

Oracle® Coherence

Integration Guide for Oracle Coherence

Release 3.5

E14537-01

June 2009

Oracle Coherence Integration Guide for Oracle Coherence, Release 3.5

E14537-01

Copyright © 2009, Oracle and/or its affiliates. All rights reserved.

Primary Author: Thomas Pfaeffle

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

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	vii
Audience.....	vii
Documentation Accessibility	vii
Related Documents	viii
Conventions	viii
1 Configuring Coherence for JPA	
Obtaining a JPA Implementation	1-1
Conventions	1-1
Using the Coherence JpaCacheStore	1-2
Mapping the Persistent Classes	1-2
Configuring JPA	1-2
Configuring Coherence	1-3
Configuring the Persistence Unit.....	1-4
2 Integrating TopLink Grid and Oracle Coherence	
Using TopLink Grid as the CacheStore for Coherence	2-1
Configuring an EclipseLinkJPACacheStore	2-1
Using Coherence as the Toplink Cache	2-2
Coherence and TopLink Grid API and Files.....	2-3
3 Integrating Hibernate and Oracle Coherence	
Using Hibernate as a CacheStore for Coherence	3-1
HibernateCacheStore and HibernateCacheLoader API	3-1
Configuring a HibernateCacheStore	3-2
Configuration Requirements	3-4
JDBC Isolation Level.....	3-4
Fault-Tolerance	3-5
Extending HibernateCacheStore.....	3-5
Creating a Hibernate CacheStore.....	3-5
Re-entrant Calls.....	3-5
Using Fully Cached DataSets	3-5
Distributed Queries	3-5

Detached Processing.....	3-6
Using Coherence as the Hibernate L2 Cache	3-6
Hibernate and Caching.....	3-6
Configuration and Tuning.....	3-6
Specifying a Coherence Cache Topology	3-7
Cache Concurrency Strategies.....	3-7
Query Cache.....	3-8
Fault-Tolerance.....	3-8
Deployment.....	3-8

4 Integrating an Oracle Coherence CacheFactory with Spring

5 Integrating WebLogic Portal and Oracle Coherence

P13N CacheProvider SPI Implementation	5-1
Sharing Data Between WSRP-Federated Portals Using Coherence	5-2

Index

List of Examples

1-1	JPA-related Classes and Interfaces	1-1
1-2	Sample persistence.xml File for JPA	1-2
1-3	Assigning Named Caches to a JPA Caching Scheme	1-3
2-1	Sample coherence-cache-config.xml File for EclipseLink	2-2
3-1	Sample coherence-cache-config.xml File for Hibernate	3-2
3-2	Sample coherence-cache-config.xml File that Uses {cache-name} Macro.....	3-3
3-3	Sample coherence-cache-config.xml File with Generalized Mappings	3-3
3-4	Specifying a Coherence Provider Class	3-6
4-1	Configuring the Cache to use SpringAwareCacheFactory	4-1
4-2	Configuring a SpringAwareCacheFactory Programmatically	4-2
4-3	Defining a SpringAwareCacheFactory in an Application Context	4-2
4-4	Configuring a CacheFactory in an Application Context.....	4-2
4-5	Configuring a CacheStore in an Application Context.....	4-2
4-6	Configuring Setter Injection to Set Properties on the Bean.....	4-3
4-7	SpringAwareCacheFactory.java	4-4

List of Tables

2-1	TopLink Grid Classes for EclipseLink JPA	2-1
2-2	EclipseLink Classes Typically used in a Toplink Grid/Coherence Integration	2-3

Preface

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

Audience

This document is targeted at software developers and architects who will be integrating Coherence with EclipseLink Essentials, Hibernate, and Spring.

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.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at <http://www.fcc.gov/cgb/consumerfacts/trs.html>, and a list of phone numbers is available at <http://www.fcc.gov/cgb/dro/trsphonebk.html>.

Related Documents

For more information on Oracle Coherence, see the following documents in the Release 3.5 documentation set:

- *Getting Started for Oracle Coherence*
- *Developer's Guide for Oracle Coherence*
- *Client Guide for Oracle Coherence*
- *Tutorial for Oracle Coherence*
- *User's Guide for Oracle Coherence*Web*

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.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Configuring Coherence for JPA

The Java Persistence API (JPA) is the standard for Object-Relational mapping (ORM) and enterprise Java persistence. Several open source and commercial implementations exist and are being developed.

Coherence ships a `CacheStore` implementation that uses JPA to load and store objects to the database. This chapter describes how to configure and use this `CacheStore`.

Note: Only resource-local and bootstrapped entity managers are currently supported. Container-managed entity managers and those that use JTA transactions are not currently supported.

Obtaining a JPA Implementation

A JPA provider is not shipped with Coherence, but is easy to obtain. Although the JPA `CacheStore` works with any JPA-compliant implementation, Oracle recommends using EclipseLink JPA, the Reference Implementation for the JPA 2.0 specification. Oracle is leading the open source EclipseLink project that includes EclipseLink JPA. EclipseLink is available from Eclipse at the following URL:

<http://www.eclipse.org/eclipselink>

Oracle Toplink includes EclipseLink as its JPA implementation. You can find more information about Toplink and EclipseLink at the following URL:

<http://www.oracle.com/technology/products/ias/toplink/index.html>

Conventions

This chapter refers to the following Java classes and interfaces:

Example 1-1 JPA-related Classes and Interfaces

```
com.tangosol.coherence.jpa.JpaCacheLoader
com.tangosol.coherence.jpa.JpaCacheStore

com.tangosol.net.NamedCache (extends java.util.Map)

com.tangosol.net.cache.CacheLoader
com.tangosol.net.cache.CacheStore
```

As the `CacheStore` interface extends `CacheLoader`, the term "CacheStore" is used generically to refer to both interfaces (the appropriate interface being determined by

whether read-only or read-write support is required). Similarly, "JpaCacheStore" refers to both implementations.

The Coherence cache configuration file is referred to as the `coherence-cache-config.xml` (the default name). The JPA persistence implementation is referred to simply as the JPA provider. The JPA run-time configuration file is referred to as the `persistence.xml`, and the JPA Object-Relational mapping file is referred to as the `orm.xml` (the default name).

Using the Coherence JpaCacheStore

The JPA is a standard API for mapping, querying and storing Java objects to a database. The characteristics of the different JPA implementations may differ, however, when it comes to caching, threading, and overall performance. TopLink provides a high-performance JPA implementation with many advanced features.

Coherence includes a default entity-based `CacheStore` implementation, `JpaCacheStore` (and a corresponding `CacheLoader` implementation, `JpaCacheLoader`). You can find additional information in the Javadoc for the implementing classes.

Mapping the Persistent Classes

The first step in being able to load and store objects through the `CacheStore` is to ensure that the classes are mapped to the database. JPA mappings are standard, and can be specified in the same way for all JPA providers.

Entities may be mapped either by annotating the entity classes or by adding an `orm.xml` or other XML mapping file. See the JPA provider documentation for more on how to map JPA entities.

Configuring JPA

A typical JPA configuration involves making changes to the `persistence.xml` file. Within the `persistence.xml` are the properties that dictate run-time operation.

[Example 1-2](#) is a sample `persistence.xml` illustrating the typical properties that are set. The transaction type should be set to `RESOURCE_LOCAL` and the required JDBC properties (`eclipselink.jdbc.driver`, `url`, `user`, and `password`) should contain the appropriate values for connecting and logging into your database. Classes that are mapped using JPA annotations should be listed in `<class>` elements.

Example 1-2 Sample persistence.xml File for JPA

```
<persistence xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance" version="1.0"
xmlns="http://java.sun.com/xml/ns/persistence">
  <persistence-unit name="EmpUnit" transaction-type="RESOURCE_LOCAL">
    <provider>
      org.eclipse.persistence.jpa.PersistenceProvider
    </provider>
    <class>com.oracle.coherence.handson.Employee</class>
    <properties>
      <property name="eclipselink.jdbc.driver"
value="oracle.jdbc.OracleDriver"/>
      <property name="eclipselink.jdbc.url"
value="jdbc:oracle:thin:@localhost:1521:XE"/>
      <property name="eclipselink.jdbc.user" value="scott"/>
      <property name="eclipselink.jdbc.password" value="tiger"/>
    </properties>
```

```

</persistence-unit>
</persistence>

```

Configuring Coherence

A `coherence-cache-config.xml` must be specified to override the default Coherence settings and define the `JpaCacheStore` caching scheme. The caching scheme should include a `<cachestore-scheme>` element that lists the `JpaCacheStore` class and includes three parameters.

- The entity name of the entity being stored. Unless it is explicitly overridden in JPA it is the unqualified name of the entity class. [Example 1-3](#) uses the built-in Coherence macro `{cache-name}` that translates to the name of the cache that is constructing and using the `CacheStore`. This works because a separate cache must be used for each type of persistent entity and Coherence ensures that the name of each cache is set to the name of the entity that is being stored in it.
- The fully qualified name of the entity class. If the classes are all in the same package and use the default JPA entity names then you can again use the `{cache-name}` macro to fill in the part that is variable across the different entity types. In this way the same caching scheme can be used for all of the entities that are cached within the same persistence unit.
- The persistence unit name, which should be the same as the name specified in the `persistence.xml`.

The various named caches are then directed to use the JPA caching scheme. [Example 1-3](#) is a sample `coherence-cache-config.xml` which defines a `NamedCache` named `Employee` that caches instances of the `Employee` class. To define additional entity caches for more classes then more `<cache-mapping>` elements may be added.

Example 1-3 Assigning Named Caches to a JPA Caching Scheme

```

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <!-- Set the name of the cache to be the entity name -->
      <cache-name>Employee</cache-name>
      <!-- Configure this cache to use the scheme defined below -->
      <scheme-name>jpa-distributed</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>
  <caching-schemes>
    <distributed-scheme>
      <scheme-name>jpa-distributed</scheme-name>
      <service-name>JpaDistributedCache</service-name>
      <backing-map-scheme>
        <read-write-backing-map-scheme>
          <internal-cache-scheme>
            <local-scheme/>
          </internal-cache-scheme>
          <!-- Define the cache scheme -->
          <cachestore-scheme>
            <class-scheme>
              <class-name>
                com.tangosol.coherence.jpa.JpaCacheStore
              </class-name>
            </class-scheme>
          </cachestore-scheme>
        </read-write-backing-map-scheme>
      </backing-map-scheme>
    </distributed-scheme>
  </caching-schemes>
</cache-config>

```

```

<!-- This param is the entity name -->
<init-param>
  <param-type>java.lang.String</param-type>
  <param-value>{cache-name}</param-value>
</init-param>

<!-- This param is the fully qualified entity class -->
<init-param>
  <param-type>java.lang.String</param-type>
  <param-value>com.acme.{cache-name}</param-value>
</init-param>

<!-- This param should match the value of the -->
<!-- persistence unit name in persistence.xml -->
<init-param>
  <param-type>java.lang.String</param-type>
  <param-value>EmpUnit</param-value>
</init-param>
</init-params>
</class-scheme>
</cachestore-scheme>
</read-write-backing-map-scheme>
</backing-map-scheme>
</distributed-scheme>
</caching-schemes>
</cache-config>

```

Configuring the Persistence Unit

When using `com.tangosol.coherence.jpa.JpaCacheStore`, entities, the persistence unit should be configured to ensure that no changes are made to entities when they are inserted or updated. Any changes made to entities by the JPA provider are not reflected in the Coherence cache and so the entity in the cache will not match the database contents. In particular, entities should not use ID generation, for example, `@GeneratedValue`, to obtain an ID. IDs should be assigned in application code before an object is put into Coherence. The ID is typically the key under which the entity is stored in Coherence.

Optimistic locking (for example, `@Version`) should not be used as it may lead to a database transaction commit failure. See *"Read-Through, Write-Through, Write-Behind, and Refresh-Ahead Caching"* in the *Oracle Coherence Getting Started Guide* and *Sample CacheStores* in the *Oracle Coherence Developer's Guide* for more information on the workings of a CacheStore and how to set up your database schema.

When using either `com.tangosol.coherence.jpa.JpaCacheStore` or `com.tangosol.coherence.jpa.JpaCacheLoader` L2 caching should be disabled in your persistence unit. Consult the documentation for your provider. In `TopLink`, this can be specified on an individual entity with `@Cache(shared=false)` or as the default in the `persistence.xml` with the following property:

```
<property name="eclipselink.cache.shared.default" value="false"/>
```

Integrating TopLink Grid and Oracle Coherence

Oracle TopLink Grid is a feature of Oracle TopLink that provides integration between the EclipseLink JPA and Coherence. TopLink Grid enables you to combine the simplicity of application development using the Java Persistence API (JPA) with the scalability and distributed processing power of Oracle Coherence Data Grid. Standard JPA applications interact directly with their primary data store, typically a relational database; however, with TopLink Grid you can store some or all of your domain model in the Coherence data grid.

Using TopLink Grid as the CacheStore for Coherence

TopLink Grid provides a default entity-based `CacheStore` implementation, `EclipseLinkJPACacheStore`, and a corresponding `CacheLoader` implementation, `EclipseLinkJPACacheLoader` in the `oracle.eclipselink.coherence.standalone` package. These are optimized implementations for use with EclipseLink JPA. The classes and their Javadoc can be found in the `toplink-grid.jar` file.

Note: Before the Oracle Release 11gR1, the API for TopLink Grid/Coherence integration was shipped in a JAR named `coherence-eclipselink.jar`.

Table 2-1 TopLink Grid Classes for EclipseLink JPA

Class Name	Description
<code>oracle.eclipselink.coherence.standalone.CacheStore</code>	An optimized implementation of <code>CacheStore</code> for use with EclipseLink JPA
<code>oracle.eclipselink.coherence.standalone.CacheLoader</code>	An optimized implementation of <code>CacheLoader</code> for use with EclipseLink JPA

Configuring an EclipseLinkJPACacheStore

This section illustrates a simple `EclipseLinkJPACacheStore` constructor, which requires the name of the cache (which is the unqualified name of the entity) and the name of the persistence unit. It configures Toplink Grid using the default configuration path, which looks for the configuration files `persistence.xml` and `orm.xml` in the classpath. There is also the ability to pass in a resource name or file specification for the EclipseLink configuration file as the third `<init-param>` (set the `<param-type>`

element to `java.lang.String` for a resource name and `java.io.File` for a file specification).

Example 2-1 illustrates a simple `coherence-cache-config.xml` file used to define a `NamedCache` named `Publisher` which caches instances of the entity (`novels`). To add additional entity caches, add additional `<cache-mapping>` elements.

Example 2-1 Sample coherence-cache-config.xml File for EclipseLink

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

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>Publisher</cache-name>
      <scheme-name>distributed-eclipselink</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <distributed-scheme>
      <scheme-name>distributed-eclipselink</scheme-name>
      <service-name>EclipseLinkJPA</service-name>
      <backing-map-scheme>
        <read-write-backing-map-scheme>
          <internal-cache-scheme>
            <local-scheme/>
          </internal-cache-scheme>
        </read-write-backing-map-scheme>
      </backing-map-scheme>
      <autostart>true</autostart>
    </distributed-scheme>
  </caching-schemes>
</cache-config>

<!-- Define the cache scheme -->
  <cachestore-scheme>
    <class-scheme>
      <class-name>
        oracle.eclipselink.coherence.standalone.EclipseLinkJPACacheStore
      </class-name>
      <init-params>
        <init-param>
          <param-type>java.lang.String</param-type>
          <param-value>{cache-name}</param-value>
        </init-param>
        <init-param>
          <param-type>java.lang.String</param-type>
          <param-value>novels</param-value>
        </init-param>
      </init-params>
    </class-scheme>
  </cachestore-scheme>
</read-write-backing-map-scheme>
</backing-map-scheme>
</distributed-scheme>
</caching-schemes>
</cache-config>
```

Using Coherence as the Toplink Cache

You can easily configure Toplink Grid to use Coherence as the primary data store, execute queries against the grid, and allow Coherence to manage the persistence of new and modified data. Coherence provides the layer between JPA and the datastore,

where direct database calls can be off-loaded from every application instance. This makes it possible for clustered application deployments to scale beyond the bounds of normal database operations.

There are three typical TopLink Grid configurations that applications can use:

- **Coherence Shared L2 Cache**—this configuration uses Coherence as the TopLink L2 (Shared) cache. This configuration applies Coherence data grid to JPA applications that rely on database hosted data that cannot be entirely pre-loaded into a Coherence cache. Some reasons why it might not be able to pre-loaded include extremely complex queries that exceed the feature set of Coherence Filters, third party database updates that create stale caches, reliance on native SQL queries, stored procedures or triggers, and so on.

In this configuration, you can scale TopLink up into large clusters while avoiding the requirement to coordinate local L2 caches. Updates made to entities are available in all Coherence cluster members immediately, upon a transaction commit.

- **Coherence Read**—this configuration is optimal for entities that require fast access to large amounts of (fairly stable) data and must write changes synchronously to the database. In these entities, cache warming would be used to populate the Coherence cache, but individual queries could be directed to the database if necessary.
- **Coherence Read/Write**—this configuration is optimal for applications that require fast access to large amounts of (fairly stable) data and that perform relatively few updates. This configuration can be combined with a `CoherenceCacheStore` using write-behind to improve application response time by performing database updates asynchronously.

See the Oracle TopLink Grid page on the Oracle Technology Network for more information on configuring Coherence for Toplink Grid.

Coherence and TopLink Grid API and Files

The API for TopLink Grid/Coherence integration are shipped in the `toplink-grid.jar`. The integration also uses the standard JPA run-time configuration file `persistence.xml` and the JPA mapping file `orm.xml` (the default name). The Coherence cache configuration file `coherence-cache-config.xml` (the default name) must be specified to override the default Coherence settings and define the `CacheStore` caching scheme.

[Table 2–2](#) lists several noteworthy EclipseLink classes which are typically used in a TopLink Grid/Coherence integration. The classes and their Javadoc can be found in the `toplink-grid.jar` file.

Table 2–2 EclipseLink Classes Typically used in a Toplink Grid/Coherence Integration

Class Name	Description
<code>oracle.eclipseLink.coherence.integrated.EclipseLinkJPACacheLoader</code>	Provides JPA-aware versions of the Coherence <code>CacheLoader</code> .
<code>oracle.eclipseLink.coherence.integrated.EclipseLinkJPACacheStore</code>	Provides JPA-aware versions of the Coherence <code>CacheStore</code> .
<code>oracle.eclipseLink.coherence.integrated.cache.CoherenceInterceptor</code>	Intercepts all TopLink calls to the internal TopLink L2 cache and redirects them to Coherence.

Table 2–2 (Cont.) EclipseLink Classes Typically used in a Toplink Grid/Coherence Integration

Class Name	Description
oracle.eclipselink.coherence.integrated.config. CoherenceReadCustomizer	Used to program a Coherence read configuration.
oracle.eclipselink.coherence.integrated.config. CoherenceReadWriteCustomizer	Used to program a Coherence read/write configuration.
oracle.eclipselink.coherence.integrated. querying.IgnoreDefaultRedirector	Allows queries to bypass the Coherence cache and be directly sent to the database.

Integrating Hibernate and Oracle Coherence

Hibernate is an object/relational mapping tool for Java environments. The functionality in Oracle Coherence and Hibernate can be combined in several ways.

- [Using Hibernate as a CacheStore for Coherence](#)
- [Using Coherence as the Hibernate L2 Cache](#)

Using Hibernate as a CacheStore for Coherence

Coherence includes a default entity-based `CacheStore` implementation, `HibernateCacheStore` (and a corresponding `CacheLoader` implementation, `HibernateCacheLoader`) in the `com.tangosol.coherence.hibernate` package.

HibernateCacheStore and HibernateCacheLoader API

The `HibernateCacheStore` and `HibernateCacheLoader` API provide the following constructors:

- `HibernateCacheLoader()`, `HibernateCacheStore()`—Default constructors which creates a new instance of a `CacheLoader` or `CacheStore`. They do not create a `SessionFactory`. To create a `SessionFactory`, use the `setSession()` method.
- `HibernateCacheLoader(java.lang.String entityName)`, `HibernateCacheStore(java.lang.String entityName)`—Creates a `SessionFactory` using the default Hibernate configuration (`hibernate.cfg.xml`) in the classpath.
- `HibernateCacheStore(java.lang.String entityName, java.lang.String sResource)`, `HibernateCacheStore(java.lang.String entityName, java.lang.String sResource)`—Creates a `SessionFactory` based on the configuration file provided (`sResource`)
- `HibernateCacheLoader(java.lang.String entityName, java.io.File configurationFile)`, `HibernateCacheStore(java.lang.String entityName, java.io.File configurationFile)`—Creates a `SessionFactory` based on the configuration file provided (`configurationFile`)
- `HibernateCacheStore(java.lang.String entityName, org.hibernate.SessionFactory sFactory)`, `HibernateCacheStore(java.lang.String entityName, org.hibernate.SessionFactory sFactory)`—Cache store constructors which accept an `entityName` and a Hibernate session factory.

More detailed technical information may be found in the Javadoc for the implementing classes.

Configuring a HibernateCacheStore

The examples below show a simple `HibernateCacheStore` constructor, accepting only an entity name. This configures Hibernate using the default configuration path, which looks for a `hibernate.cfg.xml` file in the classpath. There is also the ability to pass in a resource name or file specification for the `hibernate.cfg.xml` file as the second `<init-param>` (set the `<param-type>` element to `java.lang.String` for a resource name and `java.io.File` for a file specification). See `HibernateCacheStore` for more details.

[Example 3-1](#) illustrates a simple `coherence-cache-config.xml` file used to define a `NamedCache` named `TableA` which caches instances of a Hibernate entity (`com.company.TableA`). To add additional entity caches, add additional `<cache-mapping>` elements.

Example 3-1 Sample coherence-cache-config.xml File for Hibernate

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

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

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>TableA</cache-name>
      <scheme-name>distributed-hibernate</scheme-name>
      <init-params>
        <init-param>
          <param-name>entityname</param-name>
          <param-value>com.company.TableA</param-value>
        </init-param>
      </init-params>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <distributed-scheme>
      <scheme-name>distributed-hibernate</scheme-name>
      <backing-map-scheme>
        <read-write-backing-map-scheme>
          <internal-cache-scheme>
            <local-scheme></local-scheme>
          </internal-cache-scheme>

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

```

        </cachestore-scheme>
    </read-write-backing-map-scheme>
</backing-map-scheme>
</distributed-scheme>
</caching-schemes>
</cache-config>

```

[Example 3-2](#) illustrates that it is also possible to use the pre-defined `{cache-name}` macro to eliminate the need for the `<init-params>` portion of the cache mapping.

Example 3-2 Sample coherence-cache-config.xml File that Uses {cache-name} Macro

```

<?xml version="1.0"?>

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

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>TableA</cache-name>
      <scheme-name>distributed-hibernate</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <distributed-scheme>
      <scheme-name>distributed-hibernate</scheme-name>
      <backing-map-scheme>
        <read-write-backing-map-scheme>
          <internal-cache-scheme>
            <local-scheme></local-scheme>
          </internal-cache-scheme>

          <cachestore-scheme>
            <class-scheme>
              <class-name>
                com.tangosol.coherence.hibernate.HibernateCacheStore
              </class-name>
              <init-params>
                <init-param>
                  <param-type>java.lang.String</param-type>
                  <param-value>com.company.{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>

```

[Example 3-3](#) illustrates that, if naming conventions allow, the mapping may be completely generalized to allow a cache mapping for any qualified class name (entity name).

Example 3-3 Sample coherence-cache-config.xml File with Generalized Mappings

```

<?xml version="1.0"?>

```

```

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

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

  <caching-schemes>
    <distributed-scheme>
      <scheme-name>distributed-hibernate</scheme-name>
      <backing-map-scheme>
        <read-write-backing-map-scheme>
          <internal-cache-scheme>
            <local-scheme></local-scheme>
          </internal-cache-scheme>

          <cachestore-scheme>
            <class-scheme>
              <class-name>
                com.tangosol.coherence.hibernate.HibernateCacheStore
              </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>

```

Configuration Requirements

Hibernate entities accessed by using the `HibernateCacheStore` module must use the "assigned" ID generator and also have a defined ID property.

Be sure to disable the `hibernate.hbm2ddl.auto` property in the `hibernate.cfg.xml` used by the `HibernateCacheStore`, as this may cause excessive schema updates (and possible lockups).

JDBC Isolation Level

In cases where all access to a database is through Coherence, `CacheStore` modules naturally enforce ANSI-style Repeatable Read isolation as reads and writes are executed serially on a per-key basis (by using the Partitioned Cache Service). Increasing database isolation above Repeatable Read do not yield increased isolation as `CacheStore` operations may span multiple Partitioned Cache nodes (and thus multiple database transactions). Using database isolation levels below Repeatable Read do not result in unexpected anomalies, and may reduce processing load on the database server.

Fault-Tolerance

For single-cache-entry updates, `CacheStore` operations are fully fault-tolerant in that the cache and database are guaranteed to be consistent during any server failure (including failures during partial updates). While the mechanisms for fault-tolerance vary, this is true for both write-through and write-behind caches.

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 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 have been addressed earlier in the design process).

Extending HibernateCacheStore

In some cases, it may be desired to extend the `HibernateCacheStore` with application-specific functionality. The most obvious reason for this is to leverage a pre-existing, programmatically-configured `SessionFactory` instance.

Creating a Hibernate CacheStore

While the provided `HibernateCacheStore` module provides a solution for most entity-based caches, there may be cases where an application-specific `CacheStore` module is necessary. For example, providing parameterized queries or including or post-processing of query results.

Re-entrant Calls

In a `CacheStore`-backed cache implementation, when the application thread accesses cached data, the cache operations may trigger a call to the associated `CacheStore` implementation by using the managing `CacheService`. The `CacheStore` must not call back into the `CacheService` API. This implies, indirectly, that Hibernate should not attempt to access cache data. Therefore, all methods in `CacheLoader` or `CacheStore` should be careful to call `Session.setCacheMode(CacheMode.IGNORE)` to disable cache access. Alternatively, the Hibernate configuration may be cloned (either programmatically or by using `hibernate.cfg.xml`), with `CacheStore` implementations using the version with the cache disabled.

Using Fully Cached DataSets

There are two scenarios where using fully-cached datasets would be advantageous. One is when you are performing distributed queries on the cache; the other is when you want to provide continued application processing despite a database failure.

Distributed Queries

Distributed queries offer the potential for lower latency, higher throughput and less database server load compared to executing queries on the database server. For set-oriented queries, the dataset must be entirely cached to produce correct query results. More precisely, for a query issued against the cache to produce correct results, the query must not depend on any uncached data.

Distributed queries enable you to create hybrid caches. For example, it is possible to combine two uses of a `NamedCache`: a fully cached size-limited dataset for querying (for example, the data for the most recent week), and a partially cached historical dataset used for singleton reads. This is a good approach to avoid data duplication and minimize memory usage.

While fully cached datasets are usually bulk-loaded during application startup (or on a periodic basis), `CacheStore` integration may be used to ensure that both cache and database are kept fully synchronized.

Detached Processing

Another reason for using fully-cached datasets is to provide the ability to continue application processing even if the underlying database goes down. Using write-behind caching extends this mode of operation to support full read-write applications. With write-behind, the cache becomes (in effect) the temporary system of record. Should the database fail, updates are queued in Coherence until the connection is restored. At this point, all cache changes are sent to the database.

Using Coherence as the Hibernate L2 Cache

Coherence can be used as the L2 cache provider for Hibernate.

Hibernate and Caching

Hibernate supports three primary forms of caching:

- Session cache
- L2 cache
- Query cache

The `Session` cache is responsible for caching records within a `Session` (a Hibernate transaction, potentially spanning multiple database transactions, and typically scoped on a per-thread basis). As a non-clustered cache (by definition), the `Session` cache is managed entirely by Hibernate. The L2 and Query caches span multiple transactions, and support the use of Coherence as a cache provider. The L2 cache is responsible for caching records across multiple sessions (for primary key lookups). The query cache caches the result sets generated by Hibernate queries. Hibernate manages data in an internal representation in the L2 and Query caches, meaning that these caches are usable only by Hibernate. For more details, see the [Hibernate Reference Documentation](#) (shipped with Hibernate), specifically the section on the Second Level Cache.

Configuration and Tuning

To use the Coherence Caching Provider for Hibernate, specify the Coherence provider class in the `hibernate.cache.provider_class` property. Typically this is configured in the default Hibernate configuration file, `hibernate.cfg.xml`.

[Example 3-4](#) illustrates the property element calling `CoherenceCacheProvider` as the `hibernate.cache.provider_class`.

Example 3-4 Specifying a Coherence Provider Class

```
<property name="hibernate.cache.provider_class">com.tangosol.coherence.hibernate.CoherenceCacheProvider</property>
```

The file `coherence-hibernate.jar` (found in the `lib/` subdirectory) must be added to the application classpath.

Hibernate provides the configuration property `hibernate.cache.use_minimal_puts`, which optimizes cache access for clustered caches by increasing cache reads and decreasing cache updates. This is enabled by default by the Coherence Cache Provider. Setting this property to `false` may increase overhead for cache management and also increase the number of transaction rollbacks.

The Coherence Caching Provider includes a setting for how long a lock acquisition should be attempted before timing out. This may be specified by the Java property `tangosol.coherence.hibernate.lockattemptmillis`. The default is one minute.

Specifying a Coherence Cache Topology

By default, the Coherence Caching Provider uses a custom cache configuration located in the `coherence-hibernate.jar` named `config/hibernate-cache-config.xml`. This configuration file is used to define cache mappings for Hibernate L2 caches. If desired, an alternative cache configuration resource may be specified for Hibernate L2 caches by using the `tangosol.coherence.hibernate.cacheconfig` Java property. It is possible to configure this property to point to the application's main `coherence-cache-config.xml` file if mappings are properly configured. It may be beneficial to use dedicated cache service(s) to manage Hibernate-specific caches to ensure that any `CacheStore` modules do not cause re-entrant calls back into Coherence-managed Hibernate L2 caches.

In the scheme mapping section of the Coherence cache configuration file, the `hibernate.cache.region_prefix` property can be used to specify a cache topology. For example, if the cache configuration file includes a wildcard mapping for `near-*`, and the Hibernate region prefix property is set to `near-`, then all Hibernate caches are named using the `near-` prefix, and use the cache scheme mapping specified for the `near-*` cache name pattern.

It is possible to specify a cache topology per entity by creating a cache mapping based on the combined prefix and qualified entity name (for example, `near-com.company.EntityName`); or equivalently, by providing an empty prefix and specifying a cache mapping for each qualified entity name.

Also, L2 caches should be size-limited to avoid excessive memory usage. Query caches in particular must be size-limited as the Hibernate API does not provide any means of controlling the query cache other than a complete eviction.

Cache Concurrency Strategies

Hibernate generally emphasizes the use of optimistic concurrency for both cache and database. With optimistic concurrency in particular, transaction processing depends on having accurate data available to the application at the beginning of the transaction. If the data is inaccurate, then commit processing detects that the transaction was dependent on incorrect data, and the transaction fails to commit. While most optimistic transactions must cope with changes to underlying data by other processes, the use of caching adds the possibility of the cache itself being stale. Hibernate provides several cache concurrency strategies to control updates to the L2 cache. While this is less of an issue for Coherence due to support for cluster-wide coherent caches, appropriate selection of cache concurrency strategy aids application efficiency.

Note that cache configuration strategies may be specified at the table level. Generally, the strategy should be specified in the mapping file for the class.

For mixed read-write activity, the read-write strategy is recommended. The transactional strategy is implemented similarly to the nonstrict-read-write strategy, and relies on the optimistic concurrency features of Hibernate. Note that nonstrict-read-write may deliver better performance if its impact on optimistic concurrency is acceptable.

For read-only caching, use the nonstrict-read-write strategy if the underlying database data may change, but slightly stale data is acceptable. If the underlying database data never changes, use the read-only strategy.

Query Cache

To cache query results, set the `hibernate.cache.use_query_cache` property to "true". Then whenever issuing a cacheable query, use `Query.setCacheable(true)` to enable caching of query results. As `org.hibernate.cache.QueryKey` instances in Hibernate may not be binary-comparable (due to non-deterministic serialization of unordered data members), use a size-limited Local or Replicated cache to store query results (this forces the use of the `hashCode()` or `equals()` methods to compare keys). The default query cache name is `org.hibernate.cache.StandardQueryCache` (unless a default region prefix is provided; in this case `[prefix]` is prepended to the cache name). Use the cache configuration file to map this cache name to a Local or Replicated topology, or explicitly provide an appropriately-mapped region name when querying.

Fault-Tolerance

The Hibernate L2 cache protocol supports full fault-tolerance during client or server failure. With the read-write cache concurrency strategy, Hibernate locks items out of the cache at the start of an update transaction, meaning that client-side failures simply result in uncached entities and an uncommitted transaction. Server-side failures are handled transparently by Coherence (dependent on the specified data backup count).

Deployment

When used with application servers that do not have a unified class loader, the Coherence Cache Provider must be deployed as part of the application so that it can use the application-specific class loader (required to serialize-deserialize objects).

Integrating an Oracle Coherence CacheFactory with Spring

Spring is a platform for building and running enterprise Java/J2EE applications. This chapter describes how to configure the Oracle Coherence cache to be available to applications that run on the Spring platform.

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

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

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

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

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

To configure Coherence to use the `SpringAwareCacheFactory`, the XML code in [Example 4-1](#) should be placed in the operational override file (`tangosol-coherence-override.xml`). By default, this specifies `coherence-cache-config.xml` as the cache configuration file and `application-context.xml` as the Spring configuration file.

Example 4-1 Configuring the Cache to use `SpringAwareCacheFactory`

```
<configurable-cache-factory-config>
  <class-name system-property="tangosol.coherence.cachefactory">
```

```

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

```

As an alternative to using the configuration file, the `SpringAwareCacheFactory` can be configured programmatically as illustrated in [Example 4-2](#):

Example 4-2 Configuring a `SpringAwareCacheFactory` Programmatically

```

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

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

```

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

Example 4-3 Defining a `SpringAwareCacheFactory` in an Application Context

```

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

```

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

Example 4-4 Configuring a `CacheFactory` in an Application Context

```

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

```

The application context may have a `CacheStore` configured as in [Example 4-5](#). Note that the `EntityCacheStore` is scoped as **prototype**. This scoping is specified because Coherence will manage the lifecycle of the bean when it is retrieved from Spring, just as if Coherence had instantiated the object using `new`.

Example 4-5 Configuring a `CacheStore` in an Application Context

```

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

```

```

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

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

```

Coherence can use the `entityCacheStore` bean as illustrated in [Example 4-6](#). By using the `init-param` element, setter injection can be used to set properties on the bean retrieved from Spring. The bean will have the method `setEntityName` invoked with the cache name before it is used by Coherence.

Example 4-6 Configuring Setter Injection to Set Properties on the Bean

```

<?xml version="1.0"?>

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

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

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

```

[Example 4-7](#) lists the source for `SpringAwareCacheFactory`. It requires Coherence 3.4.x and Spring 2.x.

Example 4-7 *SpringAwareCacheFactory.java*

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

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

import com.tangosol.util.ClassHelper;

import java.util.Iterator;

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

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

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

    /**
```

```

* Construct a default DefaultConfigurableCacheFactory using the
* default configuration file name.
*/
public SpringAwareCacheFactory()
{
    super();
}

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

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

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

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

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

    m_beanFactory = beanFactory;
}

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

/**
* Create an Object using the "class-scheme" element.
* <p/>
* In addition to the functionality provided by the super class,
* this will retrieve an object from the configured Spring BeanFactory
* for class names that use the following format:
* <pre>
* &lt;class-name&gt;spring-bean:sampleCacheStore&lt;/class-name&gt;
* </pre>
*

```

```

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

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

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

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

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

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

```

```

        {
            XmlElement xmlElement = (XmlElement) iter.next();

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

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

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

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

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

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

```



Integrating WebLogic Portal and Oracle Coherence

Oracle Coherence integrates closely with Oracle WebLogic Portal. Specifically, Coherence includes the following integration points:

- Coherence*Web for HTTP session state management
- P13N CacheProvider SPI implementation
- A blueprint for efficiently sharing data between WSRP-federated portals that uses Coherence and the WebLogic Portal Custom Data Transfer mechanism

P13N CacheProvider SPI Implementation

Internally, WebLogic Portal uses its own caching service to cache portal, personalization, and commerce data as described here:

http://download.oracle.com/docs/cd/E13155_01/wlp/docs103/javadoc/com/bea/p13n/cache/package-summary.html

WebLogic Portal 8.1.6 and later includes an SPI for the P13N caching service that can be implemented by third party cache vendors. Coherence includes a P13N CacheProvider SPI implementation that, when installed into a WebLogic Portal application, transparently manages cached P13N data without requiring code changes. Additionally, combining Coherence and WebLogic Portal gives you extreme flexibility in your choice of cache topologies.

For example, if you find that your Portal servers are bumping into the 4GB heap limit (on 32-bit JVMs) or are experiencing slow GC times, you can leverage a cache client/server topology to move serializable P13N state out of your Portal JVMs and into one or more dedicated Coherence cache servers. This reduces your Portal JVM heap size and GC times. Also, you can use the Coherence Management Framework to closely monitor statistics to better tune your P13N cache settings. Finally, the Coherence CacheProvider also allows your portlets to use Coherence caching service by using the standard P13N Cache API.

To install the Coherence P13N CacheProvider:

1. Copy the `coherence-wlp.jar` and `coherence.jar` libraries included from the `lib` directory of the Coherence installation to the `APP-INF/lib` directory of your WebLogic Portal application.
2. Configure the Coherence P13N CacheProvider as the default provider in the `p13n-cache-config.xml` file found in each of your WLP application's `META-INF` directory. Specifically, add the following line immediately before the first `<cache>` element.:

```
<default-provider-id>com.tangosol.coherence.weblogic</default-provider-id>
```

See the Javadoc for the `PortalCacheProvider` class for details on configuring the Coherence `CacheProvider` and Coherence caches used by the provider.

See the following document for a list of caches used by WebLogic Portal:

```
http://download.oracle.com/docs/cd/E13155\_01/wlp/docs103/caches/caches.html
```

Sharing Data Between WSRP-Federated Portals Using Coherence

The Web Services for Remote Portlets (WSRP) protocol was designed to support the federation of portals hosted by arbitrary portal servers and server clusters. Developers use WSRP to aggregate content and the user interface (UI) from various portlets hosted by other remote portals. By itself, though, WSRP does not address the challenge of implementing scalable, reliable, and high-performance federated portals that create, access, and manage the lifecycle of data shared by distributed portlets. Fortunately, WebLogic Portal provides an extension to the WSRP specification that, when coupled with Oracle Coherence, allows WSRP Consumers and Producers to create, view, modify, and control concurrent access to shared, scoped data in a scalable, reliable, and highly performant manner.

See the following document for complete details:

```
http://www.oracle.com/technology/pub/articles/dev2arch/2005/11/federated-portal-cache.html
```

Index

A

ApplicationContext interface, 4-1
application-context.xml file, 4-1

B

BeanFactory interface, 4-1
BeanFactoryAware interface, 4-2

C

CacheFactory interface, 4-1
CacheLoader interface, 3-1
cache-mapping element, 1-3
cache-name macro, 1-3, 3-3
CacheProvider interface, 5-2
CacheService interface, 3-5
CacheStore interface, 1-1, 2-3, 3-1
cachestore-scheme element, 1-3
class element, 1-2
class-scheme element, 4-1
Coherence
 configuring for JPA, 1-3
 configuring for Spring, 4-1
 configuring for TopLink Grid, 2-3
Coherence JPA API, 1-1
Coherence Read/Write configuration, 2-3
Coherence Shared L2 Cache configuration, 2-3
coherence-cache-config.xml file, 1-2, 1-3, 2-2, 2-3,
 3-2, 4-1
coherence-hibernate.jar file, 3-7
coherence.jar, 5-1
Coherence/Toplink Grid Read configuration, 2-3
coherence-wlp.jar, 5-1
ConfigurableCacheFactory interface, 4-1

E

EclipseLink JPA, 1-1

H

HibernateCacheLoader interface, 3-1
hibernate.cache.provider_class property, 3-6
hibernate.cache.region_prefix property, 3-7
HibernateCacheStore interface, 3-1

hibernate.cache.use_minimal_puts property, 3-7
hibernate.cache.use_query_cache property, 3-8
hibernate.cfg.xml file, 3-2
hibernate.hbm2ddl.auto property, 3-4

I

init-param element, 3-2, 4-3
init-params element, 3-3

J

Java Persistence API (JPA), 1-1
JPA
 2.0 specification, 1-1
 configuring, 1-2
 implementation, obtaining, 1-1
JpaCacheStore interface, 1-2, 1-3

O

org.hibernate.cache.QueryKey instances, 3-8
orm.xml file, 1-2, 2-3

P

P13N CacheProvider SPI, 5-1
p13n-cache-config.xml file, 5-1
param-type element, 3-2
persistence.xml file, 1-2, 1-3, 2-3
persistent classes, mapping, 1-2
PortalCacheProvider class, 5-2

R

remote portals, 5-2
RESOURCE_LOCAL transaction type, 1-2

S

Session cache, 3-6
SessionFactory interface, 3-5
setCacheMode method, 3-5
SpringAwareCacheFactory interface, 4-1, 4-2
StandardQueryCache, default Hibernate query cache
 name, 3-8

T

tangosol.coherence.hibernate.cacheconfig
property, 3-7
tangosol.coherence.hibernate.lockattemptmillis
property, 3-7
tangosol-coherence-override.xml file, 4-1
TopLink Grid, 2-1
Toplink Grid/Coherence
API, 2-3
toplink-grid.jar file, 2-1

W

Web Services for Remote Portlets (WSRP), 5-2
WSRP, 5-2