Oracle® Fusion Middleware

Developing Oracle Coherence Applications for Oracle WebLogic Server



26 (26.1.0.0.0)
G56901-01
November 2026



Oracle Fusion Middleware Developing Oracle Coherence Applications for Oracle WebLogic Server, 26 (26.1.0.0.0)

G56901-01

Copyright © 2007, 2026, Oracle and/or its affiliates.

Primary Author: Oracle Corporation

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 is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware 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 that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about 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 unless otherwise set forth in an applicable agreement between you and Oracle. 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, except as set forth in an applicable agreement between you and Oracle.

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.

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

Contents

Preface

Audience i
Documentation Accessibility i
Diversity and Inclusion i
Related Documents ii
Conventions ii

1 Getting Started

Introduction to Coherence Applications 1
Typical Uses for Coherence 2
Understanding Coherence Application Configuration Files 2
Packaging and Deployment Overview 3
Main Tasks for Creating Coherence Applications 3
- Task One: Create a Coherence Application Directory Structure 3
- Task Two: Include the Coherence Application's Artifacts 3
- Task Three: Package the Coherence Application for Deployment 4

Preparing a Release-Matched WebLogic Deployment 5

2 Creating Coherence Applications for WebLogic Server

Packaging Coherence Applications 1
- Directory Structure Example 2
- Packaging a Grid Archive In an Enterprise Application 3
- Creating a Coherence Project from a Maven Archetype 3

Creating a Coherence Application Deployment Descriptor 7
Using JNDI to Override Configuration 7
Defining a Data Cache 7
Accessing a Data Cache 8
Using the Coherence API 9
Using a Coherence Application Lifecycle Listener 10
Using Coherence for Session Management 10
Creating Extend Clients in WebLogic Server 10
Using a JCache Cache in WebLogic Server 11

Copyright © 2007, 2026, Oracle and/or its affiliates.

Packaging Application Trust Metadata in a GAR 12

3 Deploying Coherence Applications in WebLogic Server

Understanding Coherence Deployment Tiers 1
Deploying Applications to Managed Coherence Servers 1
Deploying Coherence Applications as Shared Libraries 2
Referencing Shared Libraries from a Coherence Application 3
Performing a Rolling Redeploy 4
Securing Coherence Applications in WebLogic Server 5
Verifying a Rolling GAR Redeployment 5

A coherence-application.xml Deployment Descriptor Elements

coherence-application.xml Namespace Declaration and Schema Location A-1
application-lifecycle-listener A-2
cache-configuration-ref A-2
coherence-application A-2
configurable-cache-factory-config A-3
init-params A-3
pof-configuration-ref A-4

B weblogic-coh-app.xml Deployment Descriptor Elements

weblogic-coh-app.xml Namespace Declaration and Schema Location B-1
weblogic-coh-app B-1
library-ref B-2

Copyright © 2007, 2026, Oracle and/or its affiliates.

Preface

This document describes how to create, package, and deploy Coherence applications for WebLogic Server. The content is specific to using managed Coherence servers.

- Audience
- Documentation Accessibility
- Diversity and Inclusion
- Related Documents
- Conventions

Audience

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.

This book does not detail the Coherence API. See Developing Applications with Oracle Coherence for details on using the Coherence API.

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 customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Related Documents

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 Remote Console Online Help*

New and Changed WebLogic Server Features

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

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.

Copyright © 2007, 2026, Oracle and/or its affiliates.

1

Getting Started

This chapter introduces Coherence applications for WebLogic Server and provides an overview of creating and packaging Coherence Applications.

This chapter includes the following sections:

- 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.
- Typical Uses for Coherence
  Coherence is used to provide solutions such as application data caching, data grid computing, session state persistence and management and access Java Persistence API (JPA) entities in the data cache.
- Understanding Coherence Application Configuration Files
  Coherence applications contain the Cache Configuration, POF Configuration, and Coherence Application Deployment Descriptor files used to configure caches, data types and so on.
- Packaging and Deployment Overview
  Coherence applications use a specific directory structure for deployment. You can deploy a Coherence application either in an exploded directory format, or as an archived file.
- Main Tasks for Creating Coherence Applications
  A Coherence application can be created both as a standalone GAR module and packaged as part of an enterprise application.
- Preparing a Release-Matched WebLogic Deployment
  Match the domain runtime and build dependencies while retaining container ownership of Coherence resources.

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.

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. Some common use cases for Coherence include data caching, grid computing, and HTTP session replication.

Like other Jakarta 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 Jakarta EE modules and the application lifecycle is decoupled from the cluster service lifetime. Coherence applications and managed Coherence servers are not mandated by the Jakarta EE specification and are specific to WebLogic Server.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Typical Uses for Coherence

Coherence is used to provide solutions such as application data caching, data grid computing, session state persistence and management and access Java Persistence API (JPA) entities in the data cache.

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.

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 WebLogic Remote 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.

See 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.

See Introduction to Coherence*Web.

Understanding Coherence Application Configuration Files

Coherence applications contain the Cache Configuration, POF Configuration, and Coherence Application Deployment Descriptor files used to configure caches, data types and so on.

A typical Coherence application that is deployed to WebLogic Server contains the following 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 coherence-application.xml Deployment Descriptor Elements, for a complete reference of the elements in the descriptor.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Packaging and Deployment Overview

Coherence applications use a specific directory structure for deployment. You can deploy a Coherence application either in an exploded directory format, or as an archived file.

A Coherence application deployed as a collection of files within a specific directory structure is known as exploded directory format. A Coherence application deployed as an archived file is called a Grid ARchive (GAR) with a `.gar` extension. See 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. See Deploying Coherence Applications in WebLogic Server.

Main Tasks for Creating Coherence Applications

A Coherence application can be created 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.

- Task One: Create a Coherence Application Directory Structure
- Task Two: Include the Coherence Application's Artifacts
- Task Three: Package the Coherence Application for Deployment

Task One: Create a Coherence Application Directory Structure

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

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

Task Two: Include the Coherence Application's Artifacts

Include the Coherence application artifacts in the staging directory. For details on creating Coherence applications, see 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/
   ```


Copyright © 2007, 2026, Oracle and/or its affiliates.

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>
```

Application requirements

Complete Preparing a Release-Matched WebLogic Deployment and Packaging Application Trust Metadata in a GAR before first startup. Preserve container lifecycle and class-loader ownership; a standalone bootstrap example is not a replacement for WebLogic deployment management.

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. SeeDeploying Coherence Applications in WebLogic Server.

1. From the command line, change directories to the root of the staging directory.

Copyright © 2007, 2026, Oracle and/or its affiliates.

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. See *Developing Applications for Oracle WebLogic Server* for details on developing EAR. 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>
```

Preparing a Release-Matched WebLogic Deployment

Match the domain runtime and build dependencies while retaining container ownership of Coherence resources.

Use the Coherence and JDK combination certified for the actual WebLogic distribution/domain. The standalone Coherence Java 21 baseline and availability of a 26.1 JAR do not authorize replacing WebLogic-owned libraries. Apply the supported patch/installation procedure and keep domain members at the intended compatible versions.

- Record WebLogic, Coherence, Java, operating-system and application API levels before building the GAR/EAR. Compile against the server-provided APIs using provided scope where the server owns them.
- Select the matching Maven BOM/build plugins and repository from the supported installation. Generate POF and application trust metadata during the application build, not by changing server JARs.
- Keep container-owned resources under the WebLogic lifecycle. Reuse injected/JNDI cache handles and managed work; do not call Coherence.close, CacheFactory.shutdown or create unmanaged cluster instances from request code.
- Verify class-loader visibility for model classes, serializers and generated metadata on data, proxy and application tiers. A resource in a WAR is not automatically visible to an independently deployed GAR.
- Prepare hardened mode requirements before first upgraded startup. Run representative operations and failure cases with the actual application identity and deployment packaging.

Use Preparing Applications for the Hardened Default and Upgrading and Operating a Secured Deployment for the release migration checks.

Copyright © 2007, 2026, Oracle and/or its affiliates.

2

Creating Coherence Applications for WebLogic Server

Coherence caches can be accessed using dependency injection and JNDI. Coherence applications can be packaged as a Grid ARchive (GAR).
This chapter includes the following sections:

- 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.
- Creating a Coherence Application Deployment Descriptor
  The presence of the deployment descriptor indicates a valid GAR. A GAR file must contain a Coherence application deployment descriptor (`coherence-application.xml`) located in the `META-INF` directory.
- Using JNDI to Override Configuration
  Coherence provides the ability to override any XML element value in a configuration file using JNDI properties.
- Defining a Data Cache
- Accessing a Data Cache
- Using the Coherence API
- Using a Coherence Application Lifecycle Listener
  The Coherence Application Lifecycle listener allows custom processing to occur before and after the creation and destruction of Coherence caches and clustered services.
- Using Coherence for Session Management
- Creating Extend Clients in WebLogic Server
- Using a JCache Cache in WebLogic Server
- Packaging Application Trust Metadata in a GAR
  Generate and package application policy where the deployed model and executable class loader can discover it.

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 (`coherence-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.

Copyright © 2007, 2026, Oracle and/or its affiliates.

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.

Application requirements

Complete Preparing a Release-Matched WebLogic Deployment and Packaging Application Trust Metadata in a GAR before first startup. Preserve container lifecycle and class-loader ownership; a standalone bootstrap example is not a replacement for WebLogic deployment management.

- Directory Structure Example
- Packaging a Grid Archive In an Enterprise Application
- Creating a Coherence Project from a Maven Archetype
  You can create a new Coherence project using the Coherence Maven archetype.

Directory Structure Example

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

Example 2-1 Coherence Application Directory Structure

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


Copyright © 2007, 2026, Oracle and/or its affiliates.

Packaging a Grid Archive In an Enterprise Application

A GAR must be packaged in an EAR to be referenced by other Jakarta EE modules. See *Developing Applications for Oracle WebLogic Server* for details on creating an EAR. The following is an example of a Coherence application that is packaged within an EAR:

Example 2-2 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>
```

Creating a Coherence Project from a Maven Archetype

You can create a new Coherence project using the Coherence Maven archetype.

1. To create a new Coherence project using the Coherence Maven archetype, run a command similar to the following:

   ```
   mvn archetype:generate   -DarchetypeGroupId=com.oracle.coherence.archetype   -
   DarchetypeArtifactId=gar-maven-archetype   -DarchetypeVersion=26.1.0-0-0   -
   DgroupId=org.mycompany   -DartifactId=my-gar-project   -Dversion=1.0-SNAPSHOT
   ```

   This command runs Maven's `archetype:generate` goal which lets you create a new project from an archetype. Table 2-1 describes the parameters.

Table 2-1 Parameters for the Coherence Projects

Parameter	Purpose
`archetypeGroupId`	The group ID of the archetype that you want to use to create the new project. This must be `com.oracle.coherence.archetype`.
`archetypeArtifactId`	The artifact ID of the archetype that you want to use to create the new project. This must be `gar-maven-archetype`.
`archetypeVersion`	The version of the archetype that you want to use to create the new project. This must be `26.1.0-0-0`.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 2-1 (Cont.) Parameters for the Coherence Projects

Parameter	Purpose
`groupId`	The group ID for your new project. This usually starts with your organization's domain name in reverse format.
`artifactId`	The artifact ID for your new project. This is usually an identifier for this project.
`version`	The version for your new project. This is usually `1.0-SNAPSHOT` for a new project.

You can also run the command without any arguments, as shown in the following example. In this case, Maven displays a list of available archetypes and prompts you to enter the required information.

```
mvn archetype:generate
```

After you create the project, the following files get included in it:

Figure 2-1 Maven Project Files

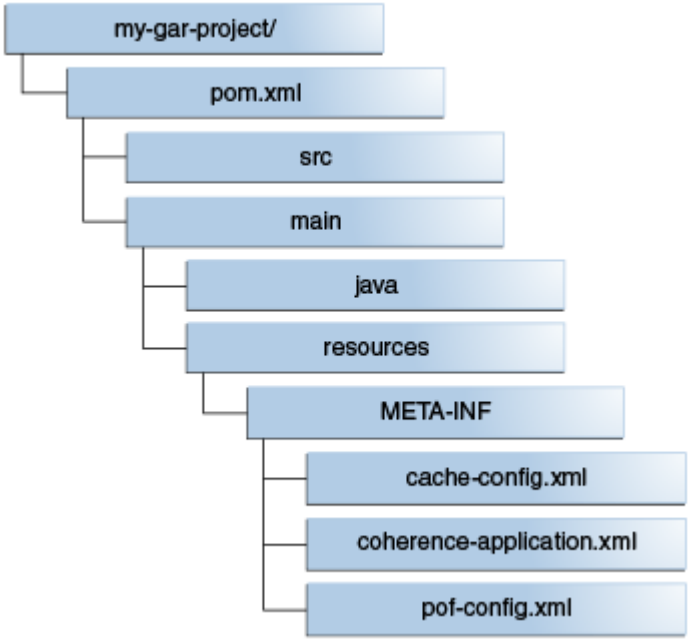


Table 2-2 describes the files included in the project.

Table 2-2 Files Created for the Coherence Project

File	Purpose
`pom.xml`	The Maven Project Object Model (POM) file that describes your new project. It includes the Maven coordinates that you specified for your project and the appropriate plug-in definitions to use the Coherence Maven plug-in to build your project into a gar file.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 2-2 (Cont.) Files Created for the Coherence Project

File	Purpose
`coherence-application.xml`	A starter Coherence GAR deployment descriptor for your GAR file.
`pof-config.xml`	A starter Coherence Portable Object Format (POF) configuration file. The POF configuration file is processed and inserted into the final GAR file if the plug-in option `generatePof` is set to `true`. By default, POF configuration metadata will not be generated.

Here are the samples of the generated files listed in the above table:

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.mycompany</groupId>
  <artifactId>my-gar-project2</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>gar</packaging>
  <dependencies>
    <dependency>
      <groupId>com.oracle.coherence</groupId>
      <artifactId>coherence</artifactId>
      <version>26.1.0-0-0</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>com.oracle.coherence</groupId>
        <artifactId>gar-maven-plugin</artifactId>
        <version>26.1.0-0-0</version>
        <extensions>true</extensions>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <version>3.3.0</version>
        <configuration>
          <forceCreation>true</forceCreation>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

coherence-application.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<coherence-application xmlns="http://xmlns.oracle.com/coherence/coherence-
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

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

pof-config.xml

```
<?xml version="1.0"?>
<!-- Note: To add custom types, create a new 'pof-config.xml' file
     containing your custom user-type elements, and place it at the
     beginning of the class path. -->

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

2. If you are using POF in your project, you must add the following parameter into your project's POM file:

Parameter	Purpose
`generatePof`	The POF configuration file is generated and inserted into the final GAR file if this plug-in option is `true`. The configuration file is generated by scanning all classes in the GAR's classpath annotated with the class `com.tangosol.io.pof.annotation.Portable`. By default, POF configuration metadata is not generated.

3. To generate a GAR with correctly generated `pof-config.xml`, add the following to your GAR plug-in configuration in the POM:

```
<build>
<plugins>
…
    <plugin>
      <groupId>com.oracle.coherence</groupId>
      <artifactId>gar-maven-plugin</artifactId>
      <version>26.1.0-0-0</version>
      <extensions>true</extensions>
      <configuration>
         <generatePof>true</generatePof>
      </configuration>
    </plugin>
…
  </plugins>
</build>
```

These commercial coordinates are for the matching 26.1 build artifacts. Use the exact patch coordinates supplied for your supported domain, and inspect the generated POM rather than deriving an invalid version range. The core dependency is provided by WebLogic. Add the matching security-config-maven-plugin generation step described in Building Examples with

Copyright © 2007, 2026, Oracle and/or its affiliates.

Application Security Metadata before packaging application classes. POF generation alone does not generate the application executable/type trust policy.

Creating a Coherence Application Deployment Descriptor

The presence of the deployment descriptor indicates a valid GAR. A GAR file must contain a Coherence application deployment descriptor (`coherence-application.xml`) located in the `META-INF` directory.

For a detailed reference of all the available elements in the descriptor, see 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>
```

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>
```

Defining a Data Cache

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

The following example creates a distributed cache named `myCache`. As an alternative, a cache mapping 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"
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

```
   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. Additionally, the replicated cache type is deprecated. View, Near, or Distributed/Partitioned Cache types should be used instead. The read and write performance and memory usage characteristics described in Table 12-1 Summary of Cache Types and Characteristics can be used to determine which cache type to use.

Accessing a Data Cache

Applications use the Coherence `NamedCache` API to interact with a Coherence cache. The Coherence cache 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 2-3 illustrates using dependency injection to get a cache named `myCache`. See *Developing Applications for Oracle WebLogic Server* for details on Jakarta EE annotations and dependency injection.

Example 2-3 Obtaining a Cache Resource by Dependency Injection

```
// fragment in a container-managed component; use the annotation API supplied by the
server
@Resource(mappedName = "MyCache")
com.tangosol.net.NamedCache<String, String> cache;
```

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 2-4 illustrates a `<resource-ref>` element that

Copyright © 2007, 2026, Oracle and/or its affiliates.

identifies `myCache` as the `NamedCache`. See *Developing JNDI Applications for Oracle WebLogic Server* for details on using JNDI in 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 2-4 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 2-4:

```
// fragment in a method that can throw NamingException
var context = new javax.naming.InitialContext();
try {
    @SuppressWarnings("unchecked")
    var cache = (com.tangosol.net.NamedCache<String, String>)
            context.lookup("java:comp/env/coherence/myCache");
    cache.put("example", "value");
} finally {
    context.close();
}
```

Application requirements

Use a consistent mapped resource name and matching key/value types. The container owns the looked-up cache; closing the naming Context does not destroy or close that cache. JNDI remains `javax.naming` on current Java. Select javax.annotation or jakarta.annotation only according to the server API level.

Using the Coherence API

Coherence provides a full-featured API for interacting with a cache and for performing data grid operations.
Some of the features of the Coherence API include:

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

See Developing Applications with Oracle Coherence for details on using the API to develop applications. See Java API Reference for Oracle Coherence for a reference of the Coherence API.

Copyright © 2007, 2026, Oracle and/or its affiliates.

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. See Developing Applications with Oracle Coherence for details on using POF in your applications.

Using a Coherence Application Lifecycle Listener

The Coherence Application Lifecycle listener allows custom processing to occur before and after the creation and destruction of Coherence caches and clustered services.

Coherence applications support the use of an application lifecycle listener. The listener class must implement the `com.tangosol.application.LifecycleListener` interface. See the Java API Reference for Oracle Coherence 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>
```

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. See Administering HTTP Session Management with Oracle Coherence*Web for details on setting up, configuring, and using Coherence*Web in WebLogic Server.

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

Copyright © 2007, 2026, Oracle and/or its affiliates.

within a WebLogic Server domain or may be external to WebLogic Server. See *Administering Clusters for Oracle WebLogic Server* for details on setting up a Coherence proxy server tier in WebLogic Server to allow remote connections. See Developing Remote Clients for Oracle Coherence for details on creating Coherence*Extend client applications.

Application requirements

Complete Preparing a Release-Matched WebLogic Deployment and Packaging Application Trust Metadata in a GAR before first startup. Preserve container lifecycle and class-loader ownership; a standalone bootstrap example is not a replacement for WebLogic deployment management.

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.

See Developing Applications with Oracle Coherence for details on using JCache with Coherence.

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 JCache API to create and use a JCache cache. For example, within a servlet:

```
// fragment: use the CacheManager owned by this application/integration
static javax.cache.Cache<ContactId, Contact> getCache(
        javax.cache.CacheManager manager, String name) {
    var cache = manager.getCache(name, ContactId.class, Contact.class);
    if (cache == null) {
        var configuration = new
com.tangosol.coherence.jcache.partitionedcache
                .PartitionedCacheConfiguration<ContactId, Contact>();
        configuration.setTypes(ContactId.class, Contact.class);
        configuration.setStatisticsEnabled(true);
        configuration.setManagementEnabled(true);
        cache = manager.createCache(name, configuration);
    }
    return cache;
}
```

JCache uses javax.cache even in a Jakarta-based application. Initialize/create the named cache once under the deployment lifecycle; concurrent createCache callers must coordinate rather than treating every IllegalStateException as a cache-missing signal. Package compatible model and generated trust metadata on the executing tiers. Close only a CacheManager owned by the application, at application shutdown.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Packaging Application Trust Metadata in a GAR

Generate and package application policy where the deployed model and executable class loader can discover it.

A GAR/EAR deployment must make application trust metadata visible to the same class loader that loads model and executable classes. Default 26.1 mode is hardened before any explicit property is set.

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

- Generate the security-config.xml resource as part of the application build and keep it in the packaged artifact. Do not replace or edit the framework metadata inside coherence.jar.
- Keep POF type IDs and schema evolution compatible across rolling deployments. Deploy model/executable implementations and their trust metadata to every tier that consumes them.
- A shared library can supply classes/metadata only when its class-loader relationship makes both resources visible. Check the deployed artifact and actual loader, not only a source directory.
- Configure transport identity and authorization through the domain and Coherence integration. An admitted class is not an authenticated user or authorization decision.
- Test a legitimate operation and a rejected unregistered payload/executable with the actual deployment identity. Include session attributes, listeners and persisted data where used.

Follow Securing Serialization and Remote Execution and Building Examples with Application Security Metadata.

Copyright © 2007, 2026, Oracle and/or its affiliates.

3

Deploying Coherence Applications in WebLogic Server

To manage Coherence servers in a WebLogic Server domain, you must deploy Coherence applications packaged as a Grid ARchive (GAR).
See Packaging Coherence Applications for details on creating a GAR.

This chapter includes the following sections:

- Understanding Coherence Deployment Tiers
- Deploying Applications to Managed Coherence Servers
- Deploying Coherence Applications as Shared Libraries
- Referencing Shared Libraries from a Coherence Application
- Performing a Rolling Redeploy
- Securing Coherence Applications in WebLogic Server
- Verifying a Rolling GAR Redeployment
  Wait for service recovery and application behavior after each targeted deployment step.

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.

See *Administering Clusters for Oracle WebLogic Server*.

Deploying Applications to Managed Coherence Servers

Coherence application GAR contains the artifacts of a Coherence application and includes a deployment descriptor. A GAR can be deployed as a standalone module or as a part of an EAR using any WebLogic Server deployment tool.
Coherence application GAR modules get deployed the same way as Jakarta EE modules and can be deployed using any WebLogic Server deployment tool: WebLogic Remote Console, the

Copyright © 2007, 2026, Oracle and/or its affiliates.

Oracle WebLogic Scripting Tool (WLST), the WebLogic Server `Deployer` class, and the WebLogic Server `<wldeploy>` ANT target.

See *Oracle WebLogic Remote Console Online Help*.

See *Understanding the WebLogic Scripting Tool*.

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. 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. 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.

Deploying Coherence Applications as Shared Libraries

Coherence applications are packaged as Grid ARchive (GAR) and deployed as shared libraries in a single-tier or multi-tier domain.
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 Jakarta 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>
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

```
   </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.

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 weblogic-coh-app.xml Deployment Descriptor Elements. For example:

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


Copyright © 2007, 2026, Oracle and/or its affiliates.

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

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 in-memory cache data loss when utilizing simultaneous redeployment across all managed Coherence servers.

> **Note**
>
> To ensure there is no persistence or write-through cache data loss during a simultaneous redeployment of GAR modules to all managed Coherence servers, you can leverage the service suspend feature. A service is considered suspended only after all the data is fully written, including active persistence mode, asynchronous persistence tasks, entries in the write-behind queue of a read-write backing map, and other asynchronous operations. Outstanding operations are completed and no new operations are allowed against the suspended services. Thus, for a controlled complete redeploy of GAR modules to all managed Coherence servers, Oracle recommends executing the Coherence `ClusterMBean` operation `suspendService("impactedService")` which suspends each service with potential asynchronous operations gracefully before undeploying GAR modules from all managed Coherence servers.

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. Only in-memory cache 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. See Managing Oracle Coherence for details on this metric.

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 Remote 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.

Copyright © 2007, 2026, Oracle and/or its affiliates.

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. 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
```

Application requirements

The examples show target-selection syntax, not production credential handling. Apply Verifying a Rolling GAR Redeployment and configure the protected administration URL and stored credentials before use. Do not copy placeholder passwords or a cleartext t3 URL into production automation.

Securing Coherence Applications in WebLogic Server

Coherence applications that are deployed to managed Coherence servers can be secured. See Securing Oracle Coherence for details on securing the Coherence applications.

Application requirements

Complete Preparing a Release-Matched WebLogic Deployment and Packaging Application Trust Metadata in a GAR before first startup. Preserve container lifecycle and class-loader ownership; a standalone bootstrap example is not a replacement for WebLogic deployment management.

Verifying a Rolling GAR Redeployment

Wait for service recovery and application behavior after each targeted deployment step.

A rolling application redeploy changes classes, policy resources and services together. Qualify the exact server/release combination and data format before using it in production.

- Retain a verified deployment artifact, configuration and recovery point. Confirm current backup ownership/service safety and capacity for one member being unavailable.

Copyright © 2007, 2026, Oracle and/or its affiliates.

- Verify serializer/POF/model compatibility and executable definitions across old/new application revisions. Deploy trust metadata with the matching classes; prepare any explicitly approved transition properties before startup.
- Target one supported deployment unit/member at a time using the documented WebLogic deployment tool. Use the domain credential store/user-config mechanism and a protected administration channel; do not put a real password into shell history.
- Wait for deployment activation, service recovery/redistribution and required health/safety. Exercise a representative request through the proxy/application tier before proceeding.
- On failure, stop the rollout, diagnose class visibility/policy and recover using the qualified rollback procedure. Restoring an old GAR does not guarantee that it can read data written by a newer serializer or persistence engine.

Use Performing a Rolling Redeploy for targeting semantics and Operating and Recovering Unified Persistence Journal for persistence recovery checks.

Copyright © 2007, 2026, Oracle and/or its affiliates.

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:

- coherence-application.xml Namespace Declaration and Schema Location
- application-lifecycle-listener
  The application-lifecycle-listener elements specify the fully qualified name of a class or the initialization parameter that is required by the implementation.
- cache-configuration-ref
- coherence-application
  The `coherence-application` element is the root element of the Coherence application deployment descriptor.
- 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.
- init-params
  The `init-params` element specifies an initialization parameter. Any number of `init-params` elements may be defined.
- pof-configuration-ref

coherence-application.xml Namespace Declaration and Schema Location

The Coherence application deployment descriptor schema is defined in the `coherence-application.xsd` file, which is located in the root of the `coherence.jar` library. The web URL is http://xmlns.oracle.com/coherence/coherence-application/1.0/coherence-application.xsd
The `<coherence-application>` element is the root element of the descriptor and includes the XSD reference, the namespace reference, and the location of the `coherence-application.xsd` file. For example:

```
<?xml version='1.0'?>
<coherence-application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.oracle.com/coherence/coherence-application"
   xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-application
   coherence-application.xsd">
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

application-lifecycle-listener

The application-lifecycle-listener elements specify the fully qualified name of a class or the initialization parameter that is required by the implementation.

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.

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.

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.

Copyright © 2007, 2026, Oracle and/or its affiliates.

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.

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.

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.

Copyright © 2007, 2026, Oracle and/or its affiliates.

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`

`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>`
		`  <init-param>`
		`  <param-type>java.lang.String</param-type>`
		`  <param-value>EmployeeTable</param-value>`
		`  </init-param>`
		`</init-params>`

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.

Copyright © 2007, 2026, Oracle and/or its affiliates.

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:

- weblogic-coh-app.xml Namespace Declaration and Schema Location
  Learn the correct text for the namespace declaration and schema location for the `weblogic-coh-app.xml` deployment descriptor file.
- weblogic-coh-app
  The weblogic-coh-app element is the root element of the WebLogic Coherence application deployment descriptor.
- library-ref
  The `library-ref` element specifies a shared library module that is intended to be used as a library in a Coherence application.

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

Learn the correct text for the namespace declaration and schema location for the `weblogic-coh-app.xml` deployment descriptor file.

The correct text is as follows:

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

weblogic-coh-app

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

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.

Copyright © 2007, 2026, Oracle and/or its affiliates.

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`.

Copyright © 2007, 2026, Oracle and/or its affiliates.