Oracle® Fusion Middleware

Developing Oracle Coherence Applications for Oracle WebLogic Server

12*c* (12.2.1)

E55229-01

October 2015

Documentation for developers and architects that describes how to develop, package, and deploy Oracle Coherence Applications for Oracle WebLogic Server.

ORACLE

Oracle Fusion Middleware Developing Oracle Coherence Applications for Oracle WebLogic Server, 12*c* (12.2.1)

E55229-01

Contents

Preface

This preface describes the document accessibility features and conventions that are used in this guide—*Developing Oracle Coherence Applications for Oracle WebLogic Server*.

Documentation Accessibility

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

Access to Oracle Support

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

Conventions

The following text conventions are used in this document:

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

1 Introduction and Roadmap

This chapter describes the contents and organization of this guide—*Developing Oracle Coherence Applications for Oracle WebLogic Server*.

This chapter includes the following sections:

1.1 Document Scope and Audience

The *Developing Oracle Coherence Applications for Oracle WebLogic Server* guide describes how to create, package, and deploy Coherence applications for WebLogic Server. The content is specific to using managed Coherence servers. This book does not detail the Coherence API. For details on using the Coherence API, see *Developing Applications with Oracle Coherence*.

This document is a resource for:

- Application developers and architects who want to develop and configure Coherence applications for WebLogic Server.
- Administrators who want to deploy Coherence applications to WebLogic Server.

1.2 Guide to This Document

- This chapter, Chapter 1, "Introduction and Roadmap," describes the organization of this document.
- Chapter 2, "Getting Started," describes Coherence applications and provides an overview of configuration, packaging and deployment, and creating Coherence applications.
- Chapter 3, "Creating Coherence Applications for WebLogic Server," describes how to package Coherence applications for Oracle WebLogic Server, how to create a Coherence application deployment descriptor, and how to access Coherence caches. This chapter also introduces Coherence*Web for session management and TopLink Grid for JPA entity caching.
- Chapter 4, "Deploying Coherence Applications in WebLogic Server," describes how to deploy Coherence applications to a Oracle WebLogic Server domain.

- Chapter A, "coherence-application.xml Deployment Descriptor Elements," provides a complete reference for the XML schema for the Coherence application deployment descriptor `coherence-application.xml`.

1.3 Related Documentation

For additional information, see the following Oracle Coherence and Oracle WebLogic Server documents:

Oracle Coherence

- *Developing Applications with Oracle Coherence*
- *Developing Remote Clients for Oracle Coherence*
- *Administering HTTP Session Management with Oracle Coherence*Web*
- *Oracle Coherence Integration Guide*
- *Managing Oracle Coherence*
- *Securing Oracle Coherence*

Oracle WebLogic Server

- *Developing Applications for Oracle WebLogic Server*
- *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*
- *Administering Clusters for Oracle WebLogic Server*
- *Oracle WebLogic Server Administration Console Online Help*

1.4 New and Changed Features in This Release

For a comprehensive listing of new Oracle WebLogic Server features introduced in this release, see *What's New in Oracle WebLogic Server*.

2 Getting Started

This chapter introduces Coherence applications for WebLogic Server and provides an overview of how to create and package Coherence Applications.

This chapter includes the following sections:

2.1 Introduction to Coherence Applications

Coherence is a distributed caching and in-memory data grid computing solution. Applications typically use Coherence to improve scalability, availability, and performance. Some common use cases for Coherence include data caching, HTTP session replication, and database cache store (such as a Java Persistence API (JPA) level-two cache).

Coherence is tightly integrated with WebLogic Server. The integration aligns the lifecycle of a Coherence cluster member with the lifecycle of a managed server: starting or stopping a managed server JVM starts and stops a Coherence cluster member. Managed servers that are cluster members are referred to as managed Coherence servers.

Like other Java EE modules, Coherence supports its own application module, which is called a Grid ARchive (GAR). The GAR contains the artifacts of a Coherence application and includes a deployment descriptor. A GAR is deployed and undeployed in the same way as Java EE modules and the application lifecycle is decoupled from the cluster service lifetime. Coherence applications and managed Coherence servers are not mandated by the JavaEE specification and are specific to WebLogic Server.

2.2 Typical Uses for Coherence

This section describes typical uses for Coherence in WebLogic Server. The WebLogic Server Coherence integration allows applications to easily use Coherence data caches and incorporate Coherence*Web for session management and TopLink Grid as an object-to-relational persistence framework.

Providing Application Data Caching and Data Grid Computing

Applications use Coherence for replicated and distributed caching. Applications access data caches either through resource injection or component-based JNDI lookup. The Oracle WebLogic Server Administration Console and Oracle WebLogic Scripting Tool are used to manage and configure Coherence clusters.

Using the Coherence integration enables you to create a data tier dedicated to caching application data and storing replicated session state. This is separate from the application tier—the WebLogic Server instances dedicated to running applications.

For more information, see Section 3, "Creating Coherence Applications for WebLogic Server."

Providing Session State Persistence and Management

Using Coherence*Web enables you to provide Coherence-based HTTP session state persistence to applications running on WebLogic Server. Coherence*Web enables HTTP session sharing and management across different Web applications, domains, and heterogeneous application servers. Session data can be stored in data caches outside of the application server, thus freeing application server heap space and enabling server restarts without losing session data.

For information on using Coherence*Web with WebLogic Server applications, see *Administering HTTP Session Management with Oracle Coherence*Web*.

Accessing Java Persistence API (JPA) Entities in the Data Cache

TopLink Grid's relational-to-object mapping capabilities allows you to store copies of database queries and result sets in Coherence data caches. With this feature, database access occurs only when no cached copy of the required data exists, or when the application performs a create, update, or delete operation that must be persisted to the database. This added optimization provides improved scalability and performance to the system.

TopLink Grid allows JPA Entity caching. This lets you support very large, shared grid caches that span cluster nodes. If the data cache does not contain the object, then the database is queried.

TopLink Grid also enables you to direct queries to Coherence. If the desired query result is not found in the cache, it can be read from the database and then placed in the cache, making it available for subsequent queries. The ability of Coherence to manage very large numbers of objects increases the likelihood of a result being found in the cache, as read operations in one cluster member become immediately available to others.

Writing JPA Entities to the database is also made possible by TopLink Grid. Applications can directly write JPA Entities to the database, then put them into the data cache (so that it reflects the database state), or put JPA Entities into the data cache, and then have the data cache write them to the database.

For more information, see Section 3.8, "Accessing and Retrieving Relational Data."

2.3 Understanding Coherence Application Configuration Files

A typical Coherence application that is deployed to WebLogic Server contains the following configuration files. For details on core Coherence configuration files, see *Developing Applications with Oracle Coherence*.

- Cache Configuration File – This file is used to specify the various types of caches which can be used within a Coherence cluster and is most often named

`coherence-cache-config.xml`. This file is commonly referred to as the cache configuration deployment descriptor. The schema for this file is the `coherence-cache-config.xsd` file. See *Developing Applications with Oracle Coherence* for a complete reference of the elements in this file.

- POF Configuration File – This file is used to specify custom data types when using Portable Object Format (POF) to serialize objects and is typically named `pof-config.xml`. This file is commonly referred to as the POF configuration deployment descriptor. The schema for this file is the `coherence-pof-config.xsd` file. See *Developing Applications with Oracle Coherence* for a complete reference of the elements in this file.
- Coherence Application Deployment Descriptor – This file is used to configure a Coherence application module that is deployed to a managed Coherence server. See Chapter A, "coherence-application.xml Deployment Descriptor Elements," for a complete reference of the elements in the descriptor.

2.4 Packaging and Deployment Overview

Coherence applications use a specific directory structure for deployment. You can deploy a Coherence application as a collection of files within this directory structure, known as exploded directory format, or as an archived file called a Grid ARchive (GAR) with a `.gar` extension. For details on packaging applications for deployment, see Section 3.1, "Packaging Coherence Applications." The directory structure is as follows:

```
MyCohApp/
  lib/
  META-INF/
    coherence-application.xml
```

A standalone GAR is deployed to all managed Coherence servers in a Coherence data tier. A GAR must also be packaged within a EAR and deployed to all managed Coherence servers that reside in an application tier. For details on deployment, see Chapter 4, "Deploying Coherence Applications in WebLogic Server."

2.5 Main Tasks for Creating Coherence Applications

The following steps summarize the procedure for creating a Coherence application both as a standalone GAR module and packaged as part of an enterprise application. The steps are detailed throughout this guide. For a complete Coherence application example, see the WebLogic Server Code Examples that are available with the WebLogic Server installation.

2.5.1 Task One: Create a Coherence Application Directory Structure

Create a staging directory that includes two subdirectories: `META-INF/` and `lib/`:

```
MyCohApp/
  lib/
  META-INF/
```

2.5.2 Task Two: Include the Coherence Application's Artifacts

Include the Coherence application artifacts in the staging directory. For details on creating Coherence applications, see Section 3, "Creating Coherence Applications for WebLogic Server."

1. Place the Coherence application class files in the root of the staging directory in the appropriate package structure. For example:

```
MyCohApp/
  com/
    myco/
      MyClass.class
      MySerializer.class
  lib/
  META-INF/
```

2. Place application dependency libraries in the `lib/` directory.

```
MyCohApp/
  com/
    myco/
      MyClass.class
      MySerializer.class
  lib/
      dependency.jar
  META-INF/
```

3. Include the `coherence-cache-config.xml` and the `pof-config.xml` file in the `META-INF/` directory:

```
MyCohApp/
  com/
    myco/
      MyClass.class
  lib/
      dependency.jar
  META-INF/
    coherence-cache-config.xml
    pof-config.xml
```

4. Create a `coherence-application.xml` file in the `META-INF` directory.

```
MyCohApp/
  com/
    myco/
      MyClass.class
  lib/
      dependency.jar
  META-INF/
    coherence-application.xml
    coherence-cache-config.xml
    pof-config.xml
```

5. Edit the `coherence-application.xml` file and include the location of the configuration files using the `<cache-configuration-ref>` and `<pof-configuration-ref>` elements, respectively:

```
<?xml version="1.0"?>
<coherence-application>
   xmlns="http://xmlns.oracle.com/coherence/coherence-application">
   <cache-configuration-ref>META-INF/coherence-cache-config.xml
```

```
   </cache-configuration-ref>
   <pof-configuration-ref>META-INF/pof-config.xml</pof-configuration-ref>
</coherence-application>
```

2.5.3 Task Three: Package the Coherence Application for Deployment

Package the Coherence application as a GAR file for deployment to a Coherence data tier. Then, package the GAR file within an EAR for deployment to a Coherence application tier. For details on deploying Coherence applications, see Chapter 4, "Deploying Coherence Applications in WebLogic Server."

1. From the command line, change directories to the root of the staging directory.
2. Use the Java `jar` command to compress the archive with a `.gar` extension. For example:

   ```
   jar cvf MyCohApp.gar *
   ```

3. Copy the GAR and package it within an enterprise application directory structure. For details on creating an EAR, see *Developing Applications for Oracle WebLogic Server*. For example:

   ```
   MyEAR/
     META-INF/
       application.xml
       weblogic-application.xml
     MyWAR.war
     MyEJB.jar
     MyCohApp.gar
   ```

   The `weblogic-application.xml` file must contain a module reference for the GAR. For example:

   ```
   <weblogic-application>
      <module>
         <name>MyCohApp</name>
         <type>GAR</type>
         <path>MyCohApp.gar</path>
      </module>
   </weblogic-application>
   ```

3

Creating Coherence Applications for WebLogic Server

This chapter describes how to use dependency injection and JNDI to access Coherence caches. Instructions are also provided for packaging Coherence applications as a Grid ARchive (GAR).

This chapter includes the following sections:

3.1 Packaging Coherence Applications

Coherence applications use a specific directory structure. You can deploy a Coherence application as a collection of files within this directory structure, known as exploded directory format, or as an archived file called a Grid ARchive (GAR) with a `.gar` extension.

A GAR module includes the artifacts that comprise a Coherence application. The `/META-INF` directory contains the deployment descriptor for the Coherence application (`cohererence-application.xml`). The presence of the deployment descriptor indicates a valid GAR. An additional subdirectory, the `/lib` directory, is used for storing dependency JAR files. Compiled Java classes that make up a Coherence application (entry processors, aggregators, filters, and so on) are placed in the root directory in the appropriate Java package structure.

A GAR module can also contain a cache configuration file (`coherence-cache-config.xml`) and a Portable Object Format (POF) serialization configuration file (`pof-config.xml`). The location of these files is defined within the

Coherence application deployment descriptor. Typically, the files are placed in the `/META-INF` directory; however, they can be located anywhere in the GAR relative to the root or even at a URL-accessible network location.

Note:

- If the configuration files are not found at runtime, then the default configuration files that are included in the `coherence.jar`, which is located in the system classpath, are used.
- If the configuration files are located in the root directory of the GAR, then they must not use the default file names; otherwise, the configuration files that are included in the `coherence.jar` file are found first and the configuration files in the GAR are never loaded.

The entire directory, once staged, is bundled into a GAR file using the `jar` command. GAR files are deployed as standalone archives to managed Coherence servers that are configured to store cached data.

Client applications that rely on the caches and resources in a GAR module must be packaged within an EAR that includes the dependent GAR. An EAR cannot contain multiple GAR modules. Multiple GAR modules must be merged into a single GAR. That is, a GAR must contain one application deployment descriptor, one cache configuration file, and one POF configuration file.

3.1.1 Directory Structure Example

The following is an example of a Coherence application directory structure, in which `myCohApp/` is the staging directory:

Example 3–1 Coherence Application Directory Structure

```
MyCohApp/
  lib/
  META-INF/
    coherence-application.xml
    coherence-cache-config.xml
    pof-config.xml
  com/myco/
    MyClass.class
```

3.1.2 Packaging a Grid Archive In an Enterprise Application

A GAR must be packaged in an EAR to be referenced by other JavaEE modules. For details on creating an EAR, see *Developing Applications for Oracle WebLogic Server*.

The following is an example of a Coherence application that is packaged within an EAR:

Example 3–2 Coherence Application Packaged in an EAR

```
MyEAR/
  META-INF/
    application.xml
    weblogic-application.xml
  MyWAR.war
```

```
MyEJB.jar
MyGAR.gar
```

Edit the `META-INF/weblogic-application.xml` descriptor and include a reference to the GAR using the `<module>` element. The reference is required so that the GAR is deployed when the EAR is deployed. For example:

```
<?xml version="1.0"?>
<weblogic-application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-application
   http://xmlns.oracle.com/weblogic/weblogic-application/1.6/
   weblogic-application.xsd"
   xmlns="http://xmlns.oracle.com/weblogic/weblogic-application">
   <module>
      <name>MyGAR</name>
      <type>GAR</type>
      <path>MyGAR.gar</path>
   </module>
</weblogic-application>
```

3.2 Creating a Coherence Application Deployment Descriptor

A GAR file must contain a Coherence application deployment descriptor (`cohererence-application.xml`) located in the `META-INF` directory. The presence of the deployment descriptor indicates a valid GAR. For a detailed reference of all the available elements in the descriptor, see Appendix A, "coherence-application.xml Deployment Descriptor Elements." The following is an example of a Coherence deployment descriptor that declares a cache configuration file and a POF configuration file that is located in the `META-INF` directory of the GAR.

```
<?xml version="1.0"?>
<coherence-application
   xmlns="http://xmlns.oracle.com/coherence/coherence-application">
   <cache-configuration-ref>META-INF/coherence-cache-config.xml
   </cache-configuration-ref>
   <pof-configuration-ref>META-INF/pof-config.xml</pof-configuration-ref>
</coherence-application>
```

3.3 Using JNDI to Override Configuration

Coherence provides the ability to override any XML element value in a configuration file using JNDI properties. The use of JNDI properties allows a single version of a configuration file to be used for deployment and then altered as required at runtime.

To define a JNDI property, add an `override-property` attribute to an XML element with a value set to a JNDI context. The following example defines a JNDI property with a `cache-config/MyGar` context for the `<cache-configuration-ref>` element in a `coherence-application.xml` deployment descriptor. The JNDI property is used at runtime to override the cache configuration reference and specify a different cache configuration file. The JNDI context of `cache-config` is a well known context used and registered by a managed Coherence server.

```
<?xml version="1.0"?>
<coherence-application
   xmlns="http://xmlns.oracle.com/coherence/coherence-application">
   <cache-configuration-ref override-property="cache-config/MyGar">
      META-INF/coherence-cache-config.xml</cache-configuration-ref>
   <pof-configuration-ref>META-INF/pof-config.xml</pof-configuration-ref>
```

```
</coherence-application>
```

3.4 Defining a Data Cache

Data caches are defined in a `coherence-cache-config.xml` file that is packaged in a GAR file. For details on Coherence caches and their configuration, see *Developing Applications with Oracle Coherence*.

The following example creates a distributed cache named `myCache`. As an alternative, a cache name may be defined with the asterisk (`*`) wildcard, which allows an application to use the distributed cache by specifying any name.

```
<?xml version="1.0" encoding="windows-1252"?>
<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
   coherence-cache-config.xsd">

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

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

Coherence does not support the use of a replicated cache scheme if a GAR module is used in multiple EAR modules (packaged either individually or as a shared GAR) on a managed Coherence server. An alternative is to use a near cache instead of a replicated cache.

3.5 Accessing a Data Cache

Applications use the Coherence `NamedCache` API to interact with a Coherence cache. This object holds resources that are shared among members of a Coherence cluster. An application can obtain a `NamedCache` object either by dependency injection or by using a JNDI lookup.

To Obtain a Cache by Dependency Injection

An `@Resource` annotation can be used in a servlet or an EJB to dynamically inject the `NamedCache`. This annotation cannot be used in a JSP. The name of the cache used in the annotation must be defined in the application's `coherence-cache-config.xml` file.

Example 3–3 illustrates using dependency injection to get a cache named `myCache`. For details on JavaEE annotations and dependency injection, see *Developing Applications for Oracle WebLogic Server*.

Example 3–3 Obtaining a Cache Resource by Dependency Injection

```
...
@Resource(mappedName="MyCache")
com.tangosol.net.NamedCache nc;
...
```

To Obtain the NamedCache by JNDI Lookup

A component-scoped JNDI tree can be used in EJBs, servlets, or JSPs to get a `NamedCache` reference.

To use a JNDI lookup, define a `resource-ref` of type `com.tangosol.net.NamedCache` in either the `web.xml` or `ejb-jar.xml` file. Example 3–4 illustrates a `<resource-ref>` element that identifies `myCache` as the `NamedCache`. For details on using JNDI in Oracle WebLogic Server, see *Developing JNDI Applications for Oracle WebLogic Server*.

> **Note:** The `<res-auth>` and `<res-sharing-scope>` elements do not appear in the example. The `<res-auth>` element is ignored because currently no resource sign-on is performed to access data caches. The `<res-sharing-scope>` element is ignored because data caches are sharable by default and this behavior cannot be overridden.

Example 3–4 Defining a NamedCache as resource-ref for JNDI Lookup

```
...
<resource-ref>
  <res-ref-name>coherence/myCache</res-ref-name>
  <res-type>com.tangosol.net.NamedCache</res-type>
  <mapped-name>MyCache</mapped-name>
</resource-ref>
...
```

The following example performs a JNDI lookup to get a `NamedCache` reference that is defined in Example 3–4:

```
try {
   Context ctx = new InitialContext();
   cache = (NamedCache) ctx.lookup("java:comp/env/coherence/myCache");
   cache.put(key, value);
}
catch (NamingException ne)
```

3.6 Using the Coherence API

Coherence provides a full-featured API for interacting with a cache and for performing data grid operations. For details on using the API to develop applications, see *Developing Applications with Oracle Coherence*. For a reference of the Coherence API, see *Java API Reference for Oracle Coherence*.

Some of the features include:

- basic `get`, `put`, and `putAll` operations
- querying a cache
- processing data in a cache using entry processors and aggregators
- event notifications

Using POF for Serialization

Objects that are placed in a cache must be serializable. The Portable Object Format (also referred to as POF) is a language agnostic binary format. POF is designed to be efficient in both space and time and is the recommended serialization option in Coherence. For details on using POF in your applications, see *Developing Applications with Oracle Coherence*.

3.7 Using a Coherence Application Lifecycle Listener

Coherence applications support the use of an application lifecycle listener. The listener allows custom processing to occur before and after the creation and destruction of Coherence caches and clustered services. The listener class must implement the `com.tangosol.application.LifecycleListener` interface. See the *Oracle Coherence Java API Reference* for details on the interface.

Override the following methods provided in the `LifecycleListener` interface and add any required functionality:

- `preStart(Context)` – called before the application is activated
- `postStart(Context)` – called after the application is started
- `preStop(Context)` – called before the application stops its services
- `postStop(Context)` – called after the application is stopped

To use an application lifecycle listener class, declare the fully qualified name of the class within the `<application-lifecycle-listener>` element in the `coherence-application.xml` deployment descriptor and include the class in the `/lib` directory of the GAR. The following is an example of declaring an application lifecycle listener class that is named `MyAppLifecycleListener`.

```
<?xml version="1.0"?>
<coherence-application
   xmlns="http://xmlns.oracle.com/coherence/coherence-application">
   <cache-configuration-ref>META-INF/coherence-cache-config.xml
   </cache-configuration-ref>
   <pof-configuration-ref>META-INF/pof-config.xml</pof-configuration-ref>
   <application-lifecycle-listener>
      <class-name>package.MyAppLifecycleListener</class-name>
   </application-lifecycle-listener>
</coherence-application>
```

3.8 Accessing and Retrieving Relational Data

TopLink Grid is an integration between TopLink and Coherence. Developers use the standard Java Persistence API (JPA) in their applications and take advantage of the scalability of Coherence. TopLink Grid is used to store some or all of a domain model in the Coherence data grid and can also be used as a level 2 cache. For detailed information on using and configuring TopLink Grid, see *Integrating Oracle Coherence*.

The TopLink Grid library (toplink-grid.jar) is located in the *`INSTALL_HOME`*`\oracle_common\modules\oracle.toplink_`*`version`* directory. The library is included as part of the WegLogic Server system classpath and does not need to be included as part of a GAR or an application (EAR) module.

3.8.1 Specifying the Eclipse Persistence Provider

The `persistence.xml` file is the JPA persistence descriptor file. This is where you configure the persistence unit, the persistence provider, and any vendor-specific extensions that this reference describes.

In the file, the `provider` element specifies the name of the vendor's persistence provider class. When working with Oracle TopLink, enter `org.eclipse.persistence.jpa.PersistenceProvider` as the value for this element.

Example 3–5 Sample persistence.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="JPA" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>com.oracle.handson.Employees</class>
   ...
```

3.8.2 Packaging a Persistence Unit

JPA persistence units (entity classes and resources, including the `persistence.xml` file) that use TopLink Grid should be packaged as a JAR and included within an EAR module. For example, the JAR can be placed in the `APP-INF/lib` directory or root of the EAR module that contains the GAR module.

Example 3–6 Coherence and JPA Application Packaged in an EAR

```
MyEAR/
  APP-INF
    /lib/MyPersistenceUnit.jar
  META-INF/
    application.xml
    weblogic-application.xml
  MyWAR.war
  MyEJB.jar
  MyGAR.gar
```

> **Note:** Persistence files cannot be placed in a WAR file. Any persistence files placed in the WAR file will not be found at runtime. Coherence only has access to the application classloader.

Unlike the application tier, the Coherence data tier only requires the deployment of a GAR. For the data tier, include the persistence unit JAR within the /lib directory of the GAR. For example:

```
MyCohApp/
  lib/
    MyPersistence.jar
  META-INF/
    coherence-application.xml
    coherence-cache-config.xml
    pof-config.xml
  com/myco/
```

```
MyClass.class
```

3.9 Using Coherence for Session Management

Web applications can chose to use Coherence for storing and replicating session state. The session management features of Coherence are implemented by the Coherence*Web component. For details on setting up, configuring, and using Coherence*Web in WebLogic Server, see *Administering HTTP Session Management with Oracle Coherence*Web*.

3.10 Creating Extend Clients in WebLogic Server

Client applications can chose to use Coherence*Extend to interact with Coherence caches without becoming members of a Coherence cluster. Client applications connect to managed Coherence proxy servers and are unaware that cache and invocation service requests are being executed remotely. Remote clients may be deployed within a WebLogic Server domain or may be external to WebLogic Server. For details on setting up a Coherence proxy server tier in WebLogic Server to allow remote connections, see *Administering Clusters for Oracle WebLogic Server*. For details on creating Coherence*Extend client applications, see *Developing Remote Clients for Oracle Coherence*.

3.11 Using a JCache Cache in WebLogic Server

Applications that are deployed to a managed Coherence container can use the JCache API and JCache provider that is implemented by Coherence. For details on using JCache with Coherence, see *Developing Applications with Oracle Coherence*.

To use JCache:

1. Add the *COHERENCE_HOME*`/lib/cache-api.jar` and *COHERENCE_HOME*`/lib/coherence-jcache.jar` libraries to the `/lib` directory in a GAR file.
2. Edit the cache configuration file that is referenced in the `coherence-application.xml` file to include either the JCache namespace or JCacheExtend namespace. For example

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

3. Use the JCahe API to create and use a JCache cache. For example within a servlet:

```
static private Cache<ContactId, Contact> getCache(String cacheName)
   {
   CachingProvider provider = Caching.getCachingProvider();
   CacheManager mgr = Caching.getCachingProvider().getCacheManager();

   Cache<ContactId, Contact> cache = null;

   try {
      cache = mgr.getCache(cacheName, ContactId.class, Contact.class);
   }
```

```
   catch (IllegalStateException e) {

      if (cache == null) {
      PartitionedCacheConfiguration config = new
         PartitionedCacheConfiguration<ContactId, Contact>();
      config.setTypes(ContactId.class, Contact.class);
      config.setStatisticsEnabled(true); config.setManagementEnabled(true);
      cache = mgr.createCache(cacheName, config);
      }
   return cache;
}
```

4

Deploying Coherence Applications in WebLogic Server

This chapter provides instructions for deploying Coherence applications to managed Coherence servers in a WebLogic Server domain. Coherence applications must be packaged as a Grid ARchive (GAR) for deployment. For details on creating a GAR, see Section 3.1, "Packaging Coherence Applications."

This chapter includes the following sections:

4.1 Understanding Coherence Deployment Tiers

Coherence is setup in tiers within a WebLogic Server domain. The tiers often include: a data tier for storing data; an application tier for consuming cached data; and a proxy tier for allowing remote clients (non cluster members) to use a cluster. The use of a dedicated storage tier that is separate from the application tier and proxy tier is a best practice that ensures optimal performance of a Coherence cluster.

The deployment tiers contain managed servers that are part of a Coherence cluster. Managed servers that are part of a Coherence cluster are referred to as managed Coherence servers. Coherence tiers are typically associated with respective WebLogic Server clusters. The use of WebLogic Server clusters simplifies the deployment of an application and the management of the deployment topology, especially in large clusters. However, managed Coherence servers in each tier can be individually managed as required.

During development and simple testing, setting up Coherence deployment tiers may be impractical. In this case, a Coherence application can be deployed to a single managed server and a single-server cluster is automatically created using default cluster settings.

For details on creating Coherence data tiers and setting up Coherence clusters for deployment, see *Administering Clusters for Oracle WebLogic Server*.

4.2 Deploying Applications to Managed Coherence Servers

Coherence application GAR modules get deployed the same way as JavaEE modules and can be deployed using any WebLogic Server deployment tool: the WebLogic Server Administration Console, the Oracle WebLogic Scripting Tool (WLST), the WebLogic Server `Deployer` class, and the WebLogic Server `<wldeploy>` ANT target. For details on using the WebLogic Server Administration Console, see *Oracle WebLogic Server Administration Console Online Help*. For details on using WLST, see *Understanding the WebLogic Scripting Tool*. For detailed instructions on the other deployment options, see *Deploying Applications to Oracle WebLogic Server*.

> **Note:** Production redeployment of an EAR containing a GAR is only supported for storage-disabled cluster clients. In addition, any changes to the code in the GAR must be backward compatible with the existing deployment. For example, entity classes that are changing must implement the `Evolvable` interface. For details on production redeployment, see *Deploying Applications to Oracle WebLogic Server*.

GAR modules should be deployed as standalone modules and also as part of an EAR. The following list describes how GAR modules are deployed in a WebLogic Server domain that uses Coherence tiers. For details on creating Coherence deployment tiers, see *Administering Clusters for Oracle WebLogic Server*.

- Data Tier – Deploy a standalone GAR to each managed Coherence server of the data tier. If the data tier is setup as a WebLogic Server cluster, deploy the GAR to the cluster and have the module copied to each managed Coherence server.
- Application Tier – Deploy the EAR that contains both the client implementation (Web Application, EJB, and so on) and the GAR to each managed Coherence server in the cluster. If the application tier is setup as a WebLogic Server cluster, deploy the EAR to the cluster and have the module copied to each managed Coherence server.
- Proxy Tier – Deploy the standalone GAR to each managed Coherence server of the proxy tier. The cache configuration file packaged in the GAR must include a proxy service definition. If the application tier is setup as a WebLogic Server cluster, deploy the GAR to the cluster and have the module copied to each managed Coherence server.
- Extend Client Tier – Deploy the EAR that contains the extend client implementation (Web Application, EJB, and so on) as well as the GAR to each managed server that hosts the extend client. The client's cache configuration file must include a remote cache service definition that defines the address of a proxy server. If the extend client tier is setup as a WebLogic Server cluster, deploy the EAR to the cluster and the WebLogic deployment infrastructure copies the module to each managed Coherence server.

4.3 Deploying Coherence Applications as Shared Libraries

A standalone GAR can be deployed as a shared library and referenced by multiple EAR files. For general information about shared libraries and their deployment, see "Creating Shared Java EE Libraries and Optional Packages" in *Developing Applications for Oracle WebLogic Server*.

To use the GAR at runtime, the `weblogic-application.xml` deployment descriptor in an EAR must contain a reference to the GAR. For example:

```
<weblogic-application>
   <library-ref>
      <library-name>ExampleGAR</library-name>
   </library-ref>
</weblogic-application>
```

The above configuration works in single-tier domain where both the application tier and data tier are on a single managed Coherence server. However, in a multi-tiered domain, additional configuration is required to ensure that a GAR that is deployed as a shared library results in storage-enabled members starting as expected.

To deploy a GAR as shared library in multi-tiered domain:

1. Edit the cache configuration file in the GAR and explicitly set the `<scope-name>` element to the GAR name. For details about configuring the scope name, see *Administering Oracle Coherence*.

   For example, if the GAR is named `ExampleGAR.gar`, the `<scope-name>` element is defined as follows:

   ```
   <cache-config>
      <defaults>
         <scope-name>ExampleGAR</scope-name>
      </defaults>
      ...
   ```

2. Deploy the GAR to the application (storage-disabled) tier as a shared library and specify the GAR name as the application name. For example, if the GAR is named `ExampleGAR.gar`, then the GAR name is specified as `ExampleGAR`.
3. Edit the `weblogic-application.xml` deployment descriptor in the EAR and include a reference for the GAR. For example:

   ```
   <weblogic-application>
      <library-ref>
         <library-name>ExampleGAR</library-name>
      </library-ref>
   </weblogic-application>
   ```

4. Deploy the EAR to the application tier.
5. Deploy the same GAR to the data (storage-enabled) tier and modify the name. For example, `ExampleGAR-DataTier`. If a name is not specified, a `-1` is appended to the deployment name because the GAR already exists as a shared library.
6. After the deployment completes, the GARs that are deployed to both tiers (for example, `ExampleGAR` and `ExampleGAR-DataTier`) join the same services and client request processing and data storage are separated as expected.

4.4 Referencing Shared Libraries from a Coherence Application

A GAR module can use shared libraries that are deployed to WebLogic Server. To use a shared library, reference the shared library within a `<library-ref>` node in the `weblogic-coh-app.xml` deployment descriptor and package the deployment descriptor in the `/META-INF` directory of the GAR module. For a detailed reference of the available elements in the descriptor, see Appendix B, "weblogic-coh-app.xml Deployment Descriptor Elements." For example:

```
<?xml version="1.0"?>
<weblogic-coh-app
   xmlns:wls="http://xmlns.oracle.com/weblogic/weblogic-coh-app">
```

```
   <library-ref>
      <library-name>mySharedLibrary</library-name>
      <specification-version>2.0</specification-version>
      <implementation-version>8.1beta</implementation-version>
      <exact-match>false</exact-match>
   </library-ref>
</weblogic-coh-app>
```

4.5 Performing a Rolling Redeploy

GAR modules that are targeted to a WebLogic Server cluster are simultaneously redeployed to all managed Coherence servers. There are no provisions that guard against data loss by ensuring that the data partitions on a server are fully transmitted to another server during redeployment.

A rolling redeploy is a technique for updating a GAR across a WebLogic Server cluster by individually redeploying the GAR on each managed Coherence server and cycling through all servers. A rolling redeploy allows cached data to be redistributed while the GAR is redeployed. Cached data is otherwise lost if a GAR is redeployed to all cache servers simultaneously.

Note: Always check the `StatusHA` metric on the partitioned service between server-targeted deployments to ensure `MACHINE_SAFE` status. For details on this metric, see *Managing Oracle Coherence*.

To perform a rolling redeploy, a GAR must be deployed using the `specifiedtargetsonly` option, which ensures that subsequent updates to the GAR results in a deployment on the current target and not on all targets that contain the GAR. The `specifiedtargetsonly` option is not available through the WebLogic Server Administration Console and must be specified using either WLST, `weblogic.Deployer`, or the `<wldeploy>` ANT target.

The full path and name to the GAR file must match exactly the path and name that was used to originally deploy the GAR. If a different path or name is used, then a `-1`, `-2`, or, `-1` and `-2` is appended to the GAR name and the rolling redeploy will not work correctly. In addition, if the GAR was originally deployed using the `upload=true` option, then you must redeploy using the `upload=true` option; otherwise, the rolling redeploy will not work correctly.

For a complete example (including a WLST script) of redeploying Coherence applications (including ensuring `MACHINE_SAFE` status), see the Coherence examples that are part of the WebLogic Server examples. The examples are available by performing a custom WebLogic Server installation and selecting to install the Server Examples. For details, see *Understanding Oracle WebLogic Server*.

WLST

```
deploy('MyCohApp', '/myapps/MyCohApp.gar', 'server1', specifiedTargetsOnly='true')
```

<wldeploy> ANT Target

```
<wldeploy
   user="${admin.username}"
   password="${admin.password}"
   adminurl="t3://${admin.host}:${admin.port}"
   debug="false"
```

```
   action="deploy"
   name="Coherence GAR"
   source="${gar.filename}"
   targets="ms3"
   specifiedtargetsonly="true"
   failonerror="true"/>
```

weblogic.Deployer

```
java weblogic.Deployer -adminurl t3://localhost:7001 -username username -password
password -targets ms3 -deploy -name MyCohApp /myapps/MyCohApp.gar
-specifiedtargetsonly
```

4.6 Loading Coherence From the Application Classloader

The Coherence library (`coherence.jar`) is included in the system classpath of WebLogic Server. It is a best practice to always use this library and not include the `coherence.jar` library within the `/lib` directory of a Web application. For advanced use cases that include the `coherence.jar` library in a Web application, the Coherence resources must be defined in the `weblogic.xml` file using the `<prefer-application-packages>` and `<prefer-application-resources>` elements. For example:

```
<container-descriptor>
   <prefer-application-packages>
     <package-name>com.tangosol.*</package-name>
     <package-name>com.oracle.common.**</package-name>
   </prefer-application-packages>

   <prefer-application-resources>
     <resource-name>com.tangosol.*</resource-name>
     <resource-name>com.oracle.common.*</resource-name>
     <resource-name>coherence-*.xml</resource-name>
     <resource-name>coherence-*.xsd</resource-name>
     <resource-name>tangosol-*.xml</resource-name>
     <resource-name>tangosol.properties</resource-name>
     <resource-name>tangosol.cer</resource-name>
     <resource-name>tangosol.dat</resource-name>
     <resource-name>internal-txn-cache-config.xml</resource-name>
     <resource-name>txn-pof-config.xml</resource-name>
     <resource-name>pof-config.xml</resource-name>
     <resource-name>management-config.xml</resource-name>
     <resource-name>processor-dictionary.xml</resource-name>
     <resource-name>reports/*</resource-name>
     </prefer-application-resources>
 </container-descriptor>
```

4.7 Securing Coherence Applications in WebLogic Server

For details on securing Coherence applications that are deployed to managed Coherence servers, see *Securing Oracle Coherence*.

A

coherence-application.xml Deployment Descriptor Elements

This appendix provides a complete reference for the elements in the Coherence application deployment descriptor `coherence-application.xml`.

This appendix includes the following sections:

A.1 coherence-application.xml Namespace Declaration and Schema Location

The correct text for the namespace declaration and schema location for the `coherence-application.xml` file is as follows.

```
<coherence-application
   xmlns="http://xmlns.oracle.com/coherence/coherence-application">
```

A.2 application-lifecycle-listener

The `application-lifecycle-listener` element specifies the fully qualified name of a class that implements the `com.tangosol.application.LifeCycleListener` interface. The class allows custom processing before and after the creation and destruction of Coherence cache and clustered services.

The following table describes the elements you can define within an `application-lifecycle-listener` element.

Table A–1 application-lifecycle-listener Elements

Element	Required/ Optional	Description
`class-name`	Required	Specifies the fully qualified name of a class that implements the `com.tangosol.application.LifeCycleListener` interface.
`init-params`	Optional	Specifies an initialization parameter that is required by the implementation. Any number of `init-params` elements may be defined.

A.3 cache-configuration-ref

The `cache-configuration-ref` element specifies the name and location of a Coherence cache configuration file. The location of the file is relative to the root directory within a Coherence Grid Archive (GAR). A URL may also be specified. If the file is not found, or if this element is not specified, then the predefined cache configuration file (`coherence-cache-config.xml`) that is located in the `coherence.jar` library on the system classpath is used by default.

Note: If the configuration file is located in the root directory of the GAR, then it must not use the default file name (`coherence-cache-config.xml`); otherwise, the configuration file that is included in the `coherence.jar` file which is located in the system classpath is found first and the configuration file in the GAR is never loaded. An alternative to renaming the file is to place the configuration file in the `META-INF` directory of the GAR.

A.4 coherence-application

The `coherence-application` element is the root element of the Coherence application deployment descriptor.

The following table describes the elements you can define within a `coherence-application` element.

Table A–2 coherence-application Elements

Element	Required/ Optional	Description
`cache-configuration-ref`	Optional	Specifies the name and location of the Coherence cache configuration file.
`pof-configuration-ref`	Optional	Specifies the name and location of the Coherence Portable Object Format (POF) configuration file.
`application-lifecycle-listener`	Optional	Specifies the fully qualified name of a class that implements the `com.tangosol.application.LifeCycleListener` interface.
`configurable-cache-factory-config`	Optional	Specifies the fully qualified name of a class that implements the `com.tangosol.net.ConfigurableCacheFactory` interface.

A.5 configurable-cache-factory-config

The `configurable-cache-factory-config` element specifies the fully qualified name of a class that implements the `com.tangosol.net.ConfigurableCacheFactory` interface. The default implementation is the `com.tangosol.net.ExtensibleConfigurableCacheFactory` class.

Using a custom `ConfigurableCacheFactory` implementation is an advanced use case and is typically used to allow applications that are scoped by different class loaders to use separate cache configuration files.

The following table describes the elements you can define within a `configurable-cache-factory-config` element.

Table A–3 configurable-cache-factory-config Elements

Element	Required/ Optional	Description
`class-name`	Required	Specifies the fully qualified name of a class that implements the `com.tangosol.net.ConfigurableCacheFactory` interface.
`init-params`	Optional	Specifies an initialization parameter that is required by the implementation. Any number of `init-params` elements may be defined.

A.6 init-params

The `init-params` element specifies an initialization parameter. Any number of `init-params` elements may be defined.

The following table describes the elements you can define within an `init-params` element.

Table A–4 init-params Elements

Element	Required/ Optional	Description
param-type	Optional	Specifies the Java type of the initialization parameter.The following standard types are supported: ■ `java.lang.String` (`string`) ■ `java.lang.Boolean` (`boolean`) ■ `java.lang.Integer` (`int`) ■ `java.lang.Long` (`long`) ■ `java.lang.Double` (`double`) ■ `java.math.BigDecimal` ■ `java.io.File` ■ `java.sql.Date` ■ `java.sql.Time` ■ `java.sql.Timestamp`

Table A–4 (Cont.) init-params Elements

Element	Required/ Optional	Description

`param-value`	Optional	Specifies the value of the initialization parameter. The value is in the format specific to the Java type of the parameter.
		For example:
		`<init-params>`
		`  <param-type>java.lang.String</param-type>`
		`  <param-value>EmployeeTable</param-value>`
		`</init-params>`

A.7 pof-configuration-ref

The `pof-configuration-ref` element specifies the name and location of a Coherence POF configuration file. The location of the file is relative to the root directory within a Coherence Grid Archive (GAR). A URL may also be specified. If the file is not found, or if this element is not specified, then the predefined POF configuration file (`pof-config.xml`) that is located in the `coherence.jar` library on the system classpath is used by default.

Note: If the configuration file is located in the root directory of the GAR, then it must not use the default file name (`pof-config.xml`); otherwise, the configuration file that is included in the `coherence.jar` file which is located in the system classpath is found first and the configuration file in the GAR is never loaded. An alternative to renaming the file is to place the configuration file in the `META-INF` directory of the GAR.

B

weblogic-coh-app.xml Deployment Descriptor Elements

This appendix provides a complete reference for the elements in the WebLogic Coherence application deployment descriptor `weblogic-coh-app.xml`.

This appendix includes the following sections:

B.1 weblogic-coh-app.xml Namespace Declaration and Schema Location

The correct text for the namespace declaration and schema location for the `weblogic-coh-app.xml` file is as follows.

```
<weglogic-coh-app
   xmlns:wls="http://xmlns.oracle.com/weblogic/weblogic-coh-app">
```

B.2 weblogic-coh-app

The `weblogic-coh-app` element is the root element of the WebLogic Coherence application deployment descriptor.

The following table describes the elements you can define within a `weblogic-coh-app` element.

Table B–1 weblogic-coh-app Elements

Element	Required/ Optional	Description
`description`	Optional	Specifies a description.
`library-ref`	Optional	Specifies a shared library module that is intended to be used as a library in a Coherence application.

B.3 library-ref

The `library-ref` element specifies a shared library module that is intended to be used as a library in a Coherence application.

The following table describes the elements you can define within a `library-ref` element.

Table B–2 library-ref Elements

Element	Required/ Optional	Description
`library-name`	Required	Specifies the name of the referenced shared library.
`specification`	Optional	Specifies the minimum specification-version required.
`implementation-version`	Optional	Specifies the minimum implementation-version required.
`exact-match`	Optional	Specifies whether there must be an exact match between the specification and implementation version that is specified and that of the referenced library. Default value is `false`.