged cache. Any custom implementation must extend the <code>com.tangosol.net.cache.SerializationPagedCache</code> class and declare the exact same set of public constructors.
<code><init-params></code>	Optional	Specifies initialization parameters, for use in custom external paged cache implementations which implement the <code>com.tangosol.run.xml.XmlConfigurable</code> interface.
<code><listener></code>	Optional	Specifies an implementation of a <code>com.tangosol.util.MapListener</code> which will be notified of events occurring on the cache.
<code><page-limit></code>	Required	Specifies the maximum number of active pages for the paged cache. Legal values are positive integers between 2 and 3600.
<code><page-duration></code>	Optional	<p>Specifies the length of time, in seconds, that a page in the paged cache is current. The value of this element must be in the following format:</p> <pre>[\d] + [[.] [\d] +] ? [MS ms S s M m H h D d] ?</pre> <p>where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) <p>If the value does not contain a unit, a unit of seconds is assumed. Legal values are between 5 and 604800 seconds (one week) and zero (no expiry). Default value is zero</p>
<code><async-store-manager></code>	Optional	Configures the paged external cache to use an asynchronous storage manager wrapper for any other storage manager. See "Pluggable Storage Manager" on page B-34 for more information.
<code><custom-store-manager></code>	Optional	Configures the paged external cache to use a custom storage manager implementation.

Table B–44 (Cont.) paged-external-scheme Subelements

Element	Required/ Optional	Description
<code><bdb-store-manager></code>	Optional	Configures the paged external cache to use Berkeley Database JE on disk databases for cache storage.
<code><lh-file-manager></code>	Optional	Configures the paged external cache to use a Coherence LH on disk database for cache storage.
<code><nio-file-manager></code>	Optional	Configures the paged external cache to use a memory-mapped file for cache storage.
<code><nio-memory-manager></code>	Optional	Configures the paged external cache to use an off JVM heap, memory region for cache storage.

partition-listener

Used in: [distributed-scheme](#)

Description

Specifies an implementation of a `com.tangosol.net.partition.PartitionListener` interface, which allows receiving partition distribution events.

Elements

[Table B–45](#) describes the elements you can define within the `partition-listener` element.

Table B–45 *partition-listener Subelements*

Element	Required/ Optional	Description
<code><class-name></code>	Required	The name of a class that implements the <code>PartitionListener</code> interface. This implementation must have a zero-parameter public constructor. Default value is the value specified in the <code>partition-listener/class-name</code> parameter in the <code>tangosol-coherence.xml</code> descriptor. See " DistributedCache Service Parameters " on page A-65 for more information.

partitioned

Used in: [backing-map-scheme](#) (within a [distributed-scheme](#) only)

Description

The `partitioned` element specifies whether the enclosed backing map is a `PartitionAwareBackingMap`. (This element is respected only for `backing-map-scheme` that is a child of a `distributed-scheme`.) If set to `true`, the specific scheme contained in the `backing-map-scheme` element will be used to configure backing maps for each individual partition of the `PartitionAwareBackingMap`; otherwise it is used for the entire backing map itself.

The concrete implementations of the `PartitionAwareBackingMap` interface are:

- `com.tangosol.net.partition.ObservableSplittingBackingCache`
- `com.tangosol.net.partition.PartitionSplittingBackingCache`
- `com.tangosol.net.partition.ReadWriteSplittingBackingMap`

Valid values are `true` or `false`. Default value is `false`.

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-46](#) describes the elements you can define within the `partitioned-quorum-policy-scheme` element.

Table B-46 *partitioned-quorum-policy-scheme Subelements*

Element	Required/ Optional	Description
<code><scheme-name></code>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<code><scheme-ref></code>	Optional	Specifies the name of another scheme to inherit from. See "Using Scheme Inheritance" on page 11-9 for more information.
<code><distribution-quorum></code>	Optional	Specifies the minimum number of ownership-enabled members of a partitioned service that must be present in order to perform partition distribution. The value must be a non-negative integer.
<code><restore-quorum></code>	Optional	Specifies the minimum number of ownership-enabled members of a partitioned service that must be present in order to restore lost primary partitions from backup. The value must be a non-negative integer.
<code><read-quorum></code>	Optional	Specifies the minimum number of storage members of a cache service that must be present in order to process "read" requests. A "read" request is any request that does not mutate the state or contents of a cache. The value must be a non-negative integer.
<code><write-quorum></code>	Optional	Specifies the minimum number of storage members of a cache service that must be present in order to process "write" requests. A "write" request is any request that may mutate the state or contents of a cache. The value must be a non-negative integer.
<code><class-name></code>	Optional	Specifies a class that provides custom quorum policies. This element cannot be used together with the default quorum elements or the <code><class-factory-name></code> element. The class must implement the <code>com.tangosol.net.ActionPolicy</code> interface. Initialization parameters can be specified using the <code><init-params></code> element.
<code><class-factory-name></code>	Optional	Specifies a factory class for creating custom action policy instances. This element cannot be used together with the default quorum elements or the <code><class-name></code> element. This element is used together with the <code><method-name></code> element. The action policy instances must implement the <code>com.tangosol.net.ActionPolicy</code> interface. In addition, initialization parameters can be specified using the <code><init-params></code> element.

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-47](#) describes the subelements you can define within the provider element.

Table B-47 *provider Subelements*

Element	Required/ Optional	Description
<code><name></code>	Optional	Specifies the name of a security provider that extends the <code>java.security.Provider</code> class. The class name can be entered using either this element or by using the <code><class-name></code> element or by using the <code><class-factory-name></code> element.
<code><class-name></code>	Optional	Specifies the name of a security provider that extends the <code>java.security.Provider</code> class. This element cannot be used together with the <code><name></code> element or the <code><class-factory-name></code> element.
<code><class-factory-name></code>	Optional	Specifies a factory class for creating <code>Provider</code> instances. The instances must implement the <code>java.security.Provider</code> class. This element cannot be used together with the <code><name></code> element or the <code><class-name></code> element. This element can be used together with the <code><method-name></code> element.
<code><method-name></code>	Optional	Specifies the name of a static factory method on the factory class which will perform object instantiation.
<code><init-params></code>	Optional	Contains class initialization parameters for the provider implementation. This element cannot be used together with the <code><name></code> 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–48](#) describes the elements you can define within the `proxy-config` element.

Table B–48 *proxy-config Subelements*

Element	Required/ Optional	Description
<code><cache-service-proxy></code>	Optional	Specifies the configuration information for a cache service proxy managed by the proxy service.
<code><invocation-service-proxy></code>	Optional	Specifies the configuration information for an invocation service proxy managed by the proxy service.

proxy-scheme

Used in: [caching-schemes](#).

Description

The `proxy-scheme` element contains the configuration information for a clustered service that allows Coherence*Extend clients to connect to the cluster and use clustered services without having to join the cluster.

Elements

[Table B-49](#) describes the subelements you can define within the `proxy-scheme` element.

Table B-49 *proxy-scheme Subelements*

Element	Required/ Optional	Description
<code><scheme-name></code>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<code><scheme-ref></code>	Optional	Specifies the name of another scheme to inherit from. See "Using Scheme Inheritance" on page 11-9 for more information.
<code><service-name></code>	Optional	Specifies the name of the service.
<code><task-hung-threshold></code>	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 <code>thread-count</code> value is positive). Legal values are positive integers or zero (indicating no default timeout). Default value is the value specified in the <code>tangosol-coherence.xml</code> descriptor. See the <code>task-hung-threshold</code> parameter in "ProxyService Parameters" on page A-71 for more information.
<code><task-timeout></code>	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 <code>thread-count</code> value is positive). If zero is specified, the default service-guardian <code><timeout-milliseconds></code> value is used. Legal values are non-negative integers. Default value is the value specified in the <code>tangosol-coherence.xml</code> descriptor. See the <code>task-timeout</code> parameter in "ProxyService Parameters" on page A-71.
<code><request-timeout></code>	Optional	<p>Specifies the maximum amount of time a client will wait for a response before abandoning the original request. The request time is measured on the client side as the time elapsed from the moment a request is sent for execution to the corresponding server node(s) and includes the following:</p> <ul style="list-style-type: none"> 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 <p>Legal values are positive integers or zero (indicating no default timeout). Default value is the value specified in the <code>tangosol-coherence.xml</code> descriptor. See the <code>request-timeout</code> parameter in "DistributedCache Service Parameters" on page A-65 for more information.</p>

Table B–49 (Cont.) proxy-scheme Subelements

Element	Required/ Optional	Description
<thread-count>	Optional	Specifies the number of daemon threads used by the service. If zero, all relevant tasks are performed on the service thread. Legal values are positive integers or zero. Default value is the value specified in the thread-count parameter of the tangosol-coherence.xml descriptor. See "ProxyService Parameters" on page A-71 for more information.
<acceptor-config>	Required	Contains the configuration of the connection acceptor used by the service to accept connections from Coherence*Extend clients and to allow them to use the services offered by the cluster without having to join the cluster.
<proxy-config>	Optional	Contains the configuration of the clustered service proxies managed by this service.
<autostart>	Optional	The autostart element is intended to be used by cache servers (that is, com.tangosol.net.DefaultCacheServer). It specifies whether this service should be automatically started at a cluster node. Legal values are true or false. Default value is false.
<guardian-timeout>	Optional	<p>Specifies the guardian timeout value to use for guarding the service and any dependant 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>.</p> <p>The value of this element must be in the following format:</p> <pre>[\d] + [[.] [\d] +] ? [MS ms S s M m H h D d] ?</pre> <p>where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) <p>If the value does not contain a unit, a unit of milliseconds is assumed.</p>

Table B–49 (Cont.) proxy-scheme Subelements

Element	Required/ Optional	Description
<code><service-failure-policy></code>	Optional	<p>Specifies the action to take when an abnormally behaving service thread cannot be terminated gracefully by the service guardian.</p> <p>Legal values are:</p> <ul style="list-style-type: none"> ▪ <code>exit-cluster</code> – 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. ▪ <code>exit-process</code> – 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. ▪ <code>logging</code> – 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 <code>com.tangosol.net.ServiceFailurePolicy</code> implementation. <p>Default value is <code>exit-cluster</code>.</p>
<code><member-listener></code>	Optional	<p>Specifies the configuration information for a class that implements the <code>com.tangosol.net.MemberListener</code> interface. The implementation must have a public default constructor.</p> <p>The <code>MemberListener</code> implementation receives service lifecycle events. The <code><member-listener></code> element is used as an alternative to programmatically adding a <code>MapListener</code> on a service.</p>
<code><proxy-quorum-policy-scheme></code>	Optional	Specifies quorum policy settings for 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-49](#) describes the subelements you can define within the `proxy-quorum-policy-scheme` element.

Table B-50 *proxy-quorum-policy-scheme Subelements*

Element	Required/ Optional	Description
<code><scheme-name></code>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<code><scheme-ref></code>	Optional	Specifies the name of another scheme to inherit from. See "Using Scheme Inheritance" on page 11-9 for more information.
<code><connect-quorum></code>	Optional	specifies the minimum number of members of a proxy service that must be present in order to allow client connections. The value must be a non-negative integer.
<code><class-name></code>	Optional	Specifies a class that provides custom quorum policies. This element cannot be used together with the <code><connect-quorum></code> element or the <code><class-factory-name></code> element. The class must implement the <code>com.tangosol.net.ActionPolicy</code> interface. Initialization parameters can be specified using the <code><init-params></code> element.
<code><class-factory-name></code>	Optional	Specifies a factory class for creating custom action policy instances. This element cannot be used together with the <code><connect-quorum></code> element or the <code><class-name></code> element. This element is used together with the <code><method-name></code> element. The action policy instances must implement the <code>com.tangosol.net.ActionPolicy</code> interface. In addition, initialization parameters can be specified using the <code><init-params></code> element.

read-write-backing-map-scheme

Used in: [caching-schemes](#), [distributed-scheme](#).

Description

The read-write-backing-map-scheme defines a backing map which provides a size limited cache of a persistent store. See [Chapter 13, "Read-Through, Write-Through, Write-Behind, and Refresh-Ahead Caching"](#) for more details.

Implementation

The read-write-backing-map-scheme is implemented by the `com.tangosol.net.cache.ReadWriteBackingMap` class.

Cache of an External Store

A read write backing map maintains a cache backed by an external persistent cache store (see `<cachestore-scheme>` subelement), cache misses will read-through to the back-end store to retrieve the data. If a writable store is provided, cache writes will propagate to the cache store as well.

Refresh-Ahead Caching

When enabled (see `<refreshahead-factor>` subelement) the cache will watch for recently accessed entries which are about to expire, and asynchronously reload them from the cache store. This insulates the application from potentially slow reads against the cache store, as items periodically expire.

Write-Behind Caching

When enabled (see `<write-delay>` subelement) the cache will delay writes to the back-end cache store. This allows for the writes to be batched (see `<write-batch-factor>` subelement) into more efficient update blocks, which occur asynchronously from the client thread.

Elements

[Table B-51](#) describes the elements you can define within the `read-write-backing-map-scheme` element.

Table B-51 *read-write-backing-map-scheme Subelements*

Element	Required/ Optional	Description
<code><scheme-name></code>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<code><scheme-ref></code>	Optional	Specifies the name of another scheme to inherit from. See "Using Scheme Inheritance" on page 11-9 for more information.
<code><class-name></code>	Optional	Specifies a custom implementation of the read write backing map. Any custom implementation must extend the <code>com.tangosol.net.cache.ReadWriteBackingMap</code> class and declare the exact same set of public constructors.

Table B-51 (Cont.) read-write-backing-map-scheme Subelements

Element	Required/ Optional	Description
<code><init-params></code>	Optional	Specifies initialization parameters, for use in custom read write backing map implementations which implement the <code>com.tangosol.run.xml.XmlConfigurable</code> interface.
<code><listener></code>	Optional	Specifies an implementation of a <code>com.tangosol.util.MapListener</code> which will be notified of events occurring on the cache.
<code><cachestore-scheme></code>	Optional	Specifies the store to cache. If unspecified the cached data will only reside within the internal cache (see <code><internal-cache-scheme></code> subelement), and only reflect operations performed on the cache itself.
<code><cachestore-timeout></code>	Optional	<p>Specifies the timeout interval to use for <code>CacheStore</code> read and write operations. If a <code>CacheStore</code> operation times out, the executing thread is interrupted and may ultimately lead to the termination of the cache service. Timeouts of asynchronous <code>CacheStore</code> operations (for example, refresh-ahead, write-behind) will not result in service termination. The value of this element must be in the following format:</p> <p><code>[\d] + [[.] [\d] +] ? [MS ms S s M m H h D d] ?</code></p> <p>where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) <p>If the value does not contain a unit, a unit of milliseconds is assumed. If 0 is specified, the default service-guardian <code><timeout-milliseconds></code> value is used. The default value if none is specified is 0.</p>
<code><internal-cache-scheme></code>	Required	<p>Specifies a cache-scheme which will be used to cache entries. Legal values are:</p> <ul style="list-style-type: none"> ■ <code>local-scheme</code> ■ <code>disk-scheme</code> ■ <code>external-scheme</code> ■ <code>paged-external-scheme</code> ■ <code>overflow-scheme</code> ■ <code>class-scheme</code>
<code><miss-cache-scheme></code>	Optional	<p>Specifies a cache-scheme for maintaining information on cache misses. The miss-cache is used track keys which were not found in the cache store. The knowledge that a key is not in the cache store allows some operations to perform faster, as they can avoid querying the potentially slow cache store. A size-limited scheme may be used to control how many misses are cached. If unspecified no cache-miss data will be maintained. Legal values are:</p> <ul style="list-style-type: none"> ■ <code>local-scheme</code>

Table B–51 (Cont.) read-write-backing-map-scheme Subelements

Element	Required/ Optional	Description
<read-only>	Optional	Specifies if the cache is read only. If <code>true</code> the cache will load data from cachestore for read operations and will not perform any writing to the cachestore when the cache is updated. Legal values are <code>true</code> or <code>false</code> . Default value is <code>false</code> .
<write-delay>	Optional	<p>Specifies the time interval for a write-behind queue to defer asynchronous writes to the cachestore by. The value of this element must be in the following format:</p> <p><code>[\d] + [[.] [\d] +] ? [MS ms S s M m H h D d] ?</code></p> <p>where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) <p>If the value does not contain a unit, a unit of seconds is assumed. If zero, synchronous writes to the cachestore (without queuing) will take place, otherwise the writes will be asynchronous and deferred by specified time interval after the last update to the value in the cache. Default is zero.</p>
<write-batch-factor>	Optional	<p>The <code>write-batch-factor</code> element is used to calculate the "soft-ripe" time for write-behind queue entries. A queue entry is considered to be "ripe" for a write operation if it has been in the write-behind queue for no less than the write-delay interval. The "soft-ripe" time is the point in time before the actual ripe time after which an entry will be included in a batched asynchronous write operation to the CacheStore (along with all other ripe and soft-ripe entries). In other words, a soft-ripe entry is an entry that has been in the write-behind queue for at least the following duration:</p> <p>$D' = (1.0 - F) * D$ where: D = write-delay interval F = write-batch-factor</p> <p>Conceptually, the write-behind thread uses the following logic when performing a batched update:</p> <ol style="list-style-type: none"> 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 <code>store()</code> (if there is only the single ripe entry) or <code>storeAll()</code> (if there are multiple ripe/soft-ripe entries). 4. The thread then repeats (1). <p>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 <code>storeAll()</code> method. The value of the element is expressed as a percentage of the write-delay interval. Legal values are nonnegative doubles less than or equal to 1.0. Default is zero.</p>

Table B–51 (Cont.) read-write-backing-map-scheme Subelements

Element	Required/ Optional	Description
<write-requeue-threshold>	Optional	Specifies the size of the write-behind queue at which additional actions could be taken. This value controls the frequency of the corresponding log messages. For example, a value of 100 will produce a log message every time the size of the write queue is a multiple of 100. If zero, write-behind requeuing is disabled. Legal values are positive integers or zero. Default is zero.
<refresh-ahead-factor>	Optional	The refresh-ahead-factor element is used to calculate the "soft-expiration" time for cache entries. Soft-expiration is the point in time before the actual expiration after which any access request for an entry will schedule an asynchronous load request for the entry. This attribute is only applicable if the internal cache is a local-scheme , configured with the <expiry-delay> subelement. The value is expressed as a percentage of the internal LocalCache expiration interval. If zero, refresh-ahead scheduling will be disabled. If 1.0, then any get operation will immediately trigger an asynchronous reload. Legal values are nonnegative doubles less than or equal to 1.0. Default value is zero.
<rollback-cachestore-failures>	Optional	Specifies whether exceptions caught during synchronous cachestore operations are rethrown to the calling thread (possibly over the network to a remote member). If the value of this element is false, an exception caught during a synchronous cachestore operation is logged locally and the internal cache is updated. If the value is true, the exception is rethrown to the calling thread and the internal cache is not changed. If the operation was called within a transactional context, this would have the effect of rolling back the current transaction. Legal values are true or false. Default value is false.

remote-addresses

Used in: [tcp-initiator](#)

Description

The `remote-addresses` element contains the address (IP or DNS name) and port of one or more TCP/IP acceptors. The TCP/IP initiator uses this information to establish a connection with a proxy service on remote cluster. TCP/IP acceptors are configured within the [proxy-scheme](#) element. The TCP/IP initiator attempts to connect to the addresses in a random order until either the list is exhausted or a TCP/IP connection is established. See *Oracle Coherence Client Guide* for additional details and example configurations.

The following example configuration instructs the initiator to connect to `192.168.0.2:9099` and `192.168.0.3:9099` in a random order:

```
<remote-addresses>
  <socket-address>
    <address>192.168.0.2</address>
    <port>9099</port>
  </socket-address>
  <socket-address>
    <address>192.168.0.3</address>
    <port>9099</port>
  </socket-address>
</remote-addresses>
```

Elements

[Table B-54](#) describes the elements you can define within the `remote-addresses` element.

Table B-52 *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 <code><socket-address></code> elements can be used within a <code><remote-addresses></code> element.
<address-provider>	Optional	Contains the configuration for a <code>com.tangosol.net.AddressProvider</code> implementation that supplies the address (IP or DNS name) and port on which the TCP/IP acceptor is listening. A <code><remote-addresses></code> element can include either a <code><socket-address></code> element or an <code><address-provider></code> 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-53](#) describes the elements you can define within the `remote-cache-scheme` element.

Table B-53 *remote-cache-scheme Subelements*

Element	Required/ Optional	Description
<code><scheme-name></code>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<code><scheme-ref></code>	Optional	Specifies the name of another scheme to inherit from. See "Using Scheme Inheritance" on page 11-9 for more information.
<code><service-name></code>	Optional	Specifies the name of the service which will manage caches created from this scheme.
<code><operation-bundling></code>	Optional	Specifies the configuration information for a bundling strategy.
<code><initiator-config></code>	Required	Contains the configuration of the connection initiator used by the service to establish a connection with the cluster.

remote-invocation-scheme

Used in: [caching-schemes](#)

Description

The `remote-invocation-scheme` element contains the configuration information necessary to execute tasks within the context of a cluster without having to first join the cluster. This scheme uses Coherence*Extend to connect to the cluster.

Elements

[Table B-54](#) describes the elements you can define within the `remote-invocation-scheme` element.

Table B-54 *remote-invocation-scheme Subelements*

Element	Required/ Optional	Description
<code><scheme-name></code>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<code><scheme-ref></code>	Optional	Specifies the name of another scheme to inherit from. See "Using Scheme Inheritance" on page 11-9 for more information.
<code><service-name></code>	Optional	Specifies the name of the service.
<code><initiator-config></code>	Required	Contains the configuration of the connection initiator used by the service to establish a connection with the cluster.

replicated-scheme

Used in: [caching-schemes](#), [near-scheme](#), [versioned-near-scheme](#), [overflow-scheme](#), [versioned-backing-map-scheme](#)

Description

The replicated scheme defines caches which fully replicate all their cache entries on each cluster nodes running the specified service. See ["Replicated Cache"](#) on page 10-5 for a more detailed description of replicated caches.

Clustered Concurrency Control

Replicated caches support cluster wide key-based locking so that data can be modified in a cluster without encountering the classic missing update problem. Note that any operation made without holding an explicit lock is still atomic but there is no guarantee that the value stored in the cache does not change between atomic operations.

Cache Storage (Backing Map)

Storage for the cache is specified by using the backing-map scheme (see `<backing-map>` subelement). For instance, a replicated cache which uses a [local-scheme](#) for its backing map will result in cache entries being stored in-memory.

Elements

[Table B-55](#) describes the elements you can define within the `replicated-scheme` element.

Table B-55 *replicated-scheme Subelements*

Element	Required/ Optional	Description
<code><scheme-name></code>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<code><scheme-ref></code>	Optional	Specifies the name of another scheme to inherit from. See "Using Scheme Inheritance" on page 11-9 for more information.
<code><service-name></code>	Optional	Specifies the name of the service which will manage caches created from this scheme. Services are configured from within the <code><services></code> element in the <code>tangosol-coherence.xml</code> file. See Appendix A, "Operational Configuration Elements" for more information.
<code><serializer></code>	Optional	Specifies either: the class configuration information for a <code>com.tangosol.io.Serializer</code> implementation used to serialize and deserialize user types, or it references a serializer class configuration that is defined in the operational configuration file (see "serializers" on page A-61).
<code><listener></code>	Optional	Specifies an implementation of a <code>com.tangosol.util.MapListener</code> which will be notified of events occurring on the cache.

Table B–55 (Cont.) replicated-scheme Subelements

Element	Required/ Optional	Description
<backing-map-scheme>	Optional	<p>Specifies what type of cache will be used within the cache server to store the entries. Legal values are:</p> <ul style="list-style-type: none"> ■ local-scheme ■ external-scheme ■ paged-external-scheme ■ overflow-scheme ■ class-scheme <p>To ensure cache coherence, the backing-map of an replicated cache must not use a read-through pattern to load cache entries. Either use a cache-aside pattern from outside the cache service, or switch to the distributed-scheme, which supports read-through clustered caching.</p>
<standard-lease-milliseconds>	Optional	<p>Specifies the duration of the standard lease in milliseconds. When a lease has aged past this number of milliseconds, the lock will automatically be released. Set this value to zero to specify a lease that never expires. The purpose of this setting is to avoid deadlocks or blocks caused by stuck threads; the value should be set higher than the longest expected lock duration (for example, higher than a transaction timeout). It's also recommended to set this value higher than <code>packet-delivery/timeout-milliseconds</code> value. Legal values are from positive long numbers or zero. Default value is the value specified for <code>packet-delivery/timeout-milliseconds</code> in the <code>tangosol-coherence.xml</code> descriptor. See "ReplicatedCache Service Parameters" on page A-69 for more information.</p>
<lease-granularity>	Optional	<p>Specifies the lease ownership granularity. Available since release 2.3. Legal values are:</p> <ul style="list-style-type: none"> ■ <code>thread</code> ■ <code>member</code> <p>A value of <code>thread</code> means that locks are held by a thread that obtained them and can only be released by that thread. A value of <code>member</code> means that locks are held by a cluster node and any thread running on the cluster node that obtained the lock can release it. Default value is the <code>lease-granularity</code> value specified in the <code>tangosol-coherence.xml</code> descriptor. See "ReplicatedCache Service Parameters" on page A-69 for more information.</p>
<mobile-issues>	Optional	<p>Specifies whether the lease issues should be transferred to the most recent lock holders. Legal values are <code>true</code> or <code>false</code>. Default value is the <code>mobile-issue</code> value specified in the <code>tangosol-coherence.xml</code> descriptor. See "ReplicatedCache Service Parameters" on page A-69 for more information.</p>

Table B–55 (Cont.) replicated-scheme Subelements

Element	Required/ Optional	Description
<guardian-timeout>	Optional	<p>Specifies the guardian timeout value to use for guarding the service and any dependant 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>.</p> <p>The value of this element must be in the following format:</p> <pre>[\d] + [[.] [\d] +] ? [MS ms S s M m H h D d] ?</pre> <p>where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) <p>If the value does not contain a unit, a unit of milliseconds is assumed.</p>
<service-failure-policy>	Optional	<p>Specifies the action to take when an abnormally behaving service thread cannot be terminated gracefully by the service guardian.</p> <p>Legal values are:</p> <ul style="list-style-type: none"> ■ <code>exit-cluster</code> – 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. ■ <code>exit-process</code> – 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. ■ <code>logging</code> – 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 <code>com.tangosol.net.ServiceFailurePolicy</code> implementation. <p>Default value is <code>exit-cluster</code>.</p>
<member-listener>	Optional	<p>Specifies the configuration information for a class that implements the <code>com.tangosol.net.MemberListener</code> interface. The implementation must have a public default constructor.</p> <p>The <code>MemberListener</code> implementation receives cache service lifecycle events. The <member-listener> element is used as an alternative to programmatically adding a <code>MapListener</code> on a service.</p>
<autostart>	Optional	<p>The <code>autostart</code> element is intended to be used by cache servers (that is, <code>com.tangosol.net.DefaultCacheServer</code>). It specifies whether the cache services associated with this cache scheme should be automatically started at a cluster node. Legal values are <code>true</code> or <code>false</code>. Default value is <code>false</code>.</p>

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 literal or a full configuration. The preferred approach is to reference a full configuration which is defined in the operational configuration file. By default, the operational configuration file contains two `serializer` class definitions: one for Java (default) and one for POF. See "[serializers](#)" on page A-61.

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-56](#) describes the elements you can define within the `serializer` element.

Table B-56 *serializer Subelements*

Element	Required/ Optional	Description
<instance>	Optional	Contains the class configuration information for a <code>com.tangosol.io.Serializer</code> implementation.

socket-address

Used in: [remote-addresses](#)

Description

The `socket-address` element specifies the address and port on which a TCP/IP acceptor is listening.

Elements

[Table B-57](#) describes the subelements you can define within the `socket-address` element.

Table B-57 *socket-address Subelements*

Element	Required/ Optional	Description
<address>	Required	Specifies the IP address (IP or DNS name) on which a TCP/IP acceptor socket is listening.
<port>	Required	Specifies the port on which a TCP/IP acceptor socket is listening. Legal values are from 1 to 65535.

socket-provider

Used in: [tcp-acceptor](#), [tcp-initiator](#), [defaults](#).

Description

The `<socket-provider>` element contains the configuration information for a socket and channel factory that implements the `com.tangosol.net.SocketProvider` interface. The socket providers configured within the `<tcp-acceptor>` and `<tcp-initiator>` elements are for use with Coherence*Extend. Socket providers for TCMP are configured in an operational override for within the `<unicast-listener>` element.

The `<socket-provider>` element accepts either a literal or a full configuration. The preferred approach is to reference a configuration which is defined in the operational configuration file. See ["socket-providers"](#) on page A-77.

Out-of-box, the operational configuration file contains two socket provider configurations: `system` (default) and `ssl`. Additional socket providers can be defined in an operational override file as required. Socket provider configurations are referred to using their `id` attribute name. The following example refers to the pre-defined SSL socket provider configuration:

```
<socket-provider>ssl</socket-provider>
```

Preconfigured override is `tangosol.coherence.socketprovider`. See [Appendix C, "Command Line Overrides"](#) for more information.

Elements

[Table B-58](#) describes the subelements you can define within the `socket-provider` element.

Table B-58 *socket-provider Subelements*

Element	Required/ Optional	Description
<code><system></code>	Optional	Specifies a socket provider that produces instances of the JVM's default socket and channel implementations.
<code><ssl></code>	Optional	Specifies a socket provider that produces socket and channel implementations which utilize SSL.
<code><instance></code>	Optional	Contains the class configuration information for a <code>com.tangosol.net.SocketProvider</code> implementation.

ssl

Used in: [socket-provider](#).

Description

The `<ssl>` element contains the configuration information for a socket provider that produces socket and channel implementations which utilize SSL.

Elements

[Table B-59](#) describes the elements you can define within the `ssl` element.

Table B-59 *ssl Subelements*

Element	Required/ Optional	Description
<code><protocol></code>	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.
<code><provider></code>	Optional	Specifies the configuration for a security provider instance.
<code><executor></code>	Optional	Specifies the configuration information for an implementation of the <code>java.util.concurrent.Executor</code> interface. A <code><class-name></code> subelement is used to provide the name of a class that implements the <code>Executor</code> interface. As an alternative, a <code><class-factory-name></code> subelement can be used to specify a factory class for creating <code>Executor</code> instances and a <code><method-name></code> subelement that specifies the name of a static factory method on the factory class which will perform object instantiation. Either approach can specify initialization parameters using the <code><init-params></code> element.
<code><identity-manager></code>	Optional	Specifies the configuration information for initializing an identity manager instance.
<code><trust-manager></code>	Optional	Specifies the configuration information for initializing a trust manager instance.
<code><hostname-verifier></code>	Optional	Specifies the configuration information for an implementation of the <code>javax.net.ssl.HostnameVerifier</code> interface. During the SSL handshake, if the URL's hostname and the server's identification hostname mismatch, the verification mechanism will call back to this instance to determine if the connection should be allowed. A <code><class-name></code> subelement is used to provide the name of a class that implements the <code>HostnameVerifier</code> interface. As an alternative, a <code><class-factory-name></code> subelement can be used to specify a factory class for creating <code>HostnameVerifier</code> instances and a <code><method-name></code> subelement that specifies the name of a static factory method on the factory class which will perform object instantiation. Either approach can specify initialization parameters using the <code><init-params></code> element.

tcp-acceptor

Used in: [acceptor-config](#).

Description

The `tcp-acceptor` element specifies the configuration information for a connection acceptor that accepts connections from Coherence*Extend clients over TCP/IP. See *Oracle Coherence Client Guide* for additional details and example configurations.

Elements

[Table B-60](#) describes the elements you can define within the `tcp-acceptor` element.

Table B-60 *tcp-acceptor Subelements*

Element	Required/ Optional	Description
<code><local-address></code>	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.
<code><address-provider></code>	Optional	Contains the configuration for a <code>com.tangosol.net.AddressProvider</code> implementation that supplies the local address (IP or DNS name) and port on which the TCP/IP server socket (opened by the connection acceptor) listens. A <code><tcp-acceptor></code> element can include either a <code><local-address></code> or an <code><address-provider></code> element but not both.
<code><socket-provider></code>	Optional	Specifies the configuration information for a socket and channel factory that implements the <code>com.tangosol.net.SocketProvider</code> interface.
<code><reuse-address></code>	Optional	Specifies whether or not a TCP/IP socket can be bound to an in-use or recently used address. This setting is deprecated because the resulting behavior is significantly different across operating system implementations. The JVM will, in general, select a reasonable default which is safe for the target operating system. Valid values are <code>true</code> and <code>false</code> . The default value depends on the operating system.
<code><keep-alive-enabled></code>	Optional	Indicates whether keep alive (<code>SO_KEEPALIVE</code>) is enabled on a TCP/IP socket. Valid values are <code>true</code> and <code>false</code> . Keep alive is enabled by default.
<code><tcp-delay-enabled></code>	Optional	Indicates whether TCP delay (Nagle's algorithm) is enabled on a TCP/IP socket. Valid values are <code>true</code> and <code>false</code> . TCP delay is disabled by default.

Table B–60 (Cont.) tcp-acceptor Subelements

Element	Required/ Optional	Description
<receive-buffer-size>	Optional	<p>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:</p> <p><code>[\d]+[.][\d]+?[K k M m G g]?[B b]?</code></p> <p>where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied:</p> <ul style="list-style-type: none"> ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) <p>If the value does not contain a factor, a factor of one is assumed. Default value is O/S dependent.</p>
<send-buffer-size>	Optional	<p>Configures the size of the underlying TCP/IP socket network send buffer. The value of this element must be in the following format:</p> <p><code>[\d]+[.][\d]+?[K k M m G g]?[B b]?</code></p> <p>where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied:</p> <ul style="list-style-type: none"> ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) <p>If the value does not contain a factor, a factor of one is assumed. Default value is O/S dependent.</p>
<listen-backlog>	Optional	<p>Configures the size of the TCP/IP server socket backlog queue. Valid values are positive integers. Default value is O/S dependent.</p>
<linger-timeout>	Optional	<p>Enables <code>SO_LINGER</code> on a TCP/IP socket with the specified linger time. The value of this element must be in the following format:</p> <p><code>[\d]+[.][\d]+?[MS ms S s M m H h D d]?</code></p> <p>where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) <p>If the value does not contain a unit, a unit of milliseconds is assumed. Linger is disabled by default.</p>
<authorized-hosts>	Optional	<p>A collection of IP addresses of TCP/IP initiator hosts that are allowed to connect to this TCP/IP acceptor.</p>
<suspect-protocol-enabled>	Optional	<p>Specifies whether or not the suspect protocol is enabled in order to detect and close rogue Coherence*Extend client connections. The suspect protocol is enabled by default.</p> <p>Valid values are <code>true</code> and <code>false</code>.</p>

Table B–60 (Cont.) tcp-acceptor Subelements

Element	Required/ Optional	Description
<suspect-buffer-size>	Optional	<p>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 in order to protect the proxy server from running out of memory.</p> <p>The value of this element must be in the following format:</p> <p><code>[\d] + [[.] [\d] +] ? [K k M m G g T t] ? [B b] ?</code></p> <p>where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied:</p> <ul style="list-style-type: none"> ■ K or k (kilo, 2¹⁰) ■ M or m (mega, 2²⁰) ■ G or g (giga, 2³⁰) ■ T or t (tera, 2⁴⁰) <p>If the value does not contain a factor, a factor of one is assumed. Default value is 10000000.</p>
<suspect-buffer-length>	Optional	<p>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 in order to protect the proxy server from running out of memory.</p> <p>Default value is 10000.</p>
<nominal-buffer-size>	Optional	<p>Specifies the outgoing connection backlog (in bytes) at which point a suspect client connection is no longer considered to be suspect.</p> <p>The value of this element must be in the following format:</p> <p><code>[\d] + [[.] [\d] +] ? [K k M m G g T t] ? [B b] ?</code></p> <p>where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied:</p> <ul style="list-style-type: none"> ■ K or k (kilo, 2¹⁰) ■ M or m (mega, 2²⁰) ■ G or g (giga, 2³⁰) ■ T or t (tera, 2⁴⁰) <p>If the value does not contain a factor, a factor of one is assumed. Default value is 2000000.</p>

Table B–60 (Cont.) tcp-acceptor Subelements

Element	Required/ Optional	Description
<nominal-buffer-length>	Optional	<p>Specifies the outgoing connection backlog (in messages) at which point a suspect client connection is no longer considered to be suspect.</p> <p>Default value is 2000.</p>
<limit-buffer-size>	Optional	<p>Specifies the outgoing connection backlog (in bytes) at which point the corresponding client connection must be closed in order to protect the proxy server from running out of memory.</p> <p>The value of this element must be in the following format:</p> <p><code>[\d]+[.[\d]]?[K M G T]?[B b]?</code></p> <p>where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied:</p> <ul style="list-style-type: none"> ■ K or k (kilo, 2¹⁰) ■ M or m (mega, 2²⁰) ■ G or g (giga, 2³⁰) ■ T or t (tera, 2⁴⁰) <p>If the value does not contain a factor, a factor of one is assumed.</p> <p>Default value is 100000000.</p>
<limit-buffer-length>	Optional	<p>Specifies the outgoing connection backlog (in messages) at which point the corresponding client connection must be closed in order to protect the proxy server from running out of memory.</p> <p>Default value is 60000.</p>

tcp-initiator

Used in: [initiator-config](#).

Description

The `tcp-initiator` element specifies the configuration information for a connection initiator that enables Coherence*Extend clients to connect to a remote cluster by using TCP/IP. See *Oracle Coherence Client Guide* for additional details and example configurations.

Elements

[Table B-61](#) describes the elements you can define within the `tcp-initiator` element.

Table B-61 *tcp-initiator Subelements*

Element	Required/Optional	Description
<code><local-address></code>	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.
<code><remote-addresses></code>	Required	Contains the address of one or more TCP/IP connection acceptors. The TCP/IP connection initiator uses this information to establish a TCP/IP connection with a remote cluster.
<code><socket-provider></code>	Optional	Specifies the configuration information for a socket and channel factory that implements the <code>com.tangosol.net.SocketProvider</code> interface.
<code><reuse-address></code>	Optional	<p>Specifies whether or not a TCP/IP socket can be bound to an in-use or recently used address.</p> <p>This setting is deprecated because the resulting behavior is significantly different across operating system implementations. The JVM will, in general, select a reasonable default which is safe for the target operating system.</p> <p>Valid values are <code>true</code> and <code>false</code>. The default value depends on the operating system.</p>
<code><keep-alive-enabled></code>	Optional	Indicates whether keep alive (<code>SO_KEEPALIVE</code>) is enabled on a TCP/IP socket. Valid values are <code>true</code> and <code>false</code> . Keep alive is enabled by default.
<code><tcp-delay-enabled></code>	Optional	Indicates whether TCP delay (Nagle's algorithm) is enabled on a TCP/IP socket. Valid values are <code>true</code> and <code>false</code> . TCP delay is disabled by default.
<code><receive-buffer-size></code>	Optional	<p>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:</p> <pre>[\d] + [[.] [\d] +] ? [K k M m G g] ? [B b] ?</pre> <p>where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied:</p> <ul style="list-style-type: none"> ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) <p>If the value does not contain a factor, a factor of one is assumed. Default value is O/S dependent.</p>

Table B–61 (Cont.) tcp-initiator Subelements

Element	Required/ Optional	Description
<send-buffer-size>	Optional	<p>Configures the size of the underlying TCP/IP socket network send buffer. The value of this element must be in the following format:</p> <p>$[\backslash d] + [[.][\backslash d]^+] ? [K k M m G g] ? [B b] ?$</p> <p>where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied:</p> <ul style="list-style-type: none"> ■ K or k (kilo, 210) ■ M or m (mega, 220) ■ G or g (giga, 230) <p>If the value does not contain a factor, a factor of one is assumed. Default value is O/S dependent.</p>
<connect-timeout>	Optional	<p>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:</p> <p>$[\backslash d] + [[.][\backslash d]^+] ? [MS ms S s M m H h D d] ?$</p> <p>where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) <p>If the value does not contain a unit, a unit of milliseconds is assumed. Default value is an infinite timeout.</p>
<linger-timeout>	Optional	<p>Enables <code>SO_LINGER</code> on a TCP/IP socket with the specified linger time. The value of this element must be in the following format:</p> <p>$[\backslash d] + [[.][\backslash d]^+] ? [MS ms S s M m H h D d] ?$</p> <p>where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) <p>If the value does not contain a unit, a unit of milliseconds is assumed. Linger is disabled by default.</p>

transactional-scheme

Used in [caching-schemes](#), [version-persistent-scheme](#), [version-transient-scheme](#).

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-62](#) describes the elements you can define within the `transactional-scheme` element.

Table B-62 *transactional-scheme Subelements*

Element	Required/ Optional	Description
<code><scheme-name></code>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<code><scheme-ref></code>	Optional	Specifies the name of another scheme to inherit from. See "Using Scheme Inheritance" on page 11-9 for more information.
<code><service-name></code>	Optional	Specifies the name of the service which manages caches created from this scheme. The default service name if no service name is provided is <code>TransactionalCache</code> .
<code><serializer></code>	Optional	Specifies either: the class configuration information for a <code>com.tangosol.io.Serializer</code> implementation used to serialize and deserialize user types, or it references a serializer class configuration that is defined in the operational configuration file (see "serializers" on page A-61).
<code><thread-count></code>	Optional	<p>Specifies the number of daemon threads used by the partitioned cache service. If zero, all relevant tasks are performed on the service thread. Legal values are positive integers or zero. Default value is the <code>thread-count</code> value specified in the <code>tangosol-coherence.xml</code> descriptor. See the <code>thread-count</code> parameter in "DistributedCache Service Parameters" on page A-65 for more information.</p> <p>Specifying the thread-count value will change the default behavior of the Transactional Framework's internal transaction caches that are used for transactional storage and recovery.</p>

Table B–62 (Cont.) transactional-scheme Subelements

Element	Required/ Optional	Description												
<local-storage>	Optional	<p>Specifies whether a cluster node will contribute storage to the cluster, that is, maintain partitions. When disabled the node is considered a cache client.</p> <p>Normally this value should be left unspecified within the configuration file, and instead set on a per-process basis using the tangosol.coherence.distributed.localstorage system property. This allows cache clients and servers to use the same configuration descriptor.</p> <p>Legal values are true or false. Default value is the local-storage value specified in the tangosol-coherence.xml descriptor. See the local-storage parameter in "DistributedCache Service Parameters" on page A-65 for more information.</p>												
<partition-count>	Optional	<p>Specifies the number of partitions that a partitioned (distributed) cache will be "chopped up" into. Each member running the partitioned cache service that has the local-storage (<local-storage> subelement) option set to true will manage a "fair" (balanced) number of partitions.</p> <p>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 in size.</p> <p>The following are good defaults for sample service storage sizes:</p> <table><tr><th>service storage</th><th>partition-count</th></tr><tr><td>100M</td><td>257</td></tr><tr><td>1G</td><td>509</td></tr><tr><td>10G</td><td>2039</td></tr><tr><td>50G</td><td>4093</td></tr><tr><td>100G</td><td>8191</td></tr></table> <p>A list of first 1,000 primes can be found at http://primes.utm.edu/lists/</p> <p>Valid values are positive integers. Default value is the value specified in the tangosol-coherence.xml descriptor. See the partition-count parameter "DistributedCache Service Parameters" on page A-65 for more information.</p>	service storage	partition-count	100M	257	1G	509	10G	2039	50G	4093	100G	8191
service storage	partition-count													
100M	257													
1G	509													
10G	2039													
50G	4093													
100G	8191													
<high-units>	Optional	<p>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.</p> <p>The value of this element must be in the following format:</p> <p>[\\d][+][[.][\\d]+]?[K k M m G g T t]?[B b]?</p> <p>where the first non-digit (from left to right) indicates the factor with which the preceding decimal value should be multiplied:</p> <ul style="list-style-type: none">■ K or k (kilo, 2^10)■ M or m (mega, 2^20)■ G or g (giga, 2^30)■ T or t (tera, 2^40) <p>If the value does not contain a factor, a factor of one is assumed. Default value is 10MB.</p>												

Table B–62 (Cont.) transactional-scheme Subelements

Element	Required/ Optional	Description
<transfer-threshold>	Optional	Specifies the threshold for the primary buckets distribution in kilo-bytes. When a new node joins the partitioned cache service or when a member of the service leaves, the remaining nodes perform a task of bucket ownership re-distribution. During this process, the existing data gets re-balanced along with the ownership information. This parameter indicates a preferred message size for data transfer communications. Setting this value lower will make the distribution process take longer, but will reduce network bandwidth utilization during this activity. Legal values are integers greater than zero. Default value is the transfer-threshold value specified in the tangosol-coherence.xml descriptor. See the transfer-threshold parameter in "DistributedCache Service Parameters" on page A-65 for more information.
<backup-count>	Optional	Specifies the number of members of the partitioned cache service that hold the backup data for each unit of storage in the cache. Value of 0 means that in the case of abnormal termination, some portion of the data in the cache will be lost. Value of N means that if up to N cluster nodes terminate immediately, the cache data will be preserved. To maintain the partitioned cache of size M, the total memory usage in the cluster does not depend on the number of cluster nodes and will be in the order of M*(N+1). Recommended values are 0 or 1. Default value is the backup-count value specified in the tangosol-coherence.xml descriptor. See the backup-count parameter in value specified in the tangosol-coherence.xml descriptor. See "DistributedCache Service Parameters" on page A-65 for more information.
<task-hung-threshold>	Optional	Specifies the amount of time in milliseconds that a task can execute before it is considered "hung". Note: a posted task that has not yet started is never considered as hung. This attribute is applied only if the Thread pool is used (the thread-count value is positive). Legal values are positive integers or zero (indicating no default timeout). Default value is the task-hung-threshold value specified in the tangosol-coherence.xml descriptor. See the task-hung-threshold parameter in "DistributedCache Service Parameters" on page A-65 for more information.
<task-timeout>	Optional	Specifies the timeout value in milliseconds for requests executing on the service worker threads. This attribute is applied only if the thread pool is used (the thread-count value is positive). If zero is specified, the default service-guardian <timeout-milliseconds> value is used. Legal values are non-negative integers. Default value is the value specified in the tangosol-coherence.xml descriptor. See the task-timeout parameter in "DistributedCache Service Parameters" on page A-65.

Table B–62 (Cont.) transactional-scheme Subelements

Element	Required/ Optional	Description
<code><request-timeout></code>	Optional	<p>Specifies the maximum amount of time a client will wait for a response before abandoning the original request. The request time is measured on the client side as the time elapsed from the moment a request is sent for execution to the corresponding server node(s) and includes the following:</p> <ul style="list-style-type: none"> 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 <p>Legal values are positive integers or zero (indicating no default timeout). Default value is the value specified in the <code>tangosol-coherence.xml</code> descriptor. See the <code>request-timeout</code> parameter in "DistributedCache Service Parameters" on page A-65 for more information.</p>
<code><guardian-timeout></code>	Optional	<p>Specifies the guardian timeout value to use for guarding the service and any dependant threads. If the element is not specified for a given service, the default guardian timeout (as specified by the <code><timeout-milliseconds></code> operational configuration element) is used. See <service-guardian>.</p> <p>The value of this element must be in the following format:</p> <pre>[\\d]+[\\.][\\d]+]?[MS ms S s M m H h D d]?</pre> <p>where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> MS or ms (milliseconds) S or s (seconds) M or m (minutes) H or h (hours) D or d (days) <p>If the value does not contain a unit, a unit of milliseconds is assumed.</p>

Table B–62 (Cont.) transactional-scheme Subelements

Element	Required/ Optional	Description
<code><service-failure-policy></code>	Optional	<p>Specifies the action to take when an abnormally behaving service thread cannot be terminated gracefully by the service guardian.</p> <p>Legal values are:</p> <ul style="list-style-type: none"> ■ <code>exit-cluster</code> – 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. ■ <code>exit-process</code> – 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. ■ <code>logging</code> – causes any detected problems to be logged, but no corrective action to be taken. ■ a custom class – an <code><instance></code> subelement is used to provide the class configuration information for a <code>com.tangosol.net.ServiceFailurePolicy</code> implementation. <p>Default value is <code>exit-cluster</code>.</p>
<code><partitioned-quorum-policy-scheme></code>	Optional	Specifies quorum policy settings for the partitioned cache service.
<code><autostart></code>	Optional	The element is intended to be used by cache servers (that is, <code>com.tangosol.net.DefaultCacheServer</code>). It specifies whether the cache services associated with this cache scheme should be automatically started at a cluster node. Legal values are <code>true</code> or <code>false</code> . Default value is <code>false</code> .

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 will contain at least one child element.

Elements

[Table B–63](#) describes the elements you can define within the `trust-manager` element.

Table B–63 *trust-manager Subelements*

Element	Required/ Optional	Description
<code><algorithm></code>	Optional	Specifies the algorithm used by the trust manager. The default value is <code>SunX509</code> .
<code><provider></code>	Optional	Specifies the configuration for a security provider instance.
<code><key-store></code>	Optional	Specifies the configuration for a key store implementation.

version-persistent-scheme

Used in: [versioned-backing-map-scheme](#).

Description

The `version-persistent-scheme` defines a cache for storing object versioning information in a clustered cache. Specifying a size limit on the specified scheme's backing-map allows control over how many version identifiers are tracked.

Elements

[Table B-64](#) describes the elements you can define within the `version-persistent-scheme` element.

Table B-64 *persistent-scheme Subelements*

Element	Required/ Optional	Description
<code><cache-name-suffix></code>	Optional	Specifies the name modifier that is used to create a cache of version objects associated with a given cache. The value of this element is appended to the base cache name. Legal value is a string. Default value is <code>-persist</code> . For example, if the base case is named <code>Sessions</code> and this name modifier is set to <code>-persist</code> , the associated version cache will be named <code>Sessions-persist</code> .
<code><replicated-scheme></code> , <code><distributed-scheme></code> , or <code><transactional-scheme></code>	Required	Specifies the scheme for the cache used to maintain the versioning information. Legal values are: <ul style="list-style-type: none"> ▪ <code>replicated-scheme</code> ▪ <code>distributed-scheme</code> ▪ <code>transactional-scheme</code>

version-transient-scheme

Used in: [versioned-near-scheme](#), [versioned-backing-map-scheme](#).

Description

The `version-transient-scheme` defines a cache for storing object versioning information for use in versioned near-caches. Specifying a size limit on the specified scheme's backing-map allows control over how many version identifiers are tracked.

Elements

The following table describes the elements you can define within the `version-transient-scheme` element.

Table B–65 *transient-scheme Subelements*

Element	Required/ Optional	Description
<code><cache-name-suffix></code>	Optional	Specifies the name modifier that is used to create a cache of version objects associated with a given cache. The value of this element is appended to the base cache name. Legal value is a string. Default value is "-version". For example, if the base case is named <code>Sessions</code> and this name modifier is set to <code>-version</code> , the associated version cache will be named <code>Sessions-version</code> .
<code><replicated-scheme></code> , <code><distributed-scheme></code> , or <code>transactional-scheme</code>	Required	Specifies the scheme for the cache used to maintain the versioning information. Legal values are: <ul style="list-style-type: none">■ <code>replicated-scheme</code>■ <code>distributed-scheme</code>■ <code>transactional-scheme</code>

versioned-backing-map-scheme

Used in: [caching-schemes](#), [distributed-scheme](#), [replicated-scheme](#), [optimistic-scheme](#).

Description

The `versioned-backing-map-scheme` is an extension of a [read-write-backing-map-scheme](#), defining a size limited cache of a persistent store. It uses object versioning to determine what updates need to be written to the persistent store. See "[Versioning](#)" for more information.

Implementation

The `versioned-backing-map-scheme` scheme is implemented by the `com.tangosol.net.cache.VersionedBackingMap` class.

Cache of an External Store

As with the [read-write-backing-map-scheme](#), a versioned backing map maintains a cache backed by an external persistent cache store (see `<cachestore-scheme>` subelement), cache misses will read-through to the back-end store to retrieve the data. Cache stores may also support updates to the back-end data store.

Refresh-Ahead and Write-Behind Caching

As with the [read-write-backing-map-scheme](#) both the refresh-ahead (see `<refresh-ahead>` subelement) and write-behind (see `<write-behind>` subelement) caching optimizations are supported. See "Read-Through, Write-Through, Write-Behind, and Refresh-Ahead Caching" in the *Oracle Coherence Getting Started Guide* for more details.

Versioning

For entries whose values implement the `com.tangosol.util.Versionable` interface, the versioned backing map will use the version identifier to determine if an update must be written to the persistent store. The primary benefit of this feature is that in the event of cluster node failover, the backup node can determine if the most recent version of an entry has already been written to the persistent store, and if so it can avoid an extraneous write.

Elements

[Table B-66](#) describes the elements you can define within the `versioned-backing-map-scheme` element.

Table B–66 *versioned-backing-map-scheme* Subelement

Element	Required/ Optional	Description
<scheme-name>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<scheme-ref>	Optional	Specifies the name of another scheme to inherit from. See "Using Scheme Inheritance" on page 11-9 for more information.
<class-name>	Optional	Specifies a custom implementation of the versioned backing map. Any custom implementation must extend the <code>com.tangosol.net.cache.VersionedBackingMap</code> class and declare the exact same set of public constructors.
<init-params>	Optional	Specifies initialization parameters, for use in custom versioned backing map implementations which implement the <code>com.tangosol.run.xml.XmlConfigurable</code> interface.
<listener>	Optional	Specifies an implementation of a <code>com.tangosol.util.MapListener</code> which will be notified of events occurring on the cache.
<cachestore-scheme>	Optional	Specifies the store to cache. If unspecified the cached data will only reside within the (see <internal-cache-scheme> subelement), and only reflect operations performed on the cache itself.
<internal-cache-scheme>	Required	Specifies a cache-scheme which will be used to cache entries. Legal values are: <ul style="list-style-type: none"> ▪ local-scheme ▪ external-scheme ▪ paged-external-scheme ▪ overflow-scheme ▪ class-scheme
<miss-cache-scheme>	Optional	Specifies a cache-scheme for maintaining information on cache misses. The miss-cache is used track keys which were not found in the cache store. The knowledge that a key is not in the cache store allows some operations to perform faster, as they can avoid querying the potentially slow cache store. A size-limited scheme may be used to control how many misses are cached. If unspecified no cache-miss data will be maintained. Legal values are: <ul style="list-style-type: none"> ▪ local-scheme
<read-only>	Optional	Specifies if the cache is read only. If true the cache will load data from cachestore for read operations and will not perform any writing to the cachestore when the cache is updated. Legal values are <code>true</code> or <code>false</code> . Default value is <code>false</code> .

Table B–66 (Cont.) versioned-backing-map-scheme Subelement

Element	Required/ Optional	Description
<code><write-delay></code>	Optional	<p>Specifies the time interval for a write-behind queue to defer asynchronous writes to the cachestore by. The value of this element must be in the following format:</p> <pre>[\d] + [[.] [\d] +] ? [MS ms S s M m H h D d] ?</pre> <p>where the first non-digits (from left to right) indicate the unit of time duration:</p> <ul style="list-style-type: none"> ■ MS or ms (milliseconds) ■ S or s (seconds) ■ M or m (minutes) ■ H or h (hours) ■ D or d (days) <p>If the value does not contain a unit, a unit of seconds is assumed. If zero, synchronous writes to the cachestore (without queuing) will take place, otherwise the writes will be asynchronous and deferred by the number of seconds after the last update to the value in the cache. Default is zero.</p>
<code><write-batch-factor></code>	Optional	<p>The write-batch-factor element is used to calculate the "soft-ripe" time for write-behind queue entries. A queue entry is considered to be "ripe" for a write operation if it has been in the write-behind queue for no less than the write-delay interval. The "soft-ripe" time is the point in time before the actual "ripe" time after which an entry will be included in a batched asynchronous write operation to the CacheStore (along with all other "ripe" and "soft-ripe" entries). This element is only applicable if asynchronous writes are enabled (that is, the value of the write-delay element is greater than zero) and the CacheStore implements the <code>storeAll()</code> method. The value of the element is expressed as a percentage of the write-delay interval. For example, if the value is zero, only "ripe" entries from the write-behind queue will be batched. On the other hand, if the value is 1.0, all currently queued entries will be batched and the value of the write-delay element will be effectively ignored. Legal values are nonnegative doubles less than or equal to 1.0. Default is zero.</p>
<code><write-requeue-threshold></code>	Optional	<p>Specifies the size of the write-behind queue at which additional actions could be taken. This value controls the frequency of the corresponding log messages. For example, a value of 100 will produce a log message every time the size of the write queue is a multiple of 100. If zero, write-behind requeuing is disabled. Legal values are positive integers or zero. Default is zero.</p>
<code><refresh-ahead-factor></code>	Optional	<p>The <code>refresh-ahead-factor</code> element is used to calculate the "soft-expiration" time for cache entries. Soft-expiration is the point in time before the actual expiration after which any access request for an entry will schedule an asynchronous load request for the entry. This attribute is only applicable if the internal cache (see <code><internal-cache-scheme></code> subelement) is a local-scheme, configured with the <code><location></code> subelement. The value is expressed as a percentage of the internal LocalCache expiration interval. If zero, refresh-ahead scheduling will be disabled. If 1.0, then any get operation will immediately trigger an asynchronous reload. Legal values are nonnegative doubles less than or equal to 1.0. Default value is zero.</p>

Table B–66 (Cont.) versioned-backing-map-scheme Subelement

Element	Required/ Optional	Description
<code><rollback-cache-store-failures></code>	Optional	Specifies whether exceptions caught during synchronous cachestore operations are rethrown to the calling thread (possibly over the network to a remote member). If the value of this element is false, an exception caught during a synchronous cachestore operation is logged locally and the internal cache is updated. If the value is true, the exception is rethrown to the calling thread and the internal cache is not changed. If the operation was called within a transactional context, this would have the effect of rolling back the current transaction. Legal values are true or false. Default value is false.
<code><version-persistent-scheme></code>	Optional	Specifies a cache-scheme for tracking the version identifier for entries in the persistent cachestore (see cachestore-scheme).
<code><version-transient-scheme></code>	Optional	Specifies a cache-scheme for tracking the version identifier for entries in the transient internal cache (see <code><internal-cache-scheme></code> subelement).
<code><manage-transient></code>	Optional	Specifies if the backing map is responsible for keeping the transient version cache up to date. If disabled the backing map manages the transient version cache only for operations for which no other party is aware (such as entry expiry). This is used when there is already a transient version cache of the same name being maintained at a higher level, for instance within a versioned-near-scheme . Legal values are true or false. Default value is false.

versioned-near-scheme

Used in: [caching-schemes](#).

Note: As of Coherence release 2.3, it is suggested that a [near-scheme](#) be used instead of `versioned-near-scheme`. Legacy Coherence applications use `versioned-near-scheme` to ensure Coherence through object versioning. As of Coherence 2.3 the `near-scheme` includes a better alternative, in the form of reliable and efficient front cache invalidation.

Description

As with the [near-scheme](#), the `versioned-near-scheme` defines a two tier cache consisting of a small and fast front-end, and higher-capacity but slower back-end cache. The front-end (see `<front-end>` subelement) and back-end (see `<back-end>` subelement) are expressed as normal cache-schemes. A typical deployment might use a [local-scheme](#) for the front-end, and a [distributed-scheme](#) for the back-end. See "Near Cache" on page 10-7 for a more detailed description of versioned near caches.

Implementation

The versioned near scheme is implemented by the `com.tangosol.net.cache.VersionedNearCache` class.

Versioning

Object versioning is used to ensure coherence between the front and back tiers. See the `<version-transient-scheme>` subelement for more information

Elements

[Table B-67](#) describes the elements you can define within the `near-scheme` element.

Table B-67 *versioned-near-scheme Subelements*

Element	Required/ Optional	Description
<code><scheme-name></code>	Optional	Specifies the scheme's name. The name must be unique within a configuration file.
<code><scheme-ref></code>	Optional	Specifies the name of another scheme to inherit from. See "Using Scheme Inheritance" on page 11-9 for more information.
<code><class-name></code>	Optional	Specifies a custom implementation of the versioned near cache. The specified class must extend the <code>com.tangosol.net.cache.VersionedNearCache</code> class and declare the exact same set of public constructors.
<code><init-params></code>	Optional	Specifies initialization parameters, for use in custom versioned near cache implementations which implement the <code>com.tangosol.run.xml.XmlConfigurable</code> interface.
<code><listener></code>	Optional	Specifies an implementation of a <code>com.tangosol.util.MapListener</code> which will be notified of events occurring on the cache.

Table B–67 (Cont.) versioned-near-scheme Subelements

Element	Required/ Optional	Description
<code><front-scheme></code>	Required	<p>Specifies the <code>cache-scheme</code> to use in creating the front-tier cache. Legal values are:</p> <ul style="list-style-type: none"> ▪ local-scheme ▪ external-scheme ▪ paged-external-scheme ▪ class-scheme <p>For example:</p> <pre><front-scheme> <local-scheme> <scheme-ref>default-eviction</scheme-ref> </local-scheme> </front-scheme></pre> <p>or</p> <pre><front-scheme> <class-scheme> <class-name>com.tangosol.util.SafeHashMap</class- name> <init-params></init-params> </class-scheme> </front-scheme></pre>
<code><back-scheme></code>	Required	<p>Specifies the <code>cache-scheme</code> to use in creating the back-tier cache. Legal values are:</p> <ul style="list-style-type: none"> ▪ distributed-scheme ▪ replicated-scheme ▪ optimistic-scheme ▪ local-scheme ▪ external-scheme ▪ paged-external-scheme ▪ class-scheme <p>For Example:</p> <pre><back-scheme> <distributed-scheme> <scheme-ref>default-distributed</scheme-ref> </distributed-scheme> </back-scheme></pre>
<code><version-transient-scheme></code>	Optional	Specifies a scheme for versioning cache entries, which ensures coherence between the front and back tiers.
<code><autostart></code>	Optional	<p>The <code>autostart</code> element is intended to be used by cache servers (that is, <code>com.tangosol.net.DefaultCacheServer</code>). It specifies whether the cache services associated with this cache scheme should be automatically started at a cluster node. Legal values are <code>true</code> or <code>false</code>. Default value is <code>false</code>.</p>

Command Line Overrides

Both the Coherence Operational Configuration deployment descriptor `tangosol-coherence.xml` and the Coherence Cache Configuration deployment descriptor `coherence-cache-config.xml` can assign a Java command line option name to any element defined in the descriptor. Some elements already have these Command Line Setting Overrides defined. You can create your own or change the predefined ones.

This feature is useful when you need to change the settings for a single JVM, or to be able to start different applications with different settings without making them use different descriptors. The most common application is passing a different multicast address and/or port to allow different applications to create separate clusters.

To create a Command Line Setting Override, add a `system-property` attribute, specifying the string you would like to assign as the name for the java command line option to the element you want to create an override to. Then, specify it in the Java command line, prefixed with "-D".

Override Example

For example, to create an override for the IP address of the multi-home server to avoid using the default `localhost`, and instead specify a specific the IP address of the interface we want Coherence to use (for instance, `192.168.0.301`). We would like to call this override `tangosol.coherence.localhost`.

First, add a system-property to the `cluster-config`, `unicast-listener`, or `address` element:

```
<address>localhost</address>
```

which will look as follows with the property we added:

```
<address system-property="tangosol.coherence.localhost">localhost</address>
```

Then use it by modifying the Java command line:

```
java -jar coherence.jar
```

Specify the IP address, `192.168.0.301` (instead of the default `localhost` specified in the configuration) as follows:

```
java -Dtangosol.coherence.localhost=192.168.0.301 -jar coherence.jar
```

Preconfigured Override Values

Table C–1 lists all of the preconfigured override values:

Table C–1 Preconfigured System Property Override Values

Override Option	Setting
<code>tangosol.coherence.cacheconfig</code>	Cache configuration descriptor filename. See "configurable-cache-factory-config" on page A-14.
<code>tangosol.coherence.cluster</code>	Cluster name. See "member-identity" on page A-38.
<code>tangosol.coherence.clusteraddress</code>	Cluster (multicast) IP address. See <code><multicast-listener-address></code> subelement of "multicast-listener" on page A-42.
<code>tangosol.coherence.clusterport</code>	Cluster (multicast) IP port. See <code><multicast-listener-port></code> subelement of "multicast-listener" on page A-42.
<code>tangosol.coherence.distributed.backup</code>	Data backup storage location. See <code>backup-storage/type</code> subelement in "DistributedCache Service Parameters" on page A-65.
<code>tangosol.coherence.distributed.backupcount</code>	Number of data backups. See <code>backup-count</code> subelement in "DistributedCache Service Parameters" on page A-65.
<code>tangosol.coherence.distributed.localstorage</code>	Local partition management enabled. See <code>local-storage</code> subelement in "DistributedCache Service Parameters" on page A-65.
<code>tangosol.coherence.distributed.threads</code>	Thread pool size. See <code>thread-count</code> subelement in "DistributedCache Service Parameters" on page A-65.
<code>tangosol.coherence.distributed.transfer</code>	Partition transfer threshold. See <code>transfer-threshold</code> subelement in "DistributedCache Service Parameters" on page A-65.
<code>tangosol.coherence.edition</code>	Product edition. See "license-config" on page A-28.
<code>tangosol.coherence.invocation.threads</code>	Invocation service thread pool size. See <code>thread-count</code> subelement in "InvocationService Parameters" on page A-70.
<code>tangosol.coherence.localhost</code>	Unicast IP address. See <code><unicast-listener-address></code> subelement in "unicast-listener" on page A-83.
<code>tangosol.coherence.localport</code>	Unicast IP port. See <code><unicast-listener-port></code> subelement in "unicast-listener" on page A-83.
<code>tangosol.coherence.localport.adjust</code>	Unicast IP port auto assignment. See <code><unicast-listener-auto></code> subelement in "unicast-listener" on page A-83.
<code>tangosol.coherence.log</code>	Logging destination. See <code><logging-config-destination></code> subelement in "logging-config" on page A-29.
<code>tangosol.coherence.log.level</code>	Logging level. See <code><logging-config-level></code> subelement in "logging-config" on page A-29.
<code>tangosol.coherence.log.limit</code>	Log output character limit. See <code><logging-config-limit></code> subelement in "logging-config" on page A-29.
<code>tangosol.coherence.machine</code>	Machine name. See "member-identity" on page A-38.

Table C-1 (Cont.) Preconfigured System Property Override Values

Override Option	Setting
<code>tangosol.coherence.management</code>	JMX management mode. See "management-config" on page A-32.
<code>tangosol.coherence.management.readonly</code>	JMX management read-only flag. "management-config" on page A-32.
<code>tangosol.coherence.management.remote</code>	Remote JMX management enabled flag. See "management-config" on page A-32.
<code>tangosol.coherence.member</code>	Member name. See "member-identity" on page A-38.
<code>tangosol.coherence.mode</code>	Operational mode. See "license-config" on page A-28.
<code>tangosol.coherence.override</code>	Deployment configuration override filename.
<code>tangosol.coherence.priority</code>	Priority. See "member-identity" on page A-38.
<code>tangosol.coherence.process</code>	Process name "member-identity" on page A-38.
<code>tangosol.coherence.proxy.threads</code>	Coherence*Extend service thread pool size. See <code>thread-count</code> subelement in "ProxyService Parameters" on page A-71.
<code>tangosol.coherence.rack</code>	Rack name. See "member-identity" on page A-38.
<code>tangosol.coherence.role</code>	Role name. See "member-identity" on page A-38.
<code>tangosol.coherence.security</code>	Cache access security enabled flag. See "security-config" on page A-59.
<code>tangosol.coherence.security.keystore</code>	Security access controller keystore file name. See "security-config" on page A-59.
<code>tangosol.coherence.security.password</code>	Keystore or cluster encryption password. "Encryption Filters" on page 9-1.
<code>tangosol.coherence.security.permissions</code>	Security access controller permissions file name. See "security-config" on page A-59.
<code>tangosol.coherence.shutdownhook</code>	Shutdown listener action. See "shutdown-listener" on page A-73.
<code>tangosol.coherence.site</code>	Site name. See "member-identity" on page A-38.
<code>tangosol.coherence.tcmp.enabled</code>	TCMP enabled flag. See <code><packet-publisher-enabled></code> subelement in "packet-publisher" on page A-52.
<code>tangosol.coherence.tcpring</code>	!TCP ring enabled flag. See "tcp-ring-listener" on page A-79.
<code>tangosol.coherence.ttl</code>	Multicast packet time to live (TTL). See <code><multicast-listener-ttl></code> subelement in "multicast-listener" on page A-42.
<code>tangosol.coherence.wka</code>	Well known IP address. See "well-known-addresses" on page A-86.
<code>tangosol.coherence.wka.port</code>	Well known IP port. See "well-known-addresses" on page A-86.

POF User Type Configuration Elements

This appendix provides a listing of the elements that can be used to specify POF user types. POF user type configuration elements are defined in the `pof-config.dtd` file that can be found in the `coherence.jar` file.

You can find additional information about the POF user type configuration file in the Javadoc for the `ConfigurablePofContext` class.

POF User Type Deployment Descriptor

Use the POF user type deployment descriptor to specify the various user types which are being passed into the cluster.

Document Location

The name and location of the descriptor defaults to `pof-config.xml`. The default POF user type descriptor (packaged in `coherence.jar`) will be used unless a custom file is found within the application's classpath. The name and location of the descriptor can also be configured using system property `tangosol.pof.config`. It is recommended that all nodes within a cluster use identical POF user type descriptors.

Document Root

The root element of the POF user type descriptor is `pof-config`. This is where you may begin specifying your user types.

Document Format

The POF user type descriptor should begin with the following DOCTYPE declaration:

```
<!DOCTYPE pof-config SYSTEM "pof-config.dtd">
```

The format of the document and the nesting of elements is illustrated in [Example D-1](#).

Example D-1 Format of a POF User Type Configuration File (`pof-config.xml`)

```
<pof-config>
  <user-type-list>
    ..
    <user-type>
      <type-id>53</type-id>
      <class-name>com.mycompany.data.Trade</class-name>
      <serializer>
        <class-name>com.tangosol.io.pof.PortableObjectSerializer</class-name>
        <init-params>
```

```
        <init-param>
          <param-type>int</param-type>
          <param-value>{type-id}</param-value>
        </init-param>
      </init-params>
    </serializer>
  </user-type>

  <user-type>
    <type-id>54</type-id>
    <class-name>com.mycompany.data.Position</class-name>
  </user-type>

  ..
</include>file:/my-pof-config.xml</include>

  ..
</user-type-list>

<allow-interfaces>false</allow-interfaces>
<allow-subclasses>false</allow-subclasses>
<default-serializer>
  <class-name>com.mycompany.data.TradeSerializer</class-name>
  <init-params>
    <init-param>
      <param-type>int</param-type>
      <param-value>{type-id}</param-value>
    </init-param>
  </init-params>
</default-serializer>
</pof-config>
```

Command Line Override

Oracle Coherence provides a powerful Command Line Setting Override Feature, which allows any element defined in this descriptor to be overridden from the Java command line if it has a system-property attribute defined in the descriptor.

Element Index

Table D-1 lists all elements which may be used from within a POF user type configuration.

Table D-1 POF Configuration Elements

Element	Used In:
<allow-interfaces>	<pof-config>
<allow-subclasses>	<pof-config>
<class-name>	<user-type>, <serializer>
<default-serializer>	<pof-config>
<include>	<user-type-list>
<init-param>	<init-params>
<init-params>	<serializer>
<param-type>	<init-param>
<param-value>	<init-param>
<pof-config>	<i>root element</i>
<serializer>	<user-type>
<type-id>	<user-type>
<user-type>	<user-type-list>
<user-type-list>	<pof-config>

allow-interfaces

Used in: `<pof-config>`

Description

The `allow-interfaces` element indicates whether the `user-type class-name` can specify Java interface types in addition to Java class types.

Valid values are `true` or `false`. Default value is `false`.

Elements

Terminal element.

allow-subclasses

Used in: `<pof-config>`

Description

The `allow-subclasses` element indicates whether the `user-type class-name` can specify a Java class type that is abstract, and whether sub-classes of any specified `user-type class-name` will be permitted at runtime and automatically mapped to the specified super-class for purposes of obtaining a serializer.

Valid values are `true` or `false`. Default value is `false`.

Elements

Terminal element.

class-name

Used in: `<user-type>`, `<serializer>`, `<default-serializer>`

Description

The `class-name` element specifies the name of a Java class or interface.

Within the `user-type` element, the `class-name` element is required, and specifies the fully qualified name of the Java class or interface that all values of the user type are type-assignable to.

Within the `serializer` element, the `class-name` element is required.

Within the `default-serializer` element, the `class-name` element is required.

Elements

Terminal element.

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 will be 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 will be used. If the serializer for the user type is also omitted, then the user type is assumed to implement the `PortableObject` interface, and the `PortableObjectSerializer` implementation is used as the `PofSerializer`.

If the `init-params` element is omitted from the default serializer element, then the following four constructors are attempted on the specific `PofSerializer` implementation, and in this order:

- `(int nTypeId, Class clz, ClassLoader loader)`
- `(int nTypeId, Class clz)`
- `(int nTypeId)`
- `()`

Elements

[Table D-2](#) describes the elements that can be defined within the `default-serializer` element.

Table D-2 *default-serializer Subelements*

Element	Required/ Optional	Description
<code><class-name></code>	Required	Specifies the name of the <code>PofSerializer</code> implementation.
<code><init-params></code>	Optional	Specifies zero or more arguments (each as an <code>init-param</code>) that correspond to the parameters of a constructor of the class that is being configured.

include

Used in: `<user-type-list>`

Description

The include element specifies the location of a `pof-config` file to load `user-type` elements from. The value is a locator string (either a valid path or URL) that identifies the location of the target `pof-config` file.

Elements

Terminal element.

init-param

Used in: [<init-params>](#)

Description

The `init-param` element provides a type for a configuration parameter and a corresponding value to pass as an argument.

Elements

[Table D-3](#) describes the subelements you can define within the `init-param` element.

Table D-3 *init-param Subelements*

Element	Required/ Optional	Description
<param-type>	Required	<p>The <code>param-type</code> element specifies the Java type of initialization parameter. Supported types are:</p> <ul style="list-style-type: none"> ▪ <code>string</code>—indicates that the value is a <code>java.lang.String</code> ▪ <code>boolean</code>—indicates that the value is a <code>java.lang.Boolean</code> ▪ <code>int</code>—indicates that the value is a <code>java.lang.Integer</code> ▪ <code>long</code>—indicates that the value is a <code>java.lang.Long</code> ▪ <code>double</code>—indicates that the value is a <code>java.lang.Double</code> ▪ <code>decimal</code>—indicates that the value is a <code>java.math.BigDecimal</code> ▪ <code>file</code>—indicates that the value is a <code>java.io.File</code> ▪ <code>date</code>—indicates that the value is a <code>java.sql.Date</code> ▪ <code>time</code>—indicates that the value is a <code>java.sql.Timestamp</code> ▪ <code>datetime</code>—indicates that the value is a <code>java.sql.Timestamp</code> ▪ <code>xml</code>—indicates that the value is the entire <code>init-param</code> <code>XmlElement</code>. <p>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.</p>
<param-value>	Required	<p>The <code>param-value</code> element specifies a value of the initialization parameter. The value is in a format specific to the type of the parameter. There are four reserved values that can be specified. Each of these values is replaced at runtime with a specific runtime value before the constructor is invoked:</p> <ul style="list-style-type: none"> ▪ <code>{type-id}</code>—replaced with the Type ID of the User Type; ▪ <code>{class-name}</code>—replaced with the name of the class for the User Type; ▪ <code>{class}</code>—replaced with the Class for the User Type; ▪ <code>{class-loader}</code>—replaced with the <code>ConfigurablePofContext</code>'s <code>ContextClassLoader</code>.

init-params

Used in: `<serializer>`, `<default-serializer>`

Description

The `init-params` element contains zero or more arguments (each as an `init-param`) that correspond to the parameters of a constructor of the class that is being configured.

Elements

[Table D-4](#) describes the elements you can define within the `init-params` element.

Table D-4 *init-params Subelements*

Element	Required/ Optional	Description
<code><init-param></code>	Required	The <code>init-param</code> element provides a type for a configuration parameter and a corresponding value to pass as an argument.

param-type

Used in: `<init-param>`

Description

The `param-type` element specifies the Java type of initialization parameter.

Supported types are:

- `string`—indicates that the value is a `java.lang.String`
- `boolean`—indicates that the value is a `java.lang.Boolean`
- `int`—indicates that the value is a `java.lang.Integer`
- `long`—indicates that the value is a `java.lang.Long`
- `double`—indicates that the value is a `java.lang.Double`
- `decimal`—indicates that the value is a `java.math.BigDecimal`
- `file`—indicates that the value is a `java.io.File`
- `date`—indicates that the value is a `java.sql.Date`
- `time`—indicates that the value is a `java.sql.Timestamp`
- `datetime`—indicates that the value is a `java.sql.Timestamp`
- `xml`—indicates that the value is the entire `init-param` `XmlElement`.

The value is converted to the specified type, and the target constructor or method must have a parameter of that type in order for the instantiation to succeed.

Elements

Terminal element.

param-value

Used in: `<init-param>`

Description

The `param-value` element specifies a value of the initialization parameter. The value is in a format specific to the type of the parameter.

There are four reserved values that can be specified. Each of these values is replaced at runtime with a specific runtime value before the constructor is invoked:

- `{type-id}`—replaced with the Type ID of the User Type;
- `{class-name}`—replaced with the name of the class for the User Type;
- `{class}`—replaced with the Class for the User Type;
- `{class-loader}`—replaced with the `ConfigurablePofContext`'s `ContextClassLoader`.

Elements

Terminal element.

pof-config

root element

Description

The `pof-config` element is the root element of the POF user type configuration descriptor.

Elements

[Table D-5](#) describes the elements you can define within the `pof-config` element.

Table D-5 *pof-config Subelements*

Element	Required/ Optional	Description
<code><allow-interfaces></code>	Optional	The <code>allow-interfaces</code> element indicates whether the user-type <code>class-name</code> can specify Java interface types in addition to Java class types. Valid values are <code>true</code> or <code>false</code> . Default value is <code>false</code> .
<code><allow-subclasses></code>	Optional	The <code>allow-subclasses</code> element indicates whether the user-type <code>class-name</code> can specify a Java class type that is abstract, and whether sub-classes of any specified user-type <code>class-name</code> will be permitted at runtime and automatically mapped to the specified super-class for purposes of obtaining a serializer. Valid values are <code>true</code> or <code>false</code> . Default value is <code>false</code> .
<code><user-type-list></code>	Required	The <code>user-type-list</code> element contains zero or more user-type elements. Each POF user type that will be used must be listed in the <code>user-type-list</code> . The <code>user-type-list</code> element may also contain zero or more <code>include</code> elements. Each <code>include</code> element is used to add user-type elements defined in another <code>pof-config</code> file.
<code><default-serializer></code>	Optional	The <code>default-serializer</code> specifies what <code>PofSerializer</code> to use to serialize and deserialize all user types defined in the <code>pof-config</code> . If a serializer is specified for a user-type, then that serializer will be used for that user-type instead of the default serializer.

serializer

Used in: [<acceptor-config>](#), [<distributed-scheme>](#), [<initiator-config>](#), [<invocation-scheme>](#), [<optimistic-scheme>](#), [<replicated-scheme>](#), [<user-type>](#)

Description

This element may be used either as part of a service scheme element such as [proxy-scheme/acceptor-config](#), and [distributed-scheme](#), or as part of a [user-type](#) element within a POF configuration file for specifying a POFSerializer.

Usage Within Service Schemes

Specifies the class configuration info for a `com.tangosol.io.Serializer` implementation used by the service to serialize and deserialize user types.

For example, the following configures a `ConfigurablePofContext` that uses the default `coherence-pof-config.xml` configuration file to write objects to and read from a stream:

```
<serializer>
  <class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>
</serializer>
```

Usage Within user-type

The `serializer` element specifies what `PofSerializer` to use to serialize and deserialize a specific user type.

A `PofSerializer` is used to serialize and deserialize user type values to and from a POF stream. Within the `serializer` element, the `class-name` element is required, and zero or more constructor parameters can be defined within an `init-params` element.

If the `serializer` element is omitted, then the user type is assumed to implement the `PortableObject` interface, and the `PortableObjectSerializer` implementation is used as the `PofSerializer`.

If the `init-params` element is omitted from the `serializer` element, then the following four constructors are attempted on the specific `PofSerializer` implementation, in this order:

- `(int nTypeId, Class clz, ClassLoader loader)`
- `(int nTypeId, Class clz)`
- `(int nTypeId)`
- `()`

Elements

[Table D-6](#) describes the elements you can define within the `serializer` element.

Table D–6 *serializer Subelements*

Element	Required/ Optional	Description
<class-name>	Required	Specifies the name of the serializer.
<init-params>	Optional	The <code>init-params</code> element contains zero or more arguments (each as an <code>init-param</code>) that correspond to the parameters of a constructor of the class that is being configured.

type-id

Used in: `<user-type>`

Description

The `type-id` element specifies an integer value ($n \geq 0$) that uniquely identifies the user type.

If none of the `user-type` elements contains a `type-id` element, then the type IDs for the user types will be based on the order in which they appear in the `user-type-list`, with the first user type being assigned the type ID 0, the second user type being assigned the type ID 1, and so on.

However, it is strongly recommended that user types IDs always be specified, to support schema versioning and evolution.

Note: Reserved IDs: The first 1000 IDs are reserved for Coherence internal use.

Elements

Terminal element.

user-type

Used in: [<user-type-list>](#)

Description

The `user-type` element contains the declaration of a POF user type. A POF user type is a uniquely identifiable, portable, versionable object class that can be communicated among systems regardless of language, operating system, hardware and location.

Within the `user-type` element, the `type-id` element is optional, but its use is strongly suggested to support schema versioning and evolution.

Within the `user-type` element, the `class-name` element is required, and specifies the fully qualified name of the Java class or interface that all values of the user type are type-assignable to.

If the `serializer` element is omitted, then the user type is assumed to implement the `PortableObject` interface, and the `PortableObjectSerializer` implementation is used as the `PofSerializer`.

Elements

[Table D-7](#) describes the elements you can define within the `user-type` element.

Table D-7 *user-type Subelements*

Element	Required/ Optional	Description
<class-name>	Required	The <code>class-name</code> element specifies the name of a Java class or interface. Within the <code>user-type</code> element, the <code>class-name</code> element is required, and specifies the fully qualified name of the Java class or interface that all values of the user type are type-assignable to. Within the <code>serializer</code> element, the <code>class-name</code> element is required.
<serializer>	Optional	<p>The <code>serializer</code> element specifies what <code>PofSerializer</code> to use to serialize and deserialize a specific user type. A <code>PofSerializer</code> is used to serialize and deserialize user type values to and from a POF stream. Within the <code>serializer</code> element, the <code>class-name</code> element is required, and zero or more constructor parameters can be defined within an <code>init-params</code> element.</p> <p>If the <code>serializer</code> element is omitted, then the user type is assumed to implement the <code>PortableObject</code> interface, and the <code>PortableObjectSerializer</code> implementation is used as the <code>PofSerializer</code>.</p> <p>If the <code>init-params</code> element is omitted from the <code>serializer</code> element, then the following four constructors are attempted on the specific <code>PofSerializer</code> implementation, and in this order:</p> <ul style="list-style-type: none"> ■ <code>(int nTypeId, Class clz, ClassLoader loader)</code> ■ <code>(int nTypeId, Class clz)</code> ■ <code>(int nTypeId)</code> ■ <code>()</code>
<type-id>	Optional	The <code>type-id</code> element specifies an integer value ($n \geq 0$) that uniquely identifies the user type. If none of the <code>user-type</code> elements contains a <code>type-id</code> element, then the type IDs for the user types will be based on the order in which they appear in the <code>user-type-list</code> , with the first user type being assigned the type ID 0, the second user type being assigned the type ID 1, and so on. However, it is strongly recommended that user types IDs always be specified, to support schema versioning and evolution.

user-type-list

Used in: <pof-config>

Description

The user-type-list element contains zero or more user-type elements. Each POF user type that will be used must be listed in the user-type-list.

The user-type-list element may also contain zero or more include elements. Each include element is used to add user-type elements defined in another pof-config file.

Elements

The following table describes the elements you can define within the user-type-list element.

Table D–8 *user-type-list Subelements*

Element	Required/ Optional	Description
<include>	Optional	The include element specifies the location of a pof-config file to load user-type elements from. The value is a locator string (either a valid path or URL) that identifies the location of the target pof-config file. Any number of <include> elements may be specified.
<user-type>	Optional	<p>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.</p> <p>Within the user-type element, the type-id element is optional, but its use is strongly suggested to support schema versioning and evolution.</p> <p>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.</p> <p>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.</p>

Coherence MBeans Reference

This appendix provides a reference of the Coherence MBeans that are used to manage and monitor different parts of Coherence. All of the MBeans' attributes and operations are described in detail.

In addition to this reference, Coherence MBeans are described in the `Registration` interface, see *Oracle Coherence Java API Reference*. An MBean-capable agent (such as JConsole) can also be used to view MBean information. See [Part VI, "Managing Coherence"](#) for more information on using the management features of Coherence.

The following sections are included in this Appendix:

- [Overview](#)
- [MBean Index](#)
- [CacheMBean](#)
- [ClusterMBean](#)
- [ClusterNodeMBean](#)
- [ConnectionManagerMBean](#)
- [ConnectionMBean](#)
- [ManagementMBean](#)
- [PointToPointMBean](#)
- [ReporterMBean](#)
- [ServiceMBean](#)
- [StorageManagerMBean](#)
- [TransactionManagerMBean](#)

Overview

The Coherence cluster management gateway uses a `Registration` interface that is specific to managing Coherence clustered resources and is an abstraction of the basic JMX registration APIs. Though the interface is closely related to the JMX infrastructure, it is independent from `javax.management.*` classes. This enables remote management support for cluster nodes that are not co-located with any JMX services. In addition, the Coherence MBeans are registered in an MBean server that could either be co-located or remote in relation to the managed object.

MBean Index

Table E–1 lists the MBeans that are provided for managing Coherence.

Table E–1 *Coherence MBeans*

MBean	Description
CacheMBean	Represents a cache. A cluster node may have zero or more instances of this managed bean.
ClusterMBean	Represents a cluster object. Each cluster node has a single instance of this managed bean.
ClusterNodeMBean	Represents a cluster member. Each cluster node has a single instance of this managed bean.
ConnectionManagerMBean	Represents a Coherence*Extend proxy. A cluster node may have zero or more instances of this managed bean.
ConnectionMBean	Represents a remote client connection through Coherence*Extend. A cluster node may have zero or more instances of this managed bean.
ManagementMBean	Represents the grid JMX infrastructure. Each cluster node has a single instance of this managed bean.
PointToPointMBean	Represents the network status between two cluster members. Each cluster node has a single instance of this managed bean.
ReporterMBean	Represents the Coherence JMX Reporter. Each cluster node has a single instance of this managed bean.
ServiceMBean	Represents a clustered Service. A cluster node may have zero or more instances of this managed bean.
StorageManagerMBean	Represents a storage instance for a storage-enabled distributed cache service. A cluster node may have zero or more instances of this managed bean.
TransactionManagerMBean	Represents a transaction manager. A cluster node may have zero or more instances of this managed bean.

CacheMBean

The CacheMBean MBean represents a cache. A cluster node may have zero or more instances of this managed bean depending on the number of caches that are configured for each data service type (distributed, replicated, and so on). This MBean provides operational and performance statistics for a cache. Some of the MBean's attributes are writable and allow the behavior of a cache to be changed in real time.

The MBean's object name is:

```
type=Cache,service=service name,name=cache name,nodeId=cluster node's id,tier=tier tag
```

Note: The CacheMBean MBean contains many attributes that are not applicable to transactional caches. A transactional cache returns a -1 value if the non-applicable attributes are invoked. See ["CacheMBeans for Transactional Caches"](#) on page 25-19 for a list of the supported attributes and transaction-specific descriptions.

Attributes

Table E-2 describes the attributes for CacheMBean.

Table E-2 CacheMBean Attributes

Attribute	Type	Access	Description
AverageGetMillis	Double	RO	The average number of milliseconds per <code>get()</code> invocation since the last time statistics were reset.
AverageHitMillis	Double	RO	The average number of milliseconds per <code>get()</code> invocation that is a hit.
AverageMissMillis	Double	RO	The average number of milliseconds per <code>get()</code> invocation that is a miss.
AveragePutMillis	Double	RO	The average number of milliseconds per <code>put()</code> invocation since the cache statistics were last reset.
BatchFactor	Double	RW	The BatchFactor attribute 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 QueueDelay interval. The soft-ripe time is the point in time prior to the actual ripe time after which an entry is included in a batch asynchronous write operation to the cache store (along with all other ripe and soft-ripe entries). This attribute is only applicable if asynchronous writes are enabled (that is, the value of the QueueDelay attribute is greater than zero) and the cache store implements the <code>storeAll()</code> method. The value of the element is expressed as a percentage of the QueueDelay interval. Valid values are doubles in the interval [0.0, 1.0].
CacheHits	Long	RO	The rough number of cache hits since the last time statistics were reset. A cache hit is a read operation invocation (that is, <code>get()</code>) for which an entry exists in this map.

Table E–2 (Cont.) CacheMBean Attributes

Attribute	Type	Access	Description
CacheHitsMillis	Long	RO	The total number of milliseconds (since the last time statistics were reset) for the <code>get()</code> operations for which an entry existed in this map.
CacheMisses	Long	RO	The rough number of cache misses since the last time statistics were reset.
CacheMissesMillis	Long	RO	The total number of milliseconds (since the last time statistics were reset) for the <code>get()</code> operations for which no entry existed in this map.
CachePrunes	Long	RO	The number of prune operations since the last time statistics were reset. A prune operation occurs every time the cache reaches its high watermark as specified by the <code>HighUnits</code> attribute.
CachePrunesMillis	Long	RO	The total number of milliseconds for the prune operations since the last time statistics were reset.
Description	String	RO	The cache description.
ExpiryDelay	Integer	RW	The time-to-live for cache entries in milliseconds. Value of zero indicates that the automatic expiry is disabled. Change of this attribute will not affect already-scheduled expiry of existing entries.
HighUnits	Integer	RW	The limit of the cache size measured in units. The cache will prune itself automatically once it reaches its maximum unit level. This is often referred to as the high water mark of the cache.
HitProbability	Double	RO	The rough probability ($0 \leq p \leq 1$) that the next invocation will be a hit, based on the statistics collected since the last time statistics were reset.
LowUnits	Integer	RW	The number of units to which the cache will shrink when it prunes. This is often referred to as a low water mark of the cache.
PersistenceType	String	RO	The persistence type for this cache. Possible values include: <code>NONE</code> , <code>READ-ONLY</code> , <code>WRITE-THROUGH</code> , <code>WRITE-BEHIND</code> .
QueueDelay	Integer	RW	The number of seconds that an entry added to a write-behind queue sits in the queue before being stored using a cache store. This attribute is only applicable if the persistence type is <code>WRITE-BEHIND</code> .
QueueSize	Integer	RO	The size of the write-behind queue size. This attribute is only applicable if the persistence type is <code>WRITE-BEHIND</code> .
RefreshFactor	Double	RW	This attribute is used to calculate the soft-expiration time for cache entries. Soft-expiration is the point in time prior to the actual expiration after which any access request for an entry will schedule an asynchronous load request for the entry. This attribute is only applicable for a read write backing map which has an internal local cache with scheduled automatic expiration. The value of this element is expressed as a percentage of the internal local cache expiration interval. Valid values are doubles in the interval <code>[0.0, 1.0]</code> . If zero, refresh-ahead scheduling is disabled.
RefreshTime	Date	RO	The timestamp when this model was last retrieved from a corresponding node. For local servers it is the local time.

Table E–2 (Cont.) CacheMBean Attributes

Attribute	Type	Access	Description
RequeueThreshold	Integer	RW	The maximum size of the write-behind queue for which failed cache store write operations are requeued. If zero, the write-behind requeueing will be disabled. This attribute is only applicable if the persistence type is <code>WRITE-BEHIND</code> .
Size	Integer	RO	The number of entries in the cache.
StoreAverageBatchSize	Long	RO	The average number of entries stored for each cache store write operation. A call to the <code>store()</code> method is counted as a batch of one, whereas a call to the <code>storeAll()</code> method is counted as a batch of the passed <code>Map</code> size. The value is <code>-1</code> if the persistence type is <code>NONE</code> .
StoreAverageReadMillis	Long	RO	The average time (in milliseconds) spent per read operation. The value is <code>-1</code> if the persistence type is <code>NONE</code> .
StoreAverageWriteMillis	Long	RO	The average time (in milliseconds) spent per write operation. The value is <code>-1</code> if the persistence type is <code>NONE</code> .
StoreFailures	Long	RO	The total number of cache store failures (load, store and erase operations). The value is <code>-1</code> if the persistence type is <code>NONE</code> .
StoreReadMillis	Long	RO	The cumulative time (in milliseconds) spent on load operations. The value is <code>-1</code> if the persistence type is <code>NONE</code> .
StoreReads	Long	RO	The total number of load operations. The value is <code>-1</code> if the persistence type is <code>NONE</code> .
StoreWriteMillis	Long	RO	The cumulative time (in milliseconds) spent on store and erase operations. The value is <code>-1</code> if the persistence type is <code>NONE</code> or <code>READ-ONLY</code> .
StoreWrites	Long	RO	The total number of store and erase operations. The value is <code>-1</code> if the persistence type is <code>NONE</code> or <code>READ-ONLY</code> .
TotalGets	Long	RO	The total number of <code>get()</code> operations since the last time statistics were reset.
TotalGetsMillis	Long	RO	The total number of milliseconds spent on <code>get()</code> operations since the last time statistics were reset.
TotalPuts	Long	RO	The total number of <code>put()</code> operations since the last time statistics were reset.
TotalPutsMillis	Long	RO	The total number of milliseconds spent on <code>put()</code> operations since the last time statistics were reset.
UnitFactor	Integer	RO	The factor by which the <code>Units</code> , <code>LowUnits</code> and <code>HighUnits</code> properties are adjusted. Using a <code>BINARY</code> unit calculator, for example, the factor of <code>1048576</code> could be used to count megabytes instead of bytes.
Units	Integer	RO	The size of the cache measured in units. This value needs to be adjusted by the <code>UnitFactor</code> .

Operations

The `CacheMBean` `MBean` includes a `resetStatistics` operation that resets all cache statistics.

ClusterMBean

The `ClusterMBean` MBean represents a cluster. Each cluster node has a single instance of this managed bean. This MBean provides operational statistics about the cluster.

The MBean's object name is:

`type=Cluster`

Attributes

[Table E-3](#) describes the attributes for `ClusterMBean`.

Table E-3 *ClusterMBean Attributes*

Attribute	Type	Access	Description
<code>ClusterName</code>	<code>String</code>	RO	The name of the cluster.
<code>ClusterSize</code>	<code>Integer</code>	RO	The total number of cluster nodes.
<code>LicenseMode</code>	<code>String</code>	RO	The license mode that this cluster is using. Possible values are <code>Evaluation</code> , <code>Development</code> or <code>Production</code> .
<code>LocalMemberId</code>	<code>Integer</code>	RO	The member id for the cluster member that is co-located with the reporting MBean server. The value is <code>-1</code> if the cluster service is not running.
<code>MemberIds</code>	<code>Integer[]</code>	RO	An array of all existing cluster member ids.
<code>Members</code>	<code>String[]</code>	RO	An array of all existing cluster members.
<code>MembersDeparted</code>	<code>String[]</code>	RO	An array of strings containing the member information for recently departed cluster members. Members are removed from this array when the member id is recycled. This information is since the node has joined the cluster and is reset when the MBean server node leaves and rejoins the cluster. The <code>MembersDepartureCount</code> is the total count of departed members and not the size of this array.
<code>MembersDepartureCount</code>	<code>Long</code>	RO	The number of times this node has observed another node's departure from the cluster since this management node has joined the cluster or statistics have been reset.
<code>OldestMemberId</code>	<code>Integer</code>	RO	The senior cluster member id. The value is <code>-1</code> if the cluster service is not running.
<code>RefreshTime</code>	<code>Date</code>	RO	The timestamp when this model was last retrieved from a corresponding node. For local servers it is the local time.
<code>Running</code>	<code>Boolean</code>	RO	Specifies whether or not the cluster is running.
<code>Version</code>	<code>String</code>	RO	The Coherence version.

Operations

[Table E-4](#) describes the operations for `ClusterMBean`.

Table E-4 *ClusterMBean Operations*

Operation	Parameters	Return Type	Description
ensureRunning	Void	Void	Ensures that the cluster service is running on this node.
shutdown	Void	Void	Shuts down the cluster service on this node.

ClusterNodeMBean

The `ClusterNodeMBean` MBean represents a cluster member. Each cluster node has a single instance of this managed bean. This MBean provides many operational and performance statistics for a node/member of a cluster. Many of the attributes are writable and allow the behavior of the node to be changed in real time.

The MBean's object name is:

```
type=Node,nodeId=cluster node's id
```

Attributes

Table E-5 describes the attributes for `ClusterNodeMBean`.

Table E-5 *ClusterNodeMBean Attributes*

Attribute	Type	Access	Description
BufferPublishSize	Integer	RW	The buffer size of the unicast datagram socket used by the Publisher, measured in the number of packets. Changing this value at runtime is an inherently unsafe operation that will pause all network communications and may result in the termination of all cluster services.
BufferReceiveSize	Integer	RW	The buffer size of the unicast datagram socket used by the Receiver, measured in the number of packets. Changing this value at runtime is an inherently unsafe operation that will pause all network communications and may result in the termination of all cluster services.
CpuCount	Integer	RO	Number of CPU cores for the machine this member is running on.
FlowControlEnabled	Boolean	RO	Indicates whether or not <code>FlowControl</code> is enabled. See "flow-control" on page A-18.
Id	Integer	RO	The short member id that uniquely identifies the member at this point in time and does not change for the life of this member.
LoggingDestination	String	RO	The output device used by the logging system. Valid values are <code>stdout</code> , <code>stderr</code> , <code>jdk</code> , <code>log4j</code> , or a file name.
LoggingFormat	String	RW	Specifies how messages will be formatted before being passed to the log destination.
LoggingLevel	Integer	RW	Specifies which logged messages will be output to the log destination. Valid values are non-negative integers. A value of -1 to disable all logger output.
LoggingLimit	Integer	RW	The maximum number of characters that the logger daemon will process from the message queue before discarding all remaining messages in the queue. Valid values are positive integers in the range 0 to <code>Integer.MAX_VALUE</code> (2147483647). Zero implies <code>Integer.MAX_VALUE</code> .
MachineId	Integer	RO	The member's machine Id.
MachineName	String	RO	A configured name that should be the same for all members that are on the same physical machine, and different for members that are on different physical machines.
MemberName	String	RO	A configured name that must be unique for every member.

Table E-5 (Cont.) ClusterNodeMBean Attributes

Attribute	Type	Access	Description
MemoryAvailableMB	Integer	RO	The total amount of memory in the JVM available for new objects in MB.
MemoryMaxMB	Integer	RO	The maximum amount of memory that the JVM will attempt to use in MB.
MulticastAddress	String	RO	The IP address of the member's multicast socket for group communication.
MulticastEnabled	Boolean	RO	Specifies whether or not this member uses multicast for group communication. If <code>false</code> , this member will use the addresses listed in the <code>WellKnownAddresses</code> attribute to join the cluster and point-to-point unicast to communicate with other members of the cluster.
MulticastPort	Integer	RO	The port of the member's multicast socket for group communication.
MulticastTTL	Integer	RO	The time-to-live for multicast packets sent out on this member's multicast socket.
MulticastThreshold	Integer	RW	The percentage (0 to 100) of the servers in the cluster that a packet will be sent to, above which the packet is sent using multicast and below which it is sent using unicast.
NackEnabled	Boolean	RO	Indicates whether or not the early packet loss detection protocol is enabled.
NackSent	Long	RO	The total number of NACK packets sent since the node statistics were last reset.
PacketDeliveryEfficiency	Float	RO	The efficiency of packet loss detection and retransmission. A low efficiency is an indication that there is a high rate of unnecessary packet retransmissions.
PacketsBundled	Long	RO	The total number of packets which were bundled prior to transmission. The total number of network transmissions is equal to $PacketsSent - PacketsBundled$.
PacketsReceived	Long	RO	The number of packets received since the node statistics were last reset.
PacketsRepeated	Long	RO	The number of duplicate packets received since the node statistics were last reset.
PacketsResent	Long	RO	The number of packets resent since the node statistics were last reset. A packet is resent when there is no ACK received within a timeout period.
PacketsResentEarly	Long	RO	The total number of packets resent ahead of schedule. A packet is resent ahead of schedule when there is a NACK indicating that the packet has not been received.
PacketsResentExcess	Long	RO	The total number of packet retransmissions which were later proven unnecessary.
PacketsSent	Long	RO	The number of packets sent since the node statistics were last reset.
Priority	Integer	RO	The priority, or weight, of the member; used to determine tie-breakers.
ProcessName	String	RO	A configured name that should be the same for members that are in the same process (JVM), and different for members that are in different processes. If not explicitly provided, the name will be calculated internally as the <code>Name</code> attribute of the system <code>RuntimeMXBean</code> , which normally represents the process identifier (PID).

Table E-5 (Cont.) ClusterNodeMBean Attributes

Attribute	Type	Access	Description
ProductEdition	String	RO	The product edition this member is running. Possible values are: Standard Edition, Enterprise Edition, Grid Edition.
PublisherPacketUtilization	Float	RO	The publisher packet utilization for this cluster node since the node socket was last reopened. This value is a ratio of the number of bytes sent to the number that would have been sent had all packets been full. A low utilization indicates that data is not being sent in large enough chunks to make efficient use of the network.
PublisherSuccessRate	Float	RO	The publisher success rate for this cluster node since the node statistics were last reset. The publisher success rate is a ratio of the number of packets successfully delivered in a first attempt to the total number of sent packets. A failure count is incremented when there is no ACK received within a timeout period. It could be caused by either very high network latency or a high packet drop rate.
QuorumStatus	String	RO	The current state of the cluster quorum.
RackName	String	RO	A configured name that should be the same for members that are on the same physical "rack" (or frame or cage), and different for members that are on different physical "racks".
ReceiverPacketUtilization	Float	RO	The receiver packet utilization for this cluster node since the socket was last reopened. This value is a ratio of the number of bytes received to the number that would have been received had all packets been full. A low utilization indicates that data is not being sent in large enough chunks to make efficient use of the network.
ReceiverSuccessRate	Float	RO	The receiver success rate for this cluster node since the node statistics were last reset. The receiver success rate is a ratio of the number of packets successfully acknowledged in a first attempt to the total number of received packets. A failure count is incremented when a re-delivery of previously received packet is detected. It could be caused by either very high inbound network latency or lost ACK packets.
RefreshTime	Date	RO	The timestamp when this model was last retrieved from a corresponding node. For local servers it is the local time.
ResendDelay	Integer	RW	The minimum number of milliseconds that a packet will remain queued in the Publisher's re-send queue before it is resent to the recipient(s) if the packet has not been acknowledged. Setting this value too low can overflow the network with unnecessary repetitions. Setting the value too high can increase the overall latency by delaying the re-sends of dropped packets. Additionally, change of this value may need to be accompanied by a change in SendAckDelay value.
RoleName	String	RO	A configured name that can be used to indicate the role of a member to the application. While managed by Coherence, this property is only used by the application.
SendAckDelay	Integer	RW	The minimum number of milliseconds between the queueing and sending of an ACK packet. This value should be not more than a half of the ResendDelay value.

Table E–5 (Cont.) ClusterNodeMBean Attributes

Attribute	Type	Access	Description
SendQueueSize	Integer	RO	The number of packets currently scheduled for delivery. This number includes both packets that are to be sent immediately and packets that have already been sent and awaiting for acknowledgment. Packets that do not receive an acknowledgment within the <code>ResendDelay</code> interval will be automatically resent.
SiteName	String	RO	A configured name that should be the same for members that are on the same physical site (for example, data center), and different for members that are on different physical sites.
SocketCount	Integer	RO	Number of CPU sockets for the machine this member is running on.
Statistics	String	RO	Statistics for this cluster node in a human readable format.
TcpRingFailures	Long	RO	The number of recovered TcpRing disconnects since the node statistics were last reset. A recoverable disconnect is an abnormal event that is registered when the TcpRing peer drops the TCP connection, but recovers after no more than the maximum configured number of attempts. A -1 value indicates that TcpRing is disabled.
Timestamp	Date	RO	The date/time value (in cluster time) that this member joined the cluster.
TrafficJamCount	Integer	RW	The maximum total number of packets in the send and resend queues that forces the publisher to pause client threads. Zero means no limit.
TrafficJamDelay	Integer	RW	The number of milliseconds to pause client threads when a traffic jam condition has been reached. Anything less than one (for example, zero) is treated as one millisecond.
UnicastAddress	String	RO	The IP address of the member's datagram socket for point-to-point communication.
UnicastPort	Integer	RO	The port of the member's datagram socket for point-to-point communication.
WeakestChannel	Integer	RO	The id of the cluster node to which this node is having the most difficulty communicating, or -1 if none is found. A channel is considered to be weak if either the point-to-point publisher or receiver success rates are below 1.0.
WellKnownAddresses	String[]	RO	An array of well-known socket addresses that this member uses to join the cluster.

Operations

[Table E–6](#) describes the operations for ClusterNodeMBean.

Table E–6 *ClusterNodeMBean Operations*

Operation	Parameters	Return Type	Description
ensureCacheService	String sCacheName	void	Ensure that a cache service for the specified cache runs at the cluster node represented by this MBean. This method will use the configurable cache factory to find out which cache service to start if necessary. Return value indicates the service name. A null value indicates a match could not be found.
ensureInvocationService	String sServiceName	void	Ensure that an invocation service with the specified name runs at the cluster node represented by this MBean.
resetStatistics	void	void	Reset the cluster node statistics.
shutdown	void	void	Stop all the clustered services running at this node (controlled shutdown). The management of this node will not be available until the node is restarted (manually or programmatically).

ConnectionManagerMBean

The `ConnectionManagerMBean` MBean represents a Coherence*Extend proxy. A cluster node may have zero or more instances of this managed bean depending on the number of configured proxies. The MBean contains statistics for throughput and connection information for proxy hosts.

The MBean's object name is:

`type=ConnectionManager,name=service name,nodeId=cluster node's id`

Attributes

[Table E-7](#) describes the attributes for `ConnectionManagerMBean`.

Table E-7 *ConnectionManagerMBean Attributes*

Attribute	Type	Access	Description
<code>ConnectionCount</code>	Integer	RO	The number of client connections.
<code>HostIP</code>	String	RO	The IP address and port of the proxy host.
<code>IncomingBufferPoolCapacity</code>	Long	RO	The pool capacity (in bytes) of the incoming buffer.
<code>IncomingBufferPoolSize</code>	Integer	RO	The number of buffers in the incoming pool.
<code>OutgoingBufferPoolCapacity</code>	Long	RO	The pool capacity (in bytes) of the outgoing buffer.
<code>OutgoingBufferPoolSize</code>	Integer	RO	The number of buffers in the outgoing pool.
<code>OutgoingByteBacklog</code>	Long	RO	The backlog (in bytes) of the outgoing queue
<code>OutgoingMessageBacklog</code>	Long	RO	The backlog of the outgoing message queue.
<code>RefreshTime</code>	Date	RO	The timestamp when this model was last retrieved from a corresponding node. For local servers it is the local time.
<code>TotalBytesReceived</code>	Long	RO	The total number of bytes received by the proxy host since the statistics were last reset.
<code>TotalBytesSent</code>	Long	RO	The total number of bytes sent by the proxy host since the statistics were last reset.
<code>TotalMessagesReceived</code>	Long	RO	The total number of messages received by the proxy host since the statistics were last reset.
<code>TotalMessagesSent</code>	Long	RO	The total number of messages sent by the proxy host since the statistics were last reset.

Operations

The `ConnectionManagerMBean` MBean has no operations.

ConnectionMBean

The `ConnectionMBean` MBean represents a remote client connection through Coherence*Extend. A cluster node may have zero or more instances of this managed bean depending on the number of active remote connections to the cluster. The MBean contains performance and usage statistics for the connection.

The MBean's object name is:

`type=Connection,name=service name ,nodeId=cluster node's id,UUID=connection's id`

Attributes

Table E–8 describes the attributes for `ConnectionMBean`.

Table E–8 *ConnectionMBean Attributes*

Attribute	Type	Access	Description
<code>ConnectionTimeMillis</code>	Long	RO	The time duration (in milliseconds) that the client has been connected.
<code>OutgoingByteBacklog</code>	Long	RO	The backlog (in bytes) of the outgoing queue
<code>OutgoingMessageBacklog</code>	Integer	RO	The backlog of the outgoing message queue.
<code>RefreshTime</code>	Date	RO	The timestamp when this model was last retrieved from a corresponding node. For local servers it is the local time.
<code>RemoteAddress</code>	String	RO	The IP address of the corresponding client.
<code>RemotePort</code>	Integer	RO	The port of the corresponding client.
<code>Timestamp</code>	Date	RO	The date/time value (in local time) that the corresponding client connected to the proxy.
<code>TotalBytesReceived</code>	Long	RO	The total number of bytes received since the last time the statistics were reset.
<code>TotalBytesSent</code>	Long	RO	The total number of bytes sent since the last time the statistics were reset.
<code>TotalMessagesReceived</code>	Long	RO	The total number of messages received since the last time the statistics were reset.
<code>TotalMessagesSent</code>	Long	RO	The total number of messages sent since the last time the statistics were reset.
<code>UUID</code>	String	RO	The unique identifier for this connection.

Operations

Table E–9 describes the operations for `ConnectionMBean`.

Table E–9 *ConnectionMBean Operations*

Operation	Parameters	Return Type	Description
<code>closeConnection</code>	void	void	Close the corresponding connection.
<code>resetStatistics</code>	void	void	Reset the connection statistics.

ManagementMBean

The ManagementMBean MBean represents the grid JMX infrastructure. Each cluster node has a single instance of this managed bean. The MBean contains management settings. Some of the attributes are writable and allow management behavior to be changed in real time.

The MBean's object name is:

type=Management

Attributes

Table E-10 describes the attributes for ManagementMBean.

Table E-10 ManagementMBean Attributes

Attribute	Type	Access	Description
ExpiryDelay	Long	RW	The number of milliseconds that the MBean server will keep a remote model snapshot before refreshing.
RefreshCount	Long	RO	The total number of snapshots retrieved since the statistics were last reset.
RefreshExcessCount	Long	RO	The number of times the MBean server predictively refreshed information and the information was not accessed.
RefreshOnQuery	Boolean	RO	Specifies whether or not the refresh-on-query MBean server is configured. If this is <code>true</code> then the RefreshPolicy value should be <code>refresh-onquery</code> .
RefreshPolicy	String	RW	The policy used to determine the behavior when refreshing remote models. Valid values are: <code>refresh-ahead</code> , <code>refresh-behind</code> , <code>refresh-expired</code> , <code>refresh-onquery</code> . Invalid values will convert to <code>refresh-expired</code> .
RefreshPredictionCount	Long	RO	The number of times the MBean server used a predictive (<code>refresh-behind</code> , <code>refresh-ahead</code> , <code>refresh-onquery</code>) algorithm to refresh MBean information.
RefreshTime	Date	RO	The timestamp when this model was last retrieved from a corresponding node. For local servers it is the local time.
RefreshTimeoutCount	Long	RO	The number of times this management node has timed out while attempting to refresh remote MBean attributes.
RemoteNotificationCount	Long	RO	The total number of remote notifications received for all MBeans by this node since the last time the statistics were reset.

Operations

The ManagementMBean MBean includes a `resetStatistics` operation that resets the `RefreshCount`, `RefreshExcessCount` and `RefreshPredictionCount` statistics.

PointToPointMBean

The `PointToPointMBean` MBean represents the network status between two cluster members. Each cluster node has a single instance of this managed bean. The MBean provides network statistics from the perspective of the current viewing member with respect to a specified viewed member. To specify the member, enter its ID using the `ViewedMemberId` attribute.

The MBean's object name is:

```
type=PointToPoint,nodeId=cluster node's id
```

Attributes

[Table E-11](#) describes the attributes for `PointToPointMBean`.

Table E-11 *PointToPointMBean Attributes*

Attribute	Type	Access	Description
<code>DeferredPackets</code>	Integer	RO	The number of packets addressed to the viewed member that the viewing member is currently deferring to send. The viewing member will delay sending these packets until the number of outstanding packets falls below the value of the <code>Threshold</code> attribute. The value of this attribute is only meaningful if the viewing member has <code>FlowControl</code> enabled. See "flow-control" on page A-18.
<code>Deferring</code>	Boolean	RO	Indicates whether or not the viewing member is currently deferring packets to the viewed member. The value of this attribute is only meaningful if the viewing member has <code>FlowControl</code> enabled. See "flow-control" on page A-18.
<code>LastIn</code>	Long	RO	The number of milliseconds that have elapsed since the viewing member last received an acknowledgment from the viewed member.
<code>LastOut</code>	Long	RO	The number of milliseconds that have elapsed since the viewing member last sent a packet to the viewed member.
<code>LastSlow</code>	Long	RO	The number of milliseconds that have elapsed since the viewing member declared the viewed member as slow, or -1 if the viewed member has never been declared slow.
<code>OutstandingPackets</code>	Integer	RO	The number of packets that the viewing member has sent to the viewed member which have yet to be acknowledged. The value of this attribute is only meaningful if the viewing member has <code>FlowControl</code> enabled. See "flow-control" on page A-18.
<code>PauseRate</code>	Float	RO	The percentage of time since the last time statistics were reset in which the viewing member considered the viewed member to be unresponsive. Under normal conditions this value should be very close to 0.0. Values near 1.0 would indicate that the viewed node is nearly inoperable, likely due to extremely long GC pauses. The value of this attribute is only meaningful if the viewing member has <code>FlowControl</code> enabled. See "flow-control" on page A-18.
<code>Paused</code>	Boolean	RO	Indicates whether or not the viewing member currently considers the viewed member to be unresponsive. The value of this attribute is only meaningful if the viewing member has <code>FlowControl</code> enabled. See "flow-control" on page A-18.

Table E–11 (Cont.) PointToPointMBean Attributes

Attribute	Type	Access	Description
PublisherSuccessRate	Float	RO	The publisher success rate from the viewing node to the viewed node since the statistics were last reset.
ReceiverSuccessRate	Float	RO	The receiver success rate from the viewing node to the viewed node since the statistics were last reset.
RefreshTime	Date	RO	The timestamp when this model was last retrieved from a corresponding node. For local servers it is the local time.
Threshold	Integer	RO	The maximum number of outstanding packets for the viewed member that the viewing member is allowed to accumulate before initiating the deferral algorithm. The value of this attribute is only meaningful if the viewing member has FlowControl enabled. See "flow-control" on page A-18.
ViewedMemberId	Integer	RW	The Id of the member being viewed.
ViewerStatistics	String[]	RO	Human readable summary of the point-to-point statistics from the viewing member for all other members.

Operations

[Table E–12](#) describes the operations for PointToPointMBean.

Table E–12 PointToPointMBean Operations

Operation	Parameters	Return Type	Description
resetStatistics	void	void	Reset the viewing member's point-to-point statistics for all other members.
trackWeakest	void	void	Instruct the Point-to-Point MBean to track the weakest member. A viewed member is considered to be weak if either the corresponding publisher or receiver success rates are below 1.0.

ReporterMBean

The `ReporterMBean` MBean represents the Coherence JMX Reporter. Each cluster node has a single instance of this managed bean. The MBean contains settings and statistics for the reporter. Many of the attributes are writable and allow the reporter configuration to be changed in real time. In addition, the MBean contains operations that start and stop the reporter as well as run reports in real time.

The MBean's object name is:

`type=Reporter`

Attributes

[Table E-13](#) describes the attributes for `ReporterMBean`.

Table E-13 *ReporterMBean Attributes*

Attribute	Type	Access	Description
<code>AutoStart</code>	Boolean	RO	Specifies whether the reporter starts automatically with the node.
<code>ConfigFile</code>	String	RW	The configuration file for the Reporter.
<code>CurrentBatch</code>	Long	RW	The batch identifier for the Reporter.
<code>IntervalSeconds</code>	Long	RW	The interval between executions in seconds.
<code>LastExectionTime</code>	Date	RO	The last time a report batch was executed. For local servers it is the local time.
<code>LastReport</code>	String	RO	The last report to execute.
<code>OutputPath</code>	String	RW	The path where report output will be located.
<code>RefreshTime</code>	Date	RO	The last time that the reporter statistics were reset. For local servers it is the local time.
<code>Reports</code>	String[]	RO	The list of reports executed.
<code>RunAverageMillis</code>	Double	RO	The average batch runtime in milliseconds since the statistics were last reset.
<code>RunLastMillis</code>	Long	RO	The last batch runtime in milliseconds since the statistics were last reset.
<code>RunMaxMillis</code>	Long	RO	The maximum batch runtime in milliseconds since the statistics were last reset.
<code>State</code>	String	RO	The state of the Reporter. Valid values are: <code>Running</code> (reports are being executed); <code>Waiting</code> (the reporter is waiting for the interval to complete); <code>Starting</code> (the reporter is being started); <code>Stopping</code> (the reporter is attempting to stop execution and waiting for running reports to complete); <code>Stopped</code> (the reporter is stopped); <code>Sleeping</code> (the reporter is sleeping).

Operations

[Table E-14](#) describes the operations for `ReporterMBean`.

Table E-14 *ReporterMBean Operations*

Operation	Parameters	Return Type	Description
Stop	void	void	Stop the reporter.
Start	void	void	Start the reporter.
RunReport	String sReportFile	void	Run the specified report configuration file (for example <code>reports/report-group.xml</code>) one time.
resetStatistics	void	void	Reset the reporter statistics.

ServiceMBean

The `ServiceMBean` MBean represents a clustered Service. A cluster node may have zero or more instances of this managed bean depending on the number of clustered services that are started. The MBean contains usage and performance statistics for a service. Some of the attributes are writable and allow the behavior of a service to be changed in real time. In addition, the MBean contains operations that are used to start and stop a service in real time.

The MBean's object name is:

```
type=Service,name=service name,nodeId=cluster node's id
```

Terminology

The terms *task* and *request* have unique definitions within Coherence. These definitions should be understood before setting the task-related and request-related attributes for `ServiceMBean`.

- **Task** — A task is an invoked object that executes on one or more nodes. The objects include filters, invocation agents (entry processors and aggregators), or single-pass agents (Invocable objects).
- **Request** — A request is the round trip required to complete a task. A request begins the moment a task is sent for execution by a client 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.

Attributes

[Table E-15](#) describes the attributes for `ServiceMBean`.

Table E-15 *ServiceMBean Attributes*

Attribute	Type	Access	Description
BackupCount	Integer	RO	The number of backups for every cache storage.
BackupCountAfterWritebehind	Integer	RO	The number of members of the partitioned (distributed) cache service that will retain backup data that does not require write-behind. That is, data that is not vulnerable to being lost even if the entire cluster were shut down.
OwnedPartitionsBackup	Integer	RO	The number of partitions that this member backs up (responsible for the backup storage).
OwnedPartitionsPrimary	Integer	RO	The number of partitions that this member owns (responsible for the primary storage).
PartitionsAll	Integer	RO	The total number of partitions that every cache storage will be divided into.
PartitionsEndangered	Integer	RO	The total number of partitions that are not currently backed up.

Table E–15 (Cont.) ServiceMBean Attributes

Attribute	Type	Access	Description
PartitionsUnbalanced	Integer	RO	The total number of primary and backup partitions which remain to be transferred until the partition distribution across the storage enabled service members is fully balanced.
PartitionsVulnerable	Integer	RO	The total number of partitions that are backed up on the same machine where the primary partition owner resides.
QuorumStatus	String	RO	The current state of the service quorum.
RefreshTime	Date	RO	The timestamp when this model was last retrieved from a corresponding node. For local servers it is the local time.
RequestAverageDuration	Float	RO	The average duration (in milliseconds) of an individual synchronous request issued by the service since the last time the statistics were reset.
RequestMaxDuration	Long	RO	The maximum duration (in milliseconds) of a synchronous request issued by the service since the last time the statistics were reset.
RequestPendingCount	Long	RO	The number of pending synchronous requests issued by the service.
RequestPendingDuration	Long	RO	The duration (in milliseconds) of the oldest pending synchronous request issued by the service.
RequestTimeoutCount	Long	RO	The total number of timed-out requests since the last time the statistics were reset.
RequestTimeoutMillis	Long	RW	The default timeout value in milliseconds for requests that can be timed-out (for example, implement the <code>com.tangosol.net.PriorityTask</code> interface) but do not explicitly specify the request timeout value.
RequestTotalCount	Long	RO	The total number of synchronous requests issued by the service since the last time the statistics were reset.
Running	Boolean	RO	Specifies whether or not the service is running.
SeniorMemberId	Integer	RO	The service senior member id. The value is -1 if the service is not running.
Statistics	String	RO	Statistics for this service in human readable format.
StatusHA	String	RO	The High Availability (HA) status for this service. The value of <code>MACHINE-SAFE</code> means that all the cluster nodes running on any given machine could be stopped at once without data loss. The value of <code>NODE-SAFE</code> means that any cluster node could be stopped without data loss. The value of <code>ENDANGERED</code> indicates that abnormal termination of any cluster node that runs this service may cause data loss.
StorageEnabled	Boolean	RO	Specifies whether or not the local storage is enabled for this cluster member.
StorageEnabledCount	Integer	RO	Specifies the total number of cluster nodes running this service for which local storage is enabled.
TaskAverageDuration	Float	RO	The average duration (in milliseconds) of an individual task execution.
TaskBacklog	Integer	RO	The size of the backlog queue that holds tasks scheduled to be executed by one of the service threads.

Table E-15 (Cont.) ServiceMBean Attributes

Attribute	Type	Access	Description
TaskCount	Long	RO	The total number of executed tasks since the last time the statistics were reset.
TaskHungCount	Integer	RO	The total number of currently executing hung tasks.
TaskHungDuration	Long	RO	The longest currently executing hung task duration in milliseconds.
TaskHungTaskId	String	RO	The id of the of the longest currently executing hung task.
TaskHungThresholdMillis	Long	RW	<p>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.</p> <p>This attribute is applied only if a thread pool is started (that is, the ThreadCount value is > 0).</p>
TaskMaxBacklog	Integer	RO	The maximum size of the backlog queue since the last time the statistics were reset.
TaskTimeoutCount	Integer	RO	The total number of timed-out tasks since the last time the statistics were reset.
TaskTimeoutMillis	Long	RW	<p>The default timeout value in milliseconds for tasks that can be timed-out (for example, implement the <code>com.tangosol.net.PriorityTask</code> interface) but do not explicitly specify the task execution timeout value.</p> <p>This attribute is applied only if a thread pool is started (that is, the ThreadCount value is > 0).</p>
ThreadAbandonedCount	Integer	RO	The number of abandoned threads from the service thread pool. A thread is abandoned and replaced with a new thread if it executes a task for a period of time longer than execution timeout and all attempts to interrupt it fail.
ThreadAverageActiveCount	Float	RO	The average number of active (not idle) threads in the service thread pool since the last time the statistics were reset.
ThreadCount	Integer	RW	The number of threads in the service thread pool. This attribute can only be changed in real time if a service is configured to use a thread pool (that is, a thread count > 0). If the value is 0 (the default) then only the service thread is used and a thread pool is never started. To initially set this value, configure the <code>thread-count</code> parameter either for the service in an operational override file or for a cache in the cache configuration file.
ThreadIdleCount	Integer	RO	The number of currently idle threads in the service thread pool.
Type	String	RO	The type identifier of the service.

Operations

[Table E-16](#) describes the operations for ServiceMBean.

Table E-16 *ServiceMBean Operations*

Operation	Parameters	Return Type	Description
reportOwnership	void	String	Format the ownership info.
resetStatistics	void	void	Reset the service statistics.
shutdown	void	void	Stop the service. This is a controlled shut-down, and is preferred to the stop method.
start	void	void	Start the service.
stop	void	void	Hard-stop the service. Use the shutdown method for normal service termination.

StorageManagerMBean

The `StorageManagerMBean` MBean represents a storage instance for a storage-enabled distributed cache service. A Storage instance manages all index, listener, and lock information for the portion of the distributed cache managed by the local member. A cluster node may have zero or more instances of this managed bean depending on the number of configured distributed caches. The MBean contains usage statistics for the storage-enabled cache.

The MBean's object name is:

`type=StorageManager,service=service name,cache=cache name,nodeId=cluster node's id`

Attributes

[Table E-17](#) describes the attributes for `StorageManagerMBean`.

Table E-17 *StorageManagerMBean Attributes*

Attribute	Type	Access	Description
<code>EventsDispatched</code>	Long	RO	The total number of events dispatched by the Storage Manager since the last time the statistics were reset.
<code>EvictionCount</code>	Long	RO	<p>The number of evictions from the backing map managed by this Storage Manager caused by entries expiry or insert operations that would make the underlying backing map to reach its configured size limit. The eviction count is used to audit the cache size in a static system:</p> $\text{Cache Size} = \text{Insert Count} - \text{Remove Count} - \text{Eviction Count}$ <p>Therefore, the eviction count is not reset by the reset statistics method.</p>
<code>IndexInfo</code>	String[]	RO	An array of information for each index applied to the portion of the partitioned cache managed by the Storage Manager. Each element is a string value that includes a <code>ValueExtractor</code> description, ordered flag (<code>true</code> to indicate that the contents of the index are ordered; <code>false</code> otherwise), and cardinality (number of unique values indexed).
<code>InsertCount</code>	Long	RO	<p>The number of inserts into the backing map managed by this Storage Manager. In addition to standard inserts caused by <code>put</code> and <code>invoke</code> operations or synthetic inserts caused by <code>get</code> operations with read-through backing map topology, this counter is incremented when distribution transfers move resources into the underlying backing map and is decremented when distribution transfers move data out.</p> <p>The insert count is used to audit the cache size in a static system:</p> $\text{Cache Size} = \text{Insert Count} - \text{Remove Count} - \text{Eviction Count}$ <p>Therefore, the insert count is not reset by the reset statistics method.</p>
<code>ListenerFilterCount</code>	Integer	RO	The number of filter-based listeners currently registered with the Storage Manager.

Table E-17 (Cont.) StorageManagerMBean Attributes

Attribute	Type	Access	Description
ListenerKeyCount	Integer	RO	The number of key-based listeners currently registered with the Storage Manager.
ListenerRegistrations	Long	RO	The total number of listener registration requests processed by the Storage Manager since the last time the statistics were reset.
LocksGranted	Integer	RO	The number of locks currently granted for the portion of the partitioned cache managed by the Storage Manager.
LocksPending	Integer	RO	The number of pending lock requests for the portion of the partitioned cache managed by the Storage Manager.
RefreshTime	Date	RO	The timestamp when this model was last retrieved from a corresponding node. For local servers it is the local time.
RemoveCount	Long	RO	<p>The number of removes from the backing map managed by this Storage Manager caused by operations such as <code>clear</code>, <code>remove</code> or <code>invoke</code>.</p> <p>The remove count is used to audit the cache size in a static system:</p> $\text{Cache Size} = \text{Insert Count} - \text{Remove Count} - \text{Eviction Count}$ <p>Therefore, the remove count is not reset by the <code>resetStatistics</code> method.</p>
TriggerInfo	String[]	RO	An array of information for each trigger applied to the portion of the partitioned cache managed by the Storage Manager. Each element is a string value that represents a human-readable description of the corresponding <code>MapTrigger</code> .

Operations

The `StorageManagerMBean` MBean includes a `resetStatistics` operation that resets storage manager statistics. This operation does not reset the `EvictionCount`, `InsertCount` or `RemoveCount` attributes.

TransactionManagerMBean

The `TransactionManagerMBean` MBean represents a transaction manager and is specific to the transactional framework. A cluster node may have zero or more instances of this managed bean depending on the number of configured transaction caches. The MBean 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 MBean's object name is:

```
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 will only be incremented on the MBean on the node that coordinated the commit of that transaction.

Attributes

[Table E-18](#) describes the attributes for `TransactionManagerMBean`.

Table E-18 *TransactionManagerMBean Attributes*

Attribute	Type	Access	Description
<code>CommitTotalMillis</code>	Long	RO	The cumulative time (in milliseconds) spent during the commit phase since the last time statistics were reset.
<code>RefreshTime</code>	Date	RO	The timestamp when this model was last retrieved from a corresponding node. For local servers it is the local time.
<code>TotalActive</code>	Long	RO	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.
<code>TotalCommitted</code>	Long	RO	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.
<code>TotalRecovered</code>	Long	RO	The total number of transactions that have been recovered by the Transaction Manager since the last time the statistics were reset. Note that the count is maintained at the coordinator node for the transaction being recovered, even though multiple nodes may have participated in the transaction.

Table E-18 (Cont.) TransactionManagerMBean Attributes

Attribute	Type	Access	Description
TotalRolledback	Long	RO	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	RO	The cumulative time (in milliseconds) spent on active transactions.
TimeoutMillis	Long	RO	The transaction timeout value in milliseconds. Note that this value will only apply to transactional connections obtained after the value is set. This attribute is currently not supported.
CommitTotalMillis	Long	RO	The cumulative time (in milliseconds) spent during the commit phase since the last time statistics were reset.
RefreshTime	Date	RO	The timestamp when this model was last retrieved from a corresponding node. For local servers it is the local time.
TotalActive	Long	RO	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	RO	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	RO	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	RO	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.

Operations

The `TransactionManagerMBean` MBean includes a `resetStatistics` operation that resets all transaction manager statistics.

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 together as PIF-POF) are derived from a common binary encoding substrate that is described in this appendix. The binary format is provided here for informative purposes and is not a requirement for using PIF-POF. See [Chapter 17, "Using Portable Object Format,"](#) for more information on using PIF-POF.

The following sections are included in this appendix:

- [Stream Format](#)
- [Binary Formats for Predefined Types](#)
- [Binary Format for User Types](#)

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 is able to determine when the Invocation has been fully received. The conversation identifier is used to support multiple logical clients and services multiplexed through a single connection, just as TCP/IP provides multiple logical port numbers for a given IP address. The message object is defined by the particular high-level conversational protocol.

A POF stream contains exactly one Value. The Value contains a Type Identifier, and if the Type Identifier does not imply a value, then it is immediately trailed by a data structure whose format is defined by the Type Identifier.

Integer Values

The stream format relies extensively on the ability to encode integer values in a compact form. Coherence refers to this integer binary format as a *packed integer*. This format uses an initial octet and one or more trailing octets as necessary; it is a variable-length format.

[Table 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
-3 (0x42)	int64
-4 (0x43)	int128*
-5 (0x44)	float32
-6 (0x45)	float64

Table F–4 (Cont.) Predefined Type Identifiers

Type ID	Description
-7 (0x46)	float128*
-8 (0x47)	decimal32*
-9 (0x48)	decimal64*
-10 (0x49)	decimal128*
-11 (0x4A)	boolean
-12 (0x4B)	octet
-13 (0x4C)	octet-string
-14 (0x4D)	char
-15 (0x4E)	char-string
-16 (0x4F)	date
-17 (0x50)	year-month-interval*
-18 (0x51)	time
-19 (0x52)	time-interval*
-20 (0x53)	datetime
-21 (0x54)	day-time-interval*
-22 (0x55)	collection
-23 (0x56)	uniform-collection
-24 (0x57)	array
-25 (0x58)	uniform-array
-26 (0x59)	sparse-array
-27 (0x5A)	uniform-sparse-array
-28 (0x5B)	map
-29 (0x5C)	uniform-keys-map
-30 (0x5D)	uniform-map
-31 (0x5E)	identity
-32 (0x5F)	reference

Type Identifiers less than or equal to -33 are a combination of a type and a value. This form is used to reduce space for these commonly used values.

[Table 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
-34 (0x61)	boolean:true
-35 (0x62)	string:zero-length
-36 (0x63)	collection:empty
-37 (0x64)	reference:null

Table F–5 (Cont.) Type Identifiers that Combine a Type and a Value

Type ID	Description
-38 (0x65)	floating-point:+infinity
-39 (0x66)	floating-point:-infinity
-40 (0x67)	floating-point:NaN
-41 (0x68)	int:-1
-42 (0x69)	int:0
-43 (0x6A)	int:1
-44 (0x6B)	int:2
-45 (0x6C)	int:3
-46 (0x6D)	int:4
-47 (0x6E)	int:5
-48 (0x6F)	int:6
-49 (0x70)	int:7
-50 (0x71)	int:8
-51 (0x72)	int:9
-52 (0x73)	int:10
-53 (0x74)	int:11
-54 (0x75)	int:12
-55 (0x76)	int:13
-56 (0x77)	int:14
-57 (0x78)	int:15
-58 (0x79)	int:16
-59 (0x7A)	int:17
-60 (0x7B)	int:18
-61 (0x7C)	int:19
-62 (0x7D)	int:20
-63 (0x7E)	int:21
-64 (0x7F)	int:22

Binary Formats for Predefined Types

This section describes the binary formats for the predefined ("intrinsic") type identifiers that are supported with PIF-POF. The types are: int, Decimal, Floating Point, Boolean, Octet, Octet String, Char, Char String, Date, Year-Month Interval, Time, Time Interval, Date-Time, Date-Time Interval, Collections, Arrays, Sparse Arrays, Key-Value Maps (Dictionaries), Identity, and Reference.

Int

Four signed integer types are supported: `int16`, `int32`, `int64`, and `int128`. If a type identifier for one of the integer types 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 a number of 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
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` is capable of representing 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 ID	Description
-1 (0x40)	<code>int16</code>
-2 (0x41)	<code>int32</code>
-3 (0x42)	<code>int64</code>
-4 (0x43)	<code>int128</code>
-5 (0x44)	<code>float32</code>
-6 (0x45)	<code>float64</code>
-7 (0x46)	<code>float128</code>
-8 (0x47)	<code>decimal32</code>

Table F-7 (Cont.) Type IDs of Integer Types that can be Coerced into Other Types

Type ID	Description
-9 (0x48)	decimal64
-10 (0x49)	decimal128
-12 (0x4B)	octet
-14 (0x4D)	char

In other words, if the recipient reads any of the above types from the stream and it encounters an encoded integer value, it automatically converts that value into the expected type. This capability allows a set of common (that is, small-magnitude) octet, character, integer, decimal and floating-point values to be encoded using the single-octet integer form (Type Identifiers in the range -41 to -64).

For purposes of unsigned types, the integer value -1 is translated to 0xFF for the octet type, and to 0xFFFF for the char type. (In the case of the char type, this does unfortunately seem to imply a UTF-16 platform encoding; however, it does not violate any of the explicit requirements of the stream format.)

Decimal

There are three floating-point decimal types supported: `decimal32`, `decimal64`, and `decimal128`. If a type identifier for one of the decimal types is encountered in the stream, it is immediately followed by two packed integer values. The first integer value is the unscaled value, and the second is the scale. These values are equivalent to the parameters to the constructor of Java's `BigDecimal` class:

```
java.math.BigDecimal(BigInteger unscaledVal, int scale).
```

In addition to the coercion of integer values into decimal values supported as described in "[Coercion of Integer Types](#)" on page F-6, 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, as well as 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 one of the floating point types 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. Floating point numbers are written to the stream using the IEEE 754 format, and using the IEEE 754r format for the `float128` type.

In addition to the coercion of integer values into decimal values as described in ["Coercion of Integer Types"](#) on page F-6, 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"](#) on page F-6, the only values for the Boolean type are `true` and `false`. As such, the only expected binary formats for Boolean values are the predefined (and compact) forms described in [Table 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). As described in ["Coercion of Integer Types"](#) on page F-6, the compact form of integer values can be used for Octet values, with the integer value -1 being translated as 0xFF.

[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. As described in the section on "[Coercion of Integer Types](#)" on page F-6, the compact form of integer values may be used for Char values, with the integer value -1 being translated as 0xFFFF.

[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
	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 is equal to 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 is written 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
	0x63 (or 0x5500)	n/a
1	0x55016A	0x56410101
1,2,3	0x55036A6B6C	0x564103010203
1, "ok"	0x55026A4E026F6B	n/a

Arrays

An indexed array of values is encoded in a POF stream using the Array type. Immediately following the Type Identifier, the stream contains the Array Size, an Integer Value indicating the number of elements in the Array, which must be greater than or equal to zero. Following the Array Size is the value of the first element of the Array (the zero index), assuming that 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 is equal to 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 is written to 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
	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 ($0 \leq i < n$) 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 is written to 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 ($0 \leq i < n$) 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
	0x63 (or 0x590040)	0x63 (or 0x5A410040) – This example assumes an element type of Int32.
1	0x5901006A40	0x5A4101000140
1,2,3	0x5903006A016B026C40	0x5A410300010102020340
1,,,,5,,,,9	0x5909006A046E087240	0x5A410900010405080940
1,,,,"ok"	0x5905006A044E026F6B40	n/a

Key-Value Maps (Dictionaries)

For key/value pairs, a Key-Value Map (also known as Dictionary data structure) format is used. There are three forms of the Key-Value Map binary encoding:

- The generic map encoding is a sequence of keys and values;
- The uniform-keys-map encoding is a sequence of keys of a uniform type and their corresponding values;
- The uniform-map encoding is a sequence of keys of a uniform type and their corresponding values of a uniform type.

The format for the Key-Value Map is the Type Identifier (map), followed by the Key-Value Map Size n as an Integer Value, followed by n key/value pairs, each of which is composed of a key encoded as Value, and a corresponding value encoded as a Value.

Table 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
	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
	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
	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 more than once 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 will have at most one Identity. Note that values within a uniform data structure cannot be assigned an identity.

Reference

A Reference is a pointer to an Identity that has already 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 already occurred within the POF stream. In the first case, a reference will not be made to an identity that has not yet been encountered, and in the second case, a reference will not be 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 ($0 \leq i$) whose value is greater than the previous Property Index, and a Property Value encoded as a Value; the User Type is finally terminated with an illegal Property Index of -1.

Like the Sparse Array, any property that is not included as part of the User Type encoding is assumed to have a default value. The default value is false for the Boolean type, zero for all numeric, octet and char types, and null for all reference types.

Versioning of User Types

Versioning of User Types supports the addition of properties to a User Type, but not the replacement or removal of properties that existed in previous versions of the User Type. By including the versioning capability as part of the general binary contract, it is possible to support both backwards and forwards 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 will ever be 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.

Log Message Glossary

The following sections are included in this appendix:

- [TCMP Log Messages](#)
- [Configuration Log Messages](#)
- [Partitioned Cache Service Log Messages](#)

TCMP Log Messages

The following are TCMP-related log messages:

Experienced a %n1 ms communication delay (probable remote GC) with Member %s

%n1 - the latency in milliseconds of the communication delay; %s - the full Member information. Severity: 2-Warning or 5-Debug Level 5 or 6-Debug Level 6 depending on the length of the delay.

Cause: This node detected a delay in receiving acknowledgment packets from the specified node, and has determined that it is likely due to a remote GC (rather than a local GC). This message indicates that the overdue acknowledgment has been received from the specified node, and that it has likely emerged from its GC.

Action: Prolonged and frequent GC's can adversely affect cluster performance and availability. If these warnings are seen frequently, review your JVM heap and GC configuration and tuning. See [Chapter 45, "Performance Tuning,"](#) for more details.

Failed to satisfy the variance: allowed=%n1 actual=%n2

%n1 - the maximum allowed latency in milliseconds; %n2 - the actual latency in milliseconds. Severity: 3-Informational or 5-Debug Level 5 depending on the message frequency.

Cause: One of the first steps in the Coherence cluster discovery protocol is the calculation of the clock difference between the new and the senior nodes. This step assumes a relatively small latency for peer-to-peer round trip UDP communications between the nodes. By default, the configured maximum allowed latency (the value of the <maximum-time-variance> configuration element) is 16 milliseconds. See "[incoming-message-handler](#)" on page A-23. Failure to satisfy that latency causes this message to be logged and increases the latency threshold, which will be reflected in a follow up message.

Action: If the latency consistently stays very high (over 100 milliseconds), consult your network administrator and see [Chapter 44, "Performing a Datagram Test for Network Performance."](#)

Created a new cluster "%s1" with Member(%s2)

%s1 - the cluster name; %s2 - the full Member information. Severity: 3-Informational.

Cause: This Coherence node attempted to join an existing cluster in the configured amount of time (specified by the `<join-timeout-milliseconds>` element, see ["multicast-listener"](#) on page A-42), but did not receive any responses from any other node. As a result, it created a new cluster with the specified name (either configured by the `<cluster-name>` element, see ["member-identity"](#) on page A-38, or calculated based on the multicast listener address and port, or the ["well-known-addresses"](#) on page A-86 list). The Member information includes the node id, creation timestamp, unicast address and port, location, process id, role, etc.)

Action: None, if this node is expected to be the first node in the cluster. Otherwise, the operational configuration has to be reviewed to determine the reason that this node does not join the existing cluster.

This Member(%s1) joined cluster "%s2" with senior Member(%s3)

%s1 - the full Member information for this node; %s2 - the cluster name; %s3 - the full Member information for the cluster senior node. Severity: 3-Informational.

Cause: This Coherence node has joined an existing cluster.

Action: None, if this node is expected to join an existing cluster. Otherwise, identify the running cluster and consider corrective actions.

Member(%s) joined Cluster with senior member %n

%s - the full Member information for a new node that joined the cluster this node belongs to; %n - the node id of the cluster senior node. Severity: 5-Debug Level 5.

Cause: A new node has joined an existing Coherence cluster.

Action: None.

Member(%s) left Cluster with senior member %n

%s - the full Member information for a node that left the cluster; %n - the node id of the cluster senior node. Severity: 5-Debug Level 5.

Cause: A node has left the cluster. This departure could be caused by the programmatic shutdown, process termination (normal or abnormal), or any other communication failure (e.g. a network disconnect or a very long GC pause). This message reports the node's departure.

Action: None, if the node departure was intentional. Otherwise, the departed node logs should be analyzed.

MemberLeft notification for Member %n received from Member(%s)

%n - the node id of the departed node; %s - the full Member information for a node that left the cluster. Severity: 5-Debug Level 5.

Cause: When a Coherence node terminates, this departure is detected by nodes earlier than others. Most commonly, a node connected via the TCP ring connection ("TCP ring buddy") would be the first to detect it. This message provides the information about the node that detected the departure first.

Action: None, if the node departure was intentional. Otherwise, the logs for both the departed and the detecting nodes should be analyzed.

Service %s joined the cluster with senior service member %n

%s - the service name; %n - the senior service member id. Severity: 5-Debug Level 5.

Cause: When a clustered service starts on a given node, Coherence initiates a handshake protocol between all cluster nodes running the specified service. This

message serves as an indication that this protocol has been initiated. If the senior node is not known at this time, it will be shown as "n/a".

Action: None.

This node appears to have partially lost the connectivity: it receives responses from MemberSet(%s1) which communicate with Member(%s2), but is not responding directly to this member; that could mean that either requests are not coming out or responses are not coming in; stopping cluster service.

%s1 - set of members that can communicate with the member indicated in %s2; %s2 - member that can communicate with set of members indicated in %s1. Severity: 1-Error.

Cause: The communication link between this member and the member indicated by %s2 has been broken. However, the set of witnesses indicated by %s1 report no communication issues with %s2. It is therefore assumed that this node is in a state of partial failure, thus resulting in the shutdown of its cluster threads.

Action: Corrective action is not necessarily required, since the rest of the cluster presumably is continuing its operation and this node may recover and rejoin the cluster. On the other hand, it may warrant an investigation into root causes of the problem (especially if it is recurring with some frequency).

validatePolls: This senior encountered an overdue poll, indicating a dead member, a significant network issue or an Operating System threading library bug (e.g. Linux NPTL): Poll

Severity: 2-Warning

Cause: When a node joins a cluster, it performs a handshake with each cluster node. A missing handshake response prevents this node from joining the service. The log message following this one will indicate the corrective action taken by this node.

Action: If this message reoccurs, further investigation into the root cause may be warranted.

Received panic from senior Member(%s1) caused by Member(%s2)

%s1 - the cluster senior member as known by this node; %s2 - a member claiming to be the senior member. Severity: 1-Error.

Cause: This occurs after a cluster is split into multiple cluster islands (usually due to a network link failure.) When a link is restored and the corresponding island seniors see each other, the panic protocol is initiated to resolve the conflict.

Action: If this issue occurs frequently, the root cause of the cluster split should be investigated.

Member %n1 joined Service %s with senior member %n2

%n1 - an id of the Coherence node that joins the service; %s - the service name; %n2 - the senior node for the service. Severity: 5-Debug Level 5.

Cause: When a clustered service starts on any cluster node, Coherence initiates a handshake protocol between all cluster nodes running the specified service. This message serves as an indication that the specified node has successfully completed the handshake and joined the service.

Action: None.

Member %n1 left Service %s with senior member %n2

%n1 - an id of the Coherence node that joins the service; %s - the service name; %n2 - the senior node for the service. Severity: 5-Debug Level 5.

Cause: When a clustered service terminates on some cluster node, all other nodes that run this service are notified about this event. This message serves as an indication that the specified clustered service at the specified node has terminated.

Action: None.

Service %s: received ServiceConfigSync containing %n entries

%s - the service name; %n - the number of entries in the service configuration map. Severity: 5-Debug Level 5.

Cause: As a part of the service handshake protocol between all cluster nodes running the specified service, the service senior member updates every new node with the full content of the service configuration map. For the partitioned cache services that map includes the full partition ownership catalog and internal ids for all existing caches. That same message is sent in the case of an abnormal service termination at the senior node, when a new node assumes the service seniority. This message serves as an indication that the specified node has received that configuration update.

Action: None.

TcpRing: connecting to member %n using TcpSocket{%s}

%s - the full information for the TcpSocket that serves as a TcpRing connector to another node; %n - the node id to which this node has connected. Severity: 5-Debug Level 5.

Cause: For quick process termination detection Coherence utilizes a feature called TcpRing, which is a sparse collection of TCP/IP-based connection between different nodes in the cluster. Each node in the cluster is connected to at least one other node, which (if at all possible) is running on a different physical box. This connection is not used for any data transfer; only trivial "heartbeat" communications are sent once a second per each link. This message indicates that the connection between this and specified node is initialized.

Action: None.

Rejecting connection to member %n using TcpSocket{%s}

%n - the node id that tries to connect to this node; %s - the full information for the TcpSocket that serves as a TcpRing connector to another node. Severity: 4-Debug Level 4.

Cause: Sometimes the TCP Ring daemons running on different nodes could attempt to join each other or the same node at the same time. In this case, the receiving node may determine that such a connection would be redundant and reject the incoming connection request. This message is logged by the rejecting node when this happens.

Action: None.

Timeout while delivering a packet; requesting the departure confirmation for Member(%s1) by MemberSet(%s2)

%s1 - the full Member information for a node that this node failed to communicate with; %s2 - the full information about the "witness" nodes that are asked to confirm the suspected member departure. Severity: 2-Warning.

Cause: Coherence uses UDP for all data communications (mostly peer-to-peer unicast), which by itself does not have any delivery guarantees. Those guarantees are built into the cluster management protocol used by Coherence (TCMP). The TCMP daemons are responsible for acknowledgment (ACK or NACK) of all incoming communications. If one or more packets are not acknowledged within the ACK interval ("ack-delay-milliseconds"), they are resent. This repeats until the packets are finally acknowledged or the timeout interval elapses ("timeout-milliseconds"). At this time, this message is logged and the "witness"

protocol is engaged, asking other cluster nodes whether or not they experience similar communication delays with the non-responding node. The witness nodes are chosen based on their roles and location.

Action: Corrective action is not necessarily required, since the rest of the cluster presumably is continuing its operation and this node may recover and rejoin the cluster. On the other hand, it may warrant an investigation into root causes of the problem (especially if it is recurring with some frequency).

This node appears to have become disconnected from the rest of the cluster containing %n nodes. All departure confirmation requests went unanswered. Stopping cluster service.

%n - the number of other nodes in the cluster this node was a member of. Severity: 1-Error.

Cause: Sometime a node that lives within a valid Java process, stops communicating to other cluster nodes. (Possible reasons include: a network failure; extremely long GC pause; swapped out process.) In that case, other cluster nodes may choose to revoke the cluster membership from the paused node and completely shun any further communication attempts by that node, causing this message be logged when the process attempts to resume cluster communications.

Action: Corrective action is not necessarily required, since the rest of the cluster presumably is continuing its operation and this node may recover and rejoin the cluster. On the other hand, it may warrant an investigation into root causes of the problem (especially if it is recurring with some frequency).

A potential communication problem has been detected. A packet has failed to be delivered (or acknowledged) after %n1 seconds, although other packets were acknowledged by the same cluster member (Member(%s1)) to this member (Member(%s2)) as recently as %n2 seconds ago. Possible causes include network failure, poor thread scheduling (see FAQ if running on Windows), an extremely overloaded server, a server that is attempting to run its processes using swap space, and unreasonably lengthy GC times.

%n1 - The number of seconds a packet has failed to be delivered or acknowledged; %s1 - the recipient of the packets indicated in the message; %s2 - the sender of the packets indicated in the message; %n2 - the number of seconds since a packet was delivered successfully between the two members indicated above. Severity: 2-Warning.

Cause: Possible causes are indicated in the text of the message.

Action: If this issue occurs frequently, the root cause should be investigated.

Node %s1 is not allowed to create a new cluster; WKA list: [%s2]

%s1 - Address of node attempting to join cluster; %s2 - List of WKA addresses. Severity: 1-Error.

Cause: The cluster is configured to use WKA, and there are no nodes present in the cluster that are in the WKA list.

Action: Ensure that at least one node in the WKA list exists in the cluster, or add this node's address to the WKA list.

This member is configured with a compatible but different WKA list than the senior Member(%s). It is strongly recommended to use the same WKA list for all cluster members.

%s - the senior node of the cluster. Severity: 2-Warning.

Cause: The WKA list on this node is different than the WKA list on the senior node.

Action: The WKA list on this node is different than the WKA list on the senior node.

UnicastUdpSocket failed to set receive buffer size to %n1 packets (%n2 bytes); actual size is %n3 packets (%n4 bytes). Consult your OS documentation regarding increasing the maximum socket buffer size. Proceeding with the actual value may cause sub-optimal performance.

%n1 - the number of packets that will fit in the buffer that Coherence attempted to allocate; %n2 - the size of the buffer Coherence attempted to allocate; %n3 - the number of packets that will fit in the actual allocated buffer size; %n4 - the actual size of the allocated buffer. Severity: 2-Warning.

Cause: See ["Operating System Tuning"](#) on page 45-1.

Action: See ["Operating System Tuning"](#) on page 45-1.

Configuration Log Messages

The following are configuration-related log messages:

java.io.IOException: Configuration file is missing: "tangosol-coherence.xml"

Severity: 1-Error.

Cause: The operational configuration descriptor cannot be loaded.

Action: Make sure that the `tangosol-coherence.xml` resource can be loaded from the class path specified in the Java command line.

Loaded operational configuration from resource "%s"

%s - the full resource path (URI) of the operational configuration descriptor. Severity: 3-Informational.

Cause: The operational configuration descriptor is loaded by Coherence from the specified location.

Action: If the location of the operational configuration descriptor was explicitly specified via system properties or programmatically, verify that the reported URI matches the expected location.

Loaded operational overrides from "%s"

%s - the URI (file or resource) of the operational configuration descriptor override. Severity: 3-Informational.

Cause: The operational configuration descriptor points to an override location, from which the descriptor override has been loaded.

Action: If the location of the operational configuration descriptor was explicitly specified via system properties, descriptor override or programmatically, verify that the reported URI matches the expected location.

Optional configuration override "%s" is not specified

%s - the URI of the operational configuration descriptor override. Severity: 3-Informational.

Cause: The operational configuration descriptor points to an override location which does not contain any resource.

Action: Verify that the operational configuration descriptor override is not supposed to exist.

java.io.IOException: Document "%s1" is cyclically referenced by the 'xml-override' attribute of element %s2

%s1 - the URI of the operational configuration descriptor or override; %s2 - the name of the XML element that contains an incorrect reference URI. Severity: 1-Error.

Cause: The operational configuration override points to itself or another override that point to it, creating an infinite recursion.

Action: Correct the invalid `xml-override` attribute's value.

java.io.IOException: Exception occurred during parsing: %s

%s - the XML parser error. Severity: 1-Error.

Cause: The specified XML is invalid and cannot be parsed.

Action: Correct the XML document.

Loaded cache configuration from "%s"

%s - the URI (file or resource) of the cache configuration descriptor. Severity: 3-Informational.

Cause: The operational configuration descriptor or a programmatically created `ConfigurableCacheFactory` instance points to a cache configuration descriptor that has been loaded.

Action: Verify that the reported URI matches the expected cache configuration descriptor location.

Partitioned Cache Service Log Messages

The following are partitioned cache-related log messages:

Asking member %n1 for %n2 primary partitions

%n1 - the node id this node asks to transfer partitions from; %n2 - the number of partitions this node is willing to take. Severity: 4-Debug Level 4.

Cause: When a storage-enabled partitioned service starts on a Coherence node, it first receives the configuration update that informs it about other storage-enabled service nodes and the current partition ownership information. That information allows it to calculate the "fair share" of partitions that each node is supposed to own at the end of the re-distribution process. This message demarcates a beginning of the transfer request to a specified node for a number of partitions to move toward the "fair" ownership distribution.

Action: None.

Transferring %n1 out of %n2 primary partitions to member %n3 requesting %n4

%n1 - the number of primary partitions this node transferring to a requesting node; %n2 - the total number of primary partitions this node currently owns; %n3 - the node id that this transfer is for; %n4 - the number of partitions that the requesting node asked for. Severity: 4-Debug Level 4.

Cause: During the partition distribution protocol, a node that owns less than a "fair share" of primary partitions requests any of the nodes that own more than the fair share to transfer a portion of owned partitions. The owner may choose to send any number of partitions less or equal to the requested amount. This message demarcates the beginning of the corresponding primary data transfer.

Action: None.

Transferring %n1 out of %n2 partitions to a machine-safe backup 1 at member %n3 (under %n4)

%n1 - the number of backup partitions this node transferring to a different node; %n2 - the total number of partitions this node currently owns that are "endangered" (do not have a backup); %n3 - the node id that this transfer is for; %n4 - the number of partitions that the transferee can take before reaching the "fair share" amount. Severity: 4-Debug Level 4.

Cause: After the primary partition ownership is completed, nodes start distributing the backups, ensuring the "strong backup" policy, that places backup ownership to nodes running on machines that are different from the primary owners' machines. This message demarcates the beginning of the corresponding backup data transfer.

Action: None.

Transferring backup%n1 for partition %n2 from member %n3 to member %n4

%n1 - the index of the backup partition that this node transferring to a different node; %n2 - the partition number that is being transferred; %n3 the node id of the previous owner of this backup partition; %n4 the node id that the backup partition is being transferred to. Severity: 5-Debug Level 5.

Cause: During the partition distribution protocol, a node that determines that a backup owner for one of its primary partitions is overloaded may choose to transfer the backup ownership to another, underloaded node. This message demarcates the beginning of the corresponding backup data transfer.

Action: None.

Failed backup transfer for partition %n1 to member %n2; restoring owner from: %n2 to: %n3

%n1 the partition number for which a backup transfer was in-progress; %n2 the node id that the backup partition was being transferred to; %n3 the node id of the previous backup owner of the partition. Severity: 4-Debug Level 4.

Cause: This node was in the process of transferring a backup partition to a new backup owner when that node left the service. This node is restoring the backup ownership to the previous backup owner.

Action: None.

Deferring the distribution due to %n1 pending configuration updates

%n1- the number of configuration updates. Severity: 5-Debug Level 5.

Cause: This node is in the process of updating the global ownership map (notifying other nodes about ownership changes) when the periodic scheduled distribution check takes place. Before the previous ownership changes (most likely due to a previously completed transfer) are finalized and acknowledged by the other service members, this node will postpone subsequent scheduled distribution checks.

Action: None.

Limiting primary transfer to %n1 KB (%n2 partitions)

%n1 - the size in KB of the transfer that was limited; %n2 the number of partitions that were transferred. Severity: 4-Debug Level 4.

Cause: When a node receives a request for some number of primary partitions from an underloaded node, it may transfer any number of partitions (up to the requested amount) to the requestor. The size of the transfer is limited by the `<transfer-threshold>` element located within a `<distributed-scheme>` element. This message indicates that the distribution algorithm limited the transfer to the specified number of partitions due to the transfer-threshold.

Action: None.

DistributionRequest was rejected because the receiver was busy. Next retry in %n1 ms

%n1 - the time in milliseconds before the next distribution check will be scheduled. Severity: 6-Debug Level 6.

Cause: This (underloaded) node issued a distribution request to another node asking for one or more partitions to be transferred. However, the other node declined to initiate the transfer as it was in the process of completing a previous transfer with a different node. This node will wait at least the specified amount of time (to allow time for the previous transfer to complete) before the next distribution check.

Action: None.

Restored from backup %n1 partitions

%n1 - the number of partitions being restored. Severity: 3-Informational.

Cause: The primary owner for some backup partitions owned by this node has left the service. This node is restoring those partitions from backup storage (assuming primary ownership). This message is followed by a list of the partitions that are being restored.

Action: None.

Re-publishing the ownership for partition %n1 (%n2)

%n1 the partition number whose ownership is being re-published; %n2 the node id of the primary partition owner, or 0 if the partition is orphaned. Severity: 4-Debug Level 4.

Cause: This node is in the process of transferring a partition to another node when a service membership change occurred, necessitating redistribution. This message indicates this node re-publishing the ownership information for the partition whose transfer is in-progress.

Action: None.

%n1> Ownership conflict for partition %n2 with member %n3 (%n4!=%n5)

%n1 - the number of attempts made to resolve the ownership conflict; %n2 - the partition whose ownership is in dispute; %n3 - the node id of the service member in disagreement about the partition ownership; %n4 - the node id of the partition's primary owner in this node's ownership map; %n5 - the node id of the partition's primary owner in the other node's ownership map. Severity: 4-Debug Level 4.

Cause: If a service membership change occurs while the partition ownership is in-flux, it is possible for the ownership to become transiently out-of-sync and require reconciliation. This message indicates that such a conflict was detected, and denotes the attempts to resolve it.

Action: None.

Assigned %n1 orphaned primary partitions

%n1 - the number of orphaned primary partitions that were re-assigned. Severity: 2-Warning.

Cause: This service member (the most senior storage-enabled) has detected that one or more partitions have no primary owner (orphaned), most likely due to several nodes leaving the service simultaneously. The remaining service members agree on the partition ownership, after which the storage-senior assigns the orphaned partitions to itself. This message is followed by a list of the assigned orphan partitions. This message indicates that data in the corresponding partitions may have been lost.

Action: None.

validatePolls: This service timed-out due to unanswered handshake request.
Manual intervention is required to stop the members that have not responded to this Poll

Severity: 1-Error.

Cause: When a node joins a clustered service, it performs a handshake with each clustered node running the service. A missing handshake response prevents this node from joining the service. Most commonly, it is caused by an unresponsive (e.g. deadlocked) service thread.

Action: Corrective action may require locating and shutting down the JVM running the unresponsive service. See Metalink Note 845363.1 for more details.

<https://metalink.oracle.com/CSP/ui/flash.html#tab=KBHome%28page=KBHome&id=%28%29%29,%28page=KBNavigator&id=%28viewingMode=1143&from=BOOKMARK&bmDocType=HOWTO&bmDocDsrc=KB&bmDocTitle=845363.1&bmDocID=845363.1%29%29>

java.lang.RuntimeException: Storage is not configured

Severity: 1-Error.

Cause: A cache request was made on a service that has no storage-enabled service members. Only storage-enabled service members may process cache requests, so there must be at least one storage-enabled member.

Action: Check the configuration/deployment to ensure that members intended to store cache data are configured to be storage-enabled. This is controlled by the the `<local-storage>` element located within a `<distributed-scheme>` element, or by the `-Dtangosol.coherence.distributed.localstorage` command-line override.

An entry was inserted into the backing map for the partitioned cache "%s" that is not owned by this member; the entry will be removed."

%s - the name of the cache into which insert was attempted. Severity: 1-Error.

Cause: The backing map for a partitioned cache may only contain keys that are owned by that member. Cache requests are routed to the service member owning the requested keys, ensuring that service members will only process requests for keys which they own. This message indicates that the backing map for a cache detected an insertion for a key which is not owned by the member. This is most likely caused by a direct use of the backing-map as opposed to the exposed cache APIs (e.g., `NamedCache`) in user code running on the cache server. This message is followed by a Java exception stack trace showing where the insertion was made.

Action: Examine the user-code implicated by the stack-trace to ensure that any backing-map operations are safe. This error can be indicative of an incorrect implementation of `KeyAssociation`

Exception occurred during filter evaluation: %s; removing the filter...

%s - the description of the filter that failed during evaluation. Severity: 1-Error.

Cause: An exception was thrown while evaluating a filter for a `MapListener` registered on this cache. As a result, some map events may not have been issued. Additionally, to prevent further failures, the filter (and associated `MapListener`) will be removed. This message is followed by a Java exception stack trace showing where the failure occurred.

Action: Review filter implementation and the associated stack trace for errors.

Exception occurred during event transformation: %s; removing the filter...

%s - the description of the filter that failed during event transformation. Severity: 1-Error.

Cause: An Exception was thrown while the specified filter was transforming a `MapEvent` for a `MapListener` registered on this cache. As a result, some map events may not have been issued. Additionally, to prevent further failures, the Filter implementation (and associated `MapListener`) will be removed. This message is followed by a Java exception stack trace showing where the failure occurred.

Action: Review the filter implementation and the associated stack trace for errors.

Exception occurred during index rebuild: %s

%s - the stack trace for the exception that occurred during index rebuild. Severity: 1-Error.

Cause: An Exception was thrown while adding or rebuilding an index. A likely cause of this is a faulty `ValueExtractor` implementation. As a result of the failure, the associated index is removed. This message is followed by a Java exception stack trace showing where the failure occurred.

Action: Review the `ValueExtractor` implementation and associated stack trace for errors.

Exception occurred during index update: %s

%s - the stack trace for the exception that occurred during index update. Severity: 1-Error.

Cause: An Exception was thrown while updating an index. A likely cause of this is a faulty `ValueExtractor` implementation. As a result of the failure, the associated index is removed. This message is followed by a Java exception stack trace showing where the failure occurred.

Action: Review the `ValueExtractor` implementation and associated stack trace for errors.

Exception occurred during query processing: %s

%s - the stack trace for the exception that occurred while processing a query. Severity: 1-Error.

Cause: An Exception was thrown while processing a query. A likely cause of this is an error in the filter implementation used by the query. This message is followed by a Java exception stack trace showing where the failure occurred.

Action: Review the filter implementation and associated stack trace for errors.

BackingMapManager %s1: returned "null" for a cache: %s2

%s1 - the classname of the BackingMapManager implementation that returned a null backing-map; %s2 - the name of the cache for which the BackingMapManager returned null. Severity: 1-Error.

Cause: A `BackingMapManager` returned `null` for a backing-map for the specified cache.

Action: Review the specified `BackingMapManager` implementation for errors and to ensure that it will properly instantiate a backing map for the specified cache.

BackingMapManager %s1: failed to instantiate a cache: %s2

%s1 - the classname of the BackingMapManager implementation that failed to create a backing-map; %s2 - the name of the cache for which the BackingMapManager failed. Severity: 1-Error.

Cause: A `BackingMapManager` unexpectedly threw an Exception while attempting to instantiate a backing-map for the specified cache.

Action: Review the specified BackingMapManager implementation for errors and to ensure that it will properly instantiate a backing map for the specified cache.

BackingMapManager %s1: failed to release a cache: %s2

%s1 - the classname of the BackingMapManager implementation that failed to release a backing-map; %s2 - the name of the cache for which the BackingMapManager failed. Severity: 1-Error.

Cause: A BackingMapManager unexpectedly threw an Exception while attempting to release a backing-map for the specified cache.

Action: Review the specified BackingMapManager implementation for errors and to ensure that it will properly release a backing map for the specified cache.

Unexpected event during backing map operation: key=%s1; expected=%s2; actual=%s3

%s1 - the key being modified by the cache; %s2 - the expected backing-map event from the cache operation in progress; %s3 - the actual MapEvent received. Severity: 6-Debug Level 6.

Cause: While performing a cache operation, an unexpected MapEvent was received on the backing-map. This indicates that a concurrent operation was performed directly on the backing-map and is most likely caused by direct manipulation of the backing-map as opposed to the exposed cache APIs (e.g. NamedCache) in user code running on the cache server.

Action: Examine any user-code that may directly modify the backing map to ensure that any backing-map operations are safe.

Application code running on "%s1" service thread(s) should not call %s2 as this may result in deadlock. The most common case is a CacheFactory call from a custom CacheStore implementation.

%s1 - the name of the service which has made a re-entrant call; %s2 - the name of the method on which a re-entrant call was made. Severity: 2-Warning.

Cause: While executing application code on the specified service, a re-entrant call (a request to the same service) was made. Coherence does not support re-entrant service calls, so any application code (CacheStore, EntryProcessor, and so on...) running on the service thread(s) should avoid making cache requests.

Action: Remove re-entrant calls from application code running on the service thread(s) and consider using alternative design strategies as outlined in [Chapter 29, "Constraints on Re-entrant Calls."](#)

Repeating %s1 for %n1 out of %n2 items due to re-distribution of %s2

%s1 - the description of the request that must be repeated; %n1 - the number of items that are outstanding due to re-distribution; %n2 - the total number of items requested; %s2 - the list of partitions that are in the process of re-distribution and for which the request must be repeated. Severity: 5-Debug Level 5.

Cause: When a cache request is made, the request is sent to the service members owning the partitions to which the request refers. If one or more of the partitions that a request refers to is in the process of being transferred (e.g. due to re-distribution), the request is rejected by the (former) partition owner and is automatically resent to the new partition owner.

Action: None.

Error while starting cluster: com.tangosol.net.RequestTimeoutException: Timeout during service start: ServiceInfo(%s)

%s - information on the service that could not be started. Severity: 1-Error.

Cause: When joining a service, every service in the cluster must respond to the join request. If one or more nodes have a service that does not respond within the timeout period, the join times out.

Action: See Metalink Note 845363.1

<https://metalink.oracle.com/CSP/ui/flash.html#tab=KBHome%28page=KBHome&id=%28%29%29,%28page=KBNavigator&id=%28viewingMode=1143&from=BOOKMARK&bmDocType=HOWTO&bmDocDsrc=KB&bmDocTitle=845363.1&bmDocID=845363.1%29%29>

Failed to restart services: com.tangosol.net.RequestTimeoutException: Timeout during service start: ServiceInfo(%s)

%s - information on the service that could not be started. Severity: 1-Error.

Cause: When joining a service, every service in the cluster must respond to the join request. If one or more nodes have a service that does not respond within the timeout period, the join times out.

Action: See Metalink Note 845363.1

<https://metalink.oracle.com/CSP/ui/flash.html#tab=KBHome%28page=KBHome&id=%28%29%29,%28page=KBNavigator&id=%28viewingMode=1143&from=BOOKMARK&bmDocType=HOWTO&bmDocDsrc=KB&bmDocTitle=845363.1&bmDocID=845363.1%29%29>

