Oracle® Coherence

User's Guide for Oracle Coherence*Web

Release 3.4.2

E14408-01

February 2009

ORACLE®

Oracle Coherence User's Guide for Oracle Coherence*Web, Release 3.4.2

E14408-01

Primary Author: Tom Pfaeffle

Contributing Author: Noah Arliss, Mark Falco, Alex Gleyzer, Gene Gleyzer, Jason Howes, James Kirsch, Adam Leftik, Rob Misek, PatrickPeralta, Everett Williams

Contents

4 Coherence*Web Session Management Features

5 Installing Coherence*Web on WebLogic Portal 10.3

List of Examples

List of Figures

List of Tables

Preface

This document describes how to install and configure Coherence*Web.

Audience

This document is intended for application developers who want to be able to manage session state in clustered environments.

Documentation Accessibility

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

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

TTY Access to Oracle Support Services

To reach AT&T Customer Assistants, dial 711 or 1.800.855.2880. An AT&T Customer Assistant will relay information between the customer and Oracle Support Services at 1.800.223.1711. Complete instructions for using the AT&T relay services are available at `http://www.consumer.att.com/relay/tty/standard2.html`. After the AT&T Customer Assistant contacts Oracle Support Services, an Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process.

Related Documents

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

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

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
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

This chapter provides general information about Coherence*Web. It describes what Coherence*Web is, what containers it is supported on, and provides a road map for the deployment decisions you should consider based on your environment.

What is Coherence*Web?

Coherence*Web is an HTTP session management module dedicated to managing session state in clustered environments. Built on top of Oracle Coherence, Coherence*Web:

- enables session sharing and management across different Web applications, domains and heterogeneous application servers.
- brings Coherence data grid's data scalability, availability, reliability, and performance to in-memory session management and storage.
- supports all of the mainstream application servers such as Oracle WebLogic Server, IBM WebSphere, Tomcat, and so on (see "Supported Web Containers" on page 1-2).
- supports numerous portal containers, including Oracle WebLogic Portal (see Chapter 5, "Installing Coherence*Web on WebLogic Portal 10.3").
- allows for session state to be managed in the various caching topologies available in Coherence (that is, Replicated, Partitioned, Near Caching, Read-Through, Write-Through, Write-Behind and Refresh-Ahead Caching, and so on).
- allows storage of session data outside of the Java EE application server, freeing application server heap space and enabling server restarts without session data loss (see "Deployment Topologies" on page 4-13).
- supports multiple advanced session models (that is, Monolithic, Traditional, and Split Session) which define how the session state is physically managed and serialized/deserialized in the cluster (see "Session Models" on page 4-1).
- supports fine-grained session and session attribute scoping by way of pluggable policies (see "Session and Session Attribute Scoping" on page 4-6).

Using Coherence*Web with WebLogic Server and WebLogic Portal

Starting with Coherence 3.4.2, Coherence*Web integrates with WebLogic Server 10.3 and WebLogic Portal 10.3 using the native WebLogic session management SPI. The result of this tighter integration with WebLogic is simplified installation and deployment that no longer requires application instrumentation (using the WebInstaller).

- Chapter 2, "Installing Coherence*Web on the WebLogic Server 10.3," describes the SPI-based WebLogic Server implementation of Coherence*Web in more detail.
- Chapter 5, "Installing Coherence*Web on WebLogic Portal 10.3," describes the SPI-based WebLogic Portal implementation of Coherence*Web in more detail.

Coherence*Web and other Application Servers

For earlier versions of WebLogic Server and other third-party application servers, Coherence*Web provides a generic installer that transparently instruments your Web applications.

Chapter 3, "Installing Coherence*Web on Other Application Servers," describes the implementation of Coherence*Web with WebInstaller in more detail.

Supported Web Containers

Table 1–1 summarizes the Web containers supported by the Coherence*Web Session Management Module. It also provides links to the information required to install Coherence*Web on them. You will notice that all of the Web containers (with the exception of Oracle WebLogic Server 10.3) share the same general installation instructions. A few, such as Oracle OC4J, Caucho, and WebLogic 10.x, require extra, container-specific steps that you must complete before starting the general installation instructions.

To install the Coherence*Web Session Management Module on WebLogic Server 10.3 and later, you can use only SPI-based installation. For instructions on installing the Management Module on WebLogic Server 10.3, see Chapter 2, "Installing Coherence*Web on the WebLogic Server 10.3.".

Note: The value in the **Server Type Alias** column is used only by the Coherence*Web WebInstaller installation. The value is passed to the WebInstaller though the `-server` command line option.

Table 1–1 Web Containers Supported by Coherence*Web

Application Server	Server Type Alias	See this Installation Section
Apache Tomcat 4.1.x	Generic	"General Instructions for Installing Coherence*Web Session Management Module"
Apache Tomcat 5.0.x	Generic	"General Instructions for Installing Coherence*Web Session Management Module"
Apache Tomcat 5.5.x	Generic	"General Instructions for Installing Coherence*Web Session Management Module"
Apache Tomcat 6.0.x	Generic	"General Instructions for Installing Coherence*Web Session Management Module"
Caucho Resin 3.0.x	Resin/3.0.x	"Installing on Caucho Resin 3.0.x"
Caucho Resin 3.1.x	Resin/3.1.x	"Installing on Caucho Resin 3.1.x"
IronFlare Orion 2.0.x	Orion/2.0.x	"General Instructions for Installing Coherence*Web Session Management Module"
IBM WebSphere 5.x	WebSphere/5.x	"General Instructions for Installing Coherence*Web Session Management Module"

*Table 1–1 (Cont.) Web Containers Supported by Coherence*Web*

Application Server	Server Type Alias	See this Installation Section
IBM WebSphere 6.x	WebSphere/6.x	"General Instructions for Installing Coherence*Web Session Management Module"
JBoss Application Server	Jetty/4.2.x, Jetty/5.1.x, *or* Generic	"General Instructions for Installing Coherence*Web Session Management Module"
Jetty 4.2.x	Jetty/4.2.x	"General Instructions for Installing Coherence*Web Session Management Module"
Jetty 5.1.x	Jetty/5.1.x	"General Instructions for Installing Coherence*Web Session Management Module"
Jetty 6.1.x	Generic	"General Instructions for Installing Coherence*Web Session Management Module"
New Atlanta ServletExec 5.0	ServletExec/5.x	"General Instructions for Installing Coherence*Web Session Management Module"
Oracle OC4J 10.1.2.x	Oracle/10.1.2.x	"Installing on Oracle OC4J 10.1.2.x"
Oracle OC4J 10.1.3.x	Oracle/10.1.3.x	"General Instructions for Installing Coherence*Web Session Management Module"
Oracle WebLogic 8.x	WebLogic/8.x	"General Instructions for Installing Coherence*Web Session Management Module"
Oracle WebLogic 9.x	WebLogic/9.x	"General Instructions for Installing Coherence*Web Session Management Module"
Oracle WebLogic 10.x	WebLogic/10.x	"Installing on Oracle WebLogic 10.x"
Oracle WebLogic 10.3 and later	NA	Only SPI-based installation is available for versions 10.3 and later. See Chapter 2, "Installing Coherence*Web on the WebLogic Server 10.3."
Sun ONE 6.x	Generic	"General Instructions for Installing Coherence*Web Session Management Module"
Sun ONE 7.x	Generic	"General Instructions for Installing Coherence*Web Session Management Module"
Sun ONE 8.x	Generic	"General Instructions for Installing Coherence*Web Session Management Module"

Installation and Deployment Road Map

This section provides a general outline of the deployment decisions you should make before you configure and install Coherence*Web. Coherence*Web is supported on a number of different application servers. The type of application server that you are deploying on determines whether you will be installing Coherence*Web using the WebLogic SPI installation or the WebInstaller. Regardless of which application server you are using, you may need to change some of the Coherence*Web configuration options to meet your particular requirements, such as packaging considerations, session model, session locking mode and deployment topology.

Choose your Cluster Node Isolation

Cluster node isolation refers to the number of Coherence nodes that will be created within each application server JVM and where the Coherence library is deployed. A few different isolation modes are supported.

For example: you may be deploying multiple applications to the container that require the use of the same cluster (or one coherence node); you may have multiple Web

applications packaged in a single EAR file that want to use a single cluster; or you may have web applications that must keep their session data separate and need to be deployed to their own individual Coherence cluster. These choices and the deployment descriptors and elements that need to be configured are described in "Cluster Node Isolation" on page 4-8.

Choose your Locking Mode

Locking mode refers to the behavior of HTTP sessions when they are accessed concurrently by multiple web container threads. You can choose to allow multiple nodes in a cluster to access an HTTP session simultaneously, to not allow more than one node in the cluster to access an HTTP session, or to not allow more than one thread in the cluster to access an HTTP session. These choices, and the deployment descriptors and elements that need to be configured are described in "Session Locking Modes" on page 4-11.

Choose How to Scope Sessions and Session Attributes

Session and session attribute scoping refers to the fine-grained control over how both session data and session attributes are scoped (or "shared") across application boundaries. Coherence*Web supports sharing sessions across web applications as well as restricting what attributes within a session are shared across the application boundaries. These choices, and the deployment descriptors and elements that need to be configured are described in "Session and Session Attribute Scoping" on page 4-6.

Choose the Installation Method

The installation procedure that you follow will depend on your application server. "Supported Web Containers" on page 1-2 provides a list of the application servers supported by Coherence*Web.

- For WebLogic Server 10.3 and WebLogic Portal 10.3, use the native WebLogic Server SPI-based installation procedure. This is described in Chapter 2, "Installing Coherence*Web on the WebLogic Server 10.3."

  Note that the installation of Coherence*Web on WebLogic Portal 10.3 is completely independent of WebLogic Server; that is, you do not have to install Coherence*Web on WebLogic Server to install it on WebLogic Portal. The installation on WebLogic Portal is described in Chapter 5, "Installing Coherence*Web on WebLogic Portal 10.3."

- For other application servers, use the generic Java EE Web application instrumentation. This installation procedure is described in Chapter 3, "Installing Coherence*Web on Other Application Servers."

2

Installing Coherence*Web on the WebLogic Server 10.3

The current release of Coherence*Web provides a deployment option on the WebLogic Server platform that enables a tighter integration with WebLogic Server. The installation and configuration options described in this chapter apply only to WebLogic Server 10.3 deployments.

Overview of the Coherence*Web SPI

Coherence*Web is not a replacement for WebLogic Server's in-memory HTTP state replication services. However, Coherence*Web should be considered when an application has large HTTP session state objects, when running into memory constraints due to storing HTTP session object data, or if you have an existing Coherence cluster and want to off-load HTTP Session storage to a Coherence cluster.

The most significant change introduced by this new deployment option is that applications deployed using the Coherence*Web SPI module no longer require an application to be instrumentation by the Coherence*Web WebInstaller.

Location of the Coherence*Web SPI

The WebLogic Server Coherence*Web SPI consists of the `coherence-web-spi.war` file, located in the `coherence\lib` directory in the Coherence distribution. The `coherence.jar` file, located in the same directory, is also necessary for enabling Coherence*Web functionality in WebLogic Server.

Requirements for Using the Coherence*Web SPI

The Coherence*Web SPI for WebLogic Server requires that a load balancer which enforces HTTP session JVM affinity is running in front of the WebLogic Server tier. WebLogic Server ships with several different proxy plug-ins which will enforce JVM session stickiness. Documentation for configuring the WebLogic Server proxy plug-in is available here:

`http://download.oracle.com/docs/cd/E12840_01/wls/docs103/cluster/load_balancing.html#wp1026940`

Coherence*Web SPI Configurations for the WebLogic Server

There are two differences between the default cache configuration for the Coherence*Web SPI for WebLogic Server and Coherence*Web:

- The Coherence*Web SPI for WebLogic Server is configured with local-storage disabled. This means a Coherence cache server must be running in its own JVM, separate from the JVM running WebLogic Server.
- The timeout for requests to the cache server to respond is 30 seconds. If a request to the cache server has not responded in 30 seconds, a `com.tangosol.net.RequestTiemoutException` exception is thrown.

The Coherence caches used by the Coherence*Web SPI are configured by the `session-cache-config.xml` file. This file is located inside the `coherence-web-spi.war` file under the `WEB-INF\classes` directory. Any cache configuration change should be put inside `session-cache-config.xml`, and then repackaged inside `coherence-web-spi.war`.

Overview Of Configuration and Deployment

The Coherence*Web distribution includes a deployable shared library that contains a native plug-in to WebLogic Server's HTTP Session Management interface. To enable Coherence*Web on WebLogic Server for a Web application, complete the following steps:

1. Apply the WebLogic Server publicly available patch ID N41D to all WebLogic Server instances that are hosting the Web applications that will use Coherence*Web.

   See the instructions for using Smart Update to install WebLogic Server patches.

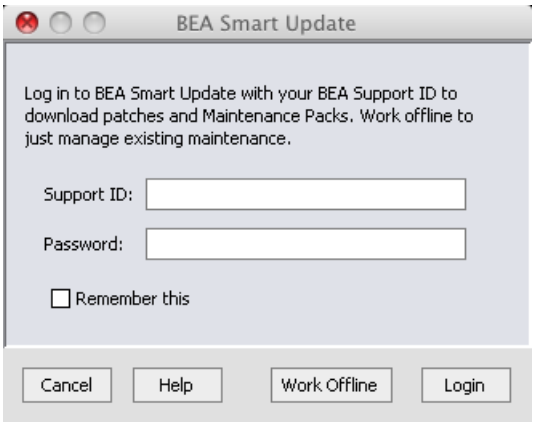
   `http://download.oracle.com/docs/cd/E11035_01/smartUpdate31/guide/install.html#wp1091614`

   For production environments it is recommended that you review the Smart Update production installation:

   `http://download.oracle.com/docs/cd/E12840_01/common/smartupdate/guide/remote.html#wp1071859`

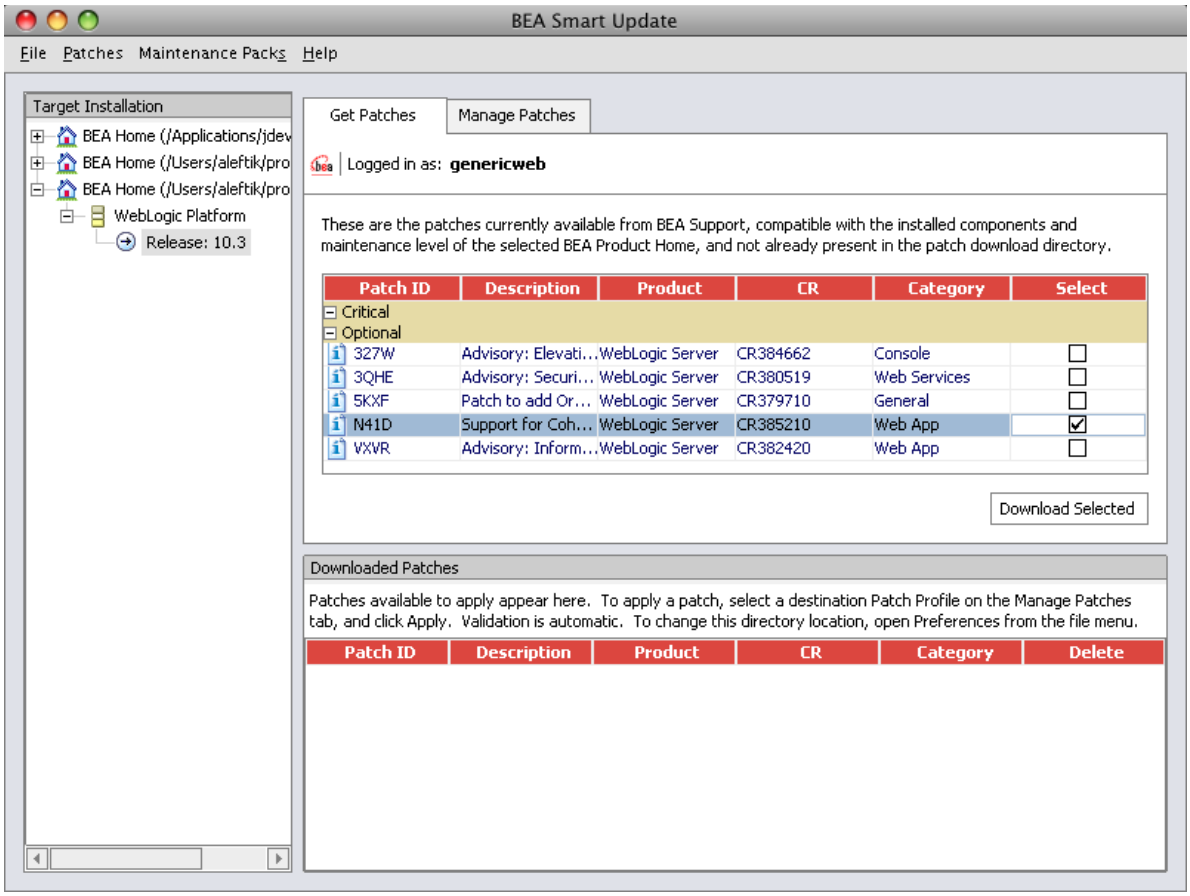
   a. Log in to Smart Update using your Support ID and Password.

Figure 2–1 WebLogic Smart Update Login Dialog Box



   b. Download and apply patch N41D. Restart the WebLogic Server.

Figure 2–2 WebLogic Smart Update Tree Browser



2. (Optional) modify the `session-cache-config.xml` file to customize the Cache topology for Coherence*Web.

   See Appendix B, "Session Cache Configuration File" for a description of the default configuration of the `session-cache-config.xml` file.

3. Start a Cache Server Tier in a separate JVM from the one running WebLogic Server.

   See "Configuring and Starting a Cache Server" on page 2-4 for more information.

4. Determine the appropriate packaging based on your deployment requirements and follow the packaging instructions.

   See "Packaging Applications and Configuring Cluster Nodes" on page 2-4 for more information.

5. (Optional) Modify the `web.xml` and `weblogic.xml` files in the WAR deployment if advanced configuration is required for a Web application using Coherence*Web.

   Coherence Web parameters that can be configured for Web applications running on the WebLogic Server are described in "Configuring Web Applications for Coherence*Web" on page 2-6. The entire set of Coherence*Web parameters are described in Appendix A–1, " Configuration Parameters for Coherence*Web."

Note: If you are deploying Coherence*Web in a WebLogic Portal environment, see Chapter 5, "Installing Coherence*Web on WebLogic Portal 10.3" for installation instructions.

Configuring and Starting a Cache Server

A Cache Server JVM is a dedicated Coherence JVM that is responsible for storing and managing all cached data (in this case, `HttpSession` state). One or more cache server JVMs must be started before the WLS/WLP JVMs can be started.

1. Create a script for starting a cache server JVM. The following is an very simple example of a script that starts a storage-enabled cache server for use with Coherence*Web. This example assumes that you are using a Sun JVM. See *JVM Tuning* in the *Developer's Guide for Oracle Coherence* for more information.

   ```
   java -server -Xms512m -Xmx512m -cp <Coherence installation dir>/lib/coherence.
   jar:<Coherence installation dir>/lib/coherence-web-spi.war -Dtangosol.
   coherence.management.remote=true -Dtangosol.coherence.cacheconfig=WEB-
   INF/classes/session-cache-config.xml -Dtangosol.coherence.session.
   localstorage=true com.tangosol.net.DefaultCacheServer
   ```

2. Start one or more cache server JVMs using the script described in the previous step.

Packaging Applications and Configuring Cluster Nodes

Coherence cluster nodes are class loader scoped. Therefore, you must configure the number of unique Coherence cluster nodes in a Coherence*Web deployment before packaging the application(s). The packing and configuration options are described in the following sections:

- Packaging and Configuring Application Server-Scoped Cluster Nodes
- Packaging and Configuring EAR-Scoped Cluster Nodes
- Packaging and Configuring WAR-Scoped Cluster Nodes

You can find detailed information about each of the options under "Cluster Node Isolation" on page 4-8.

As part of the Coherence*Web SPI for WebLogic Server you will find the `coherence.jar` and `coherence-web-spi.war` located in the `/lib` directory of the Coherence 3.4.2 distribution.

Note: The application server-scoped cluster configuration should be considered very carefully and *never* used in environments where application interaction is unknown or unpredictable.

An example of such an environment may be a deployment where multiple application teams are deploying applications written independently, without carefully coordinating and enforcing their conventions and naming standards. With this configuration, all applications are part of the same cluster and the likelihood of collisions between namespaces for caches, services, and other configuration settings is quite high and may lead to unexpected results.

For these reasons, Oracle Coherence strongly recommends that you use EAR-scoped and WAR-scoped cluster node configurations. If you are in doubt as to which deployment topology to choose, or if this warning applies to your deployment, then ***do not*** choose the application server-scoped cluster node configuration.

Packaging and Configuring Application Server-Scoped Cluster Nodes

1. Deploy `coherence-web-spi.war` as a shared library on each WebLogic Server.
2. Edit your WebLogic Server system classpath to include `coherence.jar` or copy the JAR to your `$`*`DOMAIN_HOME`*`/lib` directory.
3. Enable Coherence*Web in your Web application.

   Add the library reference stanza illustrated in Example 2–1 to the `weblogic.xml` in each WAR file deployed in the WebLogic server that intends to use Coherence*Web.

Example 2–1 Library Reference for Each WAR File

```
<weblogic-web-app>
     ...
      <library-ref>
           <library-name>coherence-web-spi</library-name>
          <specification-version>1.0.0.0</specification-version>
          <implementation-version>1.0.0.0</implementation-version>
          <exact-match>false</exact-match>
     </library-ref>
     ...
</weblogic-web-app>
```

Packaging and Configuring EAR-Scoped Cluster Nodes

1. Deploy `coherence-web-spi.war` as a shared library on each WebLogic Server.
2. Place `coherence.jar` in the EAR's `APP-INF/lib` directory.
3. Enable Coherence*Web.

   Create a shared library reference in each Web application in the EAR by adding the stanza illustrated in Example 2–2 to the `weblogic.xml` file:

Example 2–2 Library Reference for Each Web Application in the EAR

```
<weblogic-web-app>
     ...
      <library-ref>
```

```
            <library-name>coherence-web-spi</library-name>
           <specification-version>1.0.0.0</specification-version>
           <implementation-version>1.0.0.0</implementation-version>
           <exact-match>false</exact-match>
      </library-ref>
      ...
</weblogic-web-app>
```

Packaging and Configuring WAR-Scoped Cluster Nodes

1. Deploy `coherence-web-spi.war` as a shared library on each WebLogic Server.
2. Place `coherence.jar` in the WAR's `WEB-INF/lib` directory.
3. Enable Coherence*Web.

   Create a shared library reference by adding the stanza illustrated in Example 2–3 to the `weblogic.xml` file in the Web application's `WEB-INF` directory.

Example 2–3 Library Reference for the Web Application

```
<weblogic-web-app>
      ...
       <library-ref>
            <library-name>coherence-web-spi</library-name>
           <specification-version>1.0.0.0</specification-version>
           <implementation-version>1.0.0.0</implementation-version>
           <exact-match>false</exact-match>
      </library-ref>
      ...
<weblogic-web-app>
```

Configuring Web Applications for Coherence*Web

Since Coherence*Web is in control of the HTTP session lifecycle, most data from the `<session-descriptor>` element in either `weblogic.xml` or `weblogic-application.xml` is ignored.

The Coherence*Web SPI ships with a configuration that should be sufficient for most Web applications. If you need to make any changes to the configuration or override any previous settings, you can apply any of the Coherence*Web parameters described in Appendix A, "Coherence*Web Configuration Parameters," You can apply these parameters by using the `<context-param>` element in the `web.xml` file.

Table 2–1 lists the Coherence*Web parameters and identifies whether they have a WebLogic-specific default. For full descriptions of the parameters, see Appendix A, "Coherence*Web Configuration Parameters."

Table 2–1 Parameters that can be Configured in web.xml

Parameter	WebLogic-Specific Default
coherence-factory-class	No WebLogic-specific default.
coherence-sessioncollection-class	If unspecified, defaults to `com.tangosol.coherence.servlet.SplitHttpSessionCollection`.
coherence-cluster-owned	No WebLogic-specific default.
coherence-contextless-session-retain-millis	No WebLogic-specific default.
coherence-reaperdaemon-cluster-coordinated	No WebLogic-specific default.

Table 2–1 (Cont.) Parameters that can be Configured in web.xml

Parameter	WebLogic-Specific Default
coherence-reaperdaemon-sweep-modulo	No WebLogic-specific default.
coherence-reaperdaemon-assume-locality	If unspecified, defaults to `false`
coherence-reaperdaemon-cycle-second	No WebLogic-specific default.
coherence-reaperdaemon-priority	No WebLogic-specific default.
coherence-session-cachename	No WebLogic-specific default.
coherence-session-deathcert-cachename	No WebLogic-specific default.
coherence-session-management-cachename	No WebLogic-specific default.
coherence-session-expire-seconds	No WebLogic-specific default.
coherence-shutdown-delay-seconds	No WebLogic-specific default.
coherence-session-member-locking	If unspecified, defaults to `true`
coherence-session-thread-locking	No WebLogic-specific default.
coherence-session-strict-spec	No WebLogic-specific default.
coherence-sticky-sessions	If unspecified, defaults to `true`
coherence-distributioncontroller-class	If unspecified, defaults to `com.tangosol.coherence.servlet.AbstractHttpSessionCollection.HybridController`
coherence-scopecontroller-class	If unspecified, defaults to `com.tangosol.coherence.servlet.AbstractHttpSessionCollection.ApplicationScopeController`
coherence-preserve-attributes	If unspecified, defaults to `true`
coherence-local-session-cachename	No WebLogic-specific default.
coherence-local-attribute-cachename	No WebLogic-specific default.
coherence-session-overflow-cachename	No WebLogic-specific default.
coherence-attribute-overflow-threshold	No WebLogic-specific default.

Table 2–2 describes the generated HTTP session cookie parameters that can be configured in `weblogic.xml` or `weblogic-application.xml` file using the `<session-descriptor>` element.

Table 2–2 HTTP Session Cookie Parameters

Parameter Name	Default	Description
`cookie-comment`	null	Specifies the comment that identifies the session tracking cookie in the cookie file.
`cookie-domain`	null	Specifies the domain for which the cookie is valid. For example, setting `cookie-domain` to `.mydomain.com` returns cookies to any server in the `*.mydomain.com` domain. The domain name must have at least two components. Setting a name to `*.com` or `*.net` is not valid. If not set, this attribute defaults to the server that issued the cookie. For more information, see `Cookie.setDomain()` in the Servlet specification from Sun Microsystems.
`cookies-enabled`	true	Use of session cookies is enabled by default and is recommended, but you can disable them by setting this property to false. You might turn this option off to test.
`cookie-max-age-secs`	-1	Sets the life span of the session cookie, in seconds, after which it expires on the client. The default value is -1 (unlimited). For more information about cookies, see *Using Sessions and Session Persistence*.
`cookie-path`	null	Defines the session tracking cookie path. If not set, this attribute defaults to `/` (slash), where the browser sends cookies to all URLs served by WebLogic Server. You may set the path to a narrower mapping, to limit the request URLs to which the browser sends cookies.
`cookie-secure`	false	Tells the browser to only send the cookie back over an HTTPS connection. This ensures that the cookie ID is secure and should only be used on Web sites that use HTTPS. Session Cookies over HTTP no longer work if this feature is enabled. You should disable the `url-rewriting-enabled` element if you intend to use this feature.
`id-length`	52	Sets the size of the session ID. The minimum value is 8 bytes and the maximum value is `Integer.MAX_VALUE`. If you are writing a WAP application, you must use URL rewriting because the WAP protocol does not support cookies. Also, some WAP devices have a 128-character limit on URL length (including attributes), which limits the amount of data that can be transmitted using URL re-writing. To allow more space for attributes, use this attribute to limit the size of the session ID that is randomly generated by WebLogic Server. You can also limit the length to a fixed 52 characters, and disallow special characters, by setting the `WAPEnabled` attribute. For more information, see *URL Rewriting and Wireless Access Protocol* in Developing Web Applications for WebLogic Server.

In the WebLogic SPI module, the Coherence*Web configuration parameters listed in Table 2–3 are not controlled by Coherence*Web and must be specified as outlined in the table.

*Table 2–3 Coherence*Web Configuration Parameters which Must be Specified*

Parameter	Alternative configuration setting to be used
coherence-servletcontext-clustered	Not Supported.
coherence-servletcontext-cachename	Not Supported.
coherence-eventlisteners	Not Supported.
coherence-enable-sessioncontext	Not Supported.
coherence-session-cookies-enabled	This value is set by the WebLogic session-descriptor `cookies-enabled` element in `weblogic.xml` or `weblogic-application.xml`.
coherence-session-cookie-domain	This value is set by the WebLogic session-descriptor `cookie-domain` element in `weblogic.xml` or `weblogic-application.xml`.
coherence-session-cookie-path	This value is set by the WebLogic session-descriptor `cookie-path` element in `weblogic.xml` or `weblogic-application.xml`.
coherence-session-cookie-max-age	This value is set by the WebLogic session-descriptor `cookie-max-age-secs` element in `weblogic.xml` or `weblogic-application.xml`.
coherence-session-urlencode-enabled	Not Supported.
coherence-session-urlencode-name	Not Supported.
coherence-session-urldecode-bycontainer	Not Supported.
coherence-session-urlencode-bycontainer	Not Supported.
coherence-session-id-length	This value is set by the WebLogic session-descriptor `id-length` element in `weblogic.xml` or `weblogic-application.xml`.

Using SAML SSO with Coherence*Web

WebLogic Server provides basic single sign-on (SSO) functionality by default. To use SAML SSO functionality with Coherence*Web, you must modify the contents of the `saml2.war` Web application.

1. Backup the existing `saml2.war` in the WebLogic Server installation.
2. Un-jar the `saml2.war` into a temporary directory.

   The `saml2.war` is located in the *`$WL_SERVER_HOME`*`/server/lib` directory.
3. Create a `lib` directory and a `classes` directory under the `WEB-INF` directory.
4. Un-jar the `coherence-web-spi.war` to retrieve `coherence-web.jar` and `coherence-web-spi.jar`. Copy these two jars to the `/WEB-INF/lib` directory.
5. Copy the `session-cache-config.xml` file, located in the `/WEB-INF/classes` directory from the un-jarred `coherence-web-spi.war`, into the `/WEB-INF/classes` directory.
6. Place the `coherence.jar` in the appropriate location, based on the cluster node scoping you selected: application server-, EAR-, or WAR-scoped.
7. Add the code in Example 2–4 to the `/WEB-INF/web.xml` file:

Example 2–4 Enabling Coherence Web Sessions in web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
     <web-app>
     ...
     <context-param>
          <param-name>coherence-web-sessions-enabled</param-name>
          <param-value>true</param-value>
     </context-param>
     ...
     </web-app>
```

8. Re-assemble the `saml2.war` by using the `jar` command, for example:

   ```
   jar cvf saml2.war $tempdir
   ```

9. Backup the existing `saml2.war` in the WebLogic Server installation.
10. Replace the `saml2.war` in the WebLogic installation with the modified `saml2.war` file.

Known Limitations

By default, Coherence*Web creates a single HTTP Session across all Web applications for each client and scopes the session attributes to each Web application. This means that if a session is invalidated in one Web application, that same session is invalidated for *all* Web applications in WebLogic Server using Coherence*Web.

This functionality requires that the session cookie path is set to "/", making the same session cookie available to all Web applications. If you do not want this behavior, a potential work-around is to reduce the scope the session cookie by adding the entry illustrated in Example 2–5 to the `weblogic.xml` file in each Web application.

Example 2–5 Ensuring a Unique Session for Each Web Application

```
<weblogic-web-app>
     ...
     <session-descriptor>
          <cookie-path>[path of web-app context, for example
"/mainApp/subApp"]</cookie-path>
     </session-descriptor>
     ...
</weblogic-web-app>
```

This will ensure a unique session is created for each Web application. It is also possible to scope the session to all Web applications within an EAR file by setting the session cookie path to the context root of the deployed EAR.

This work-around will not work if you deploy an EAR or Web application with "/" as the context path, or if you require WebLogic SSO. WebLogic SSO requires that the session cookie path be set to "/".

3

Installing Coherence*Web on Other Application Servers

This chapter provides instructions on how to use the Coherence*Web WebInstaller to install Coherence*Web for Java EE applications on a variety of different application servers.

Before Proceeding: Consult the "Supported Web Containers" on page 1-2 to see if you need to perform any application server-specific installation steps.

When deploying Coherence*Web on WebLogic Server you now have two options:

- Use the WebInstaller approach described in this chapter
- Use the SPI-based installation for WebLogic Server 10.3 or later. This is described in Chapter 2, "Installing Coherence*Web on the WebLogic Server 10.3."

Installing Coherence*Web Using the WebInstaller

Coherence*Web can be enabled for Java EE applications on a number of different Web containers. To enable Coherence*Web, you must run the ready-to-deploy application through the automated Coherence*Web WebInstaller before deploying it. The automated installer prepares the application for deployment. It performs the installation process in two discrete steps: an inspect step and an install step. For more information on what the installer does during these steps, see "How the Coherence*Web Installer Instruments a Java EE Application" on page 3-7.

The installer can be run either from the Java command line or from Ant tasks. The Java command line method is described in the following sections. For Ant task-based installation, see "Coherence*Web WebInstaller Ant Task" on page 3-9.

Application Server-Specific Installation Instructions

All of the Web containers listed in "Supported Web Containers" on page 1-2 (with the exception of Oracle WebLogic Server 10.3) share the same general installation instructions. These instructions are described in "General Instructions for Installing Coherence*Web Session Management Module" on page 3-4.

A few of the Web containers, such as Oracle OC4J, Caucho, and WebLogic 10.x, require extra, container-specific steps that you must complete before starting the general

installation instructions. The following sections describe application server-specific installation steps.

- Installing on Oracle WebLogic 10.x
- Installing on Caucho Resin 3.0.x
- Installing on Caucho Resin 3.1.x
- Installing on Oracle OC4J 10.1.2.x

Installing on Oracle WebLogic 10.x

Complete the following steps to install the Coherence*Web Session Management Module into an Oracle WebLogic 10.x server:

1. From within the Coherence library directory, extract the `coherence-web.jar` from the `webInstaller.jar`:

   ```
   jar -xvf webInstaller.jar web-install/coherence-web.jar
   ```

   This will extract the `coherence-web.jar` file into a sub-directory named `web-install`. Use the following commands to move the `coherence-web.jar` file up one level into the library directory:

   On Windows:

   ```
   move web-install\coherence-web.jar .
   rmdir web-install
   ```

   On Unix:

   ```
   mv web-install/coherence-web.jar .
   rmdir web-install
   ```

2. For each WebLogic 10.x installation that will be running in the server cluster, update the libraries using the following command (note that it is broken up into multiple lines here only for formatting purposes; this is a single command entered on one line):

   ```
   java -cp coherence.jar;coherence-web.jar com.tangosol.coherence.servlet.
   WebPluginInstaller <wls-home-path> -install
   ```

   For example, on Windows:

   ```
   java -cp coherence.jar;coherence-web.jar com.tangosol.coherence.servlet.
   WebPluginInstaller C:\bea\weblogic\wlserver_10 -install
   ```

3. Follow the instructions described in "General Instructions for Installing Coherence*Web Session Management Module" on page 3-4 to complete the installation. Use the value `WebLogic/10.x` for the *server type*.

Installing on Caucho Resin 3.0.x

Complete the following steps to install the Coherence*Web Session Management Module into a Caucho Resin 3.0.x server:

1. From within the Coherence library directory, extract the `coherence-web.jar` from the `webInstaller.jar`:

   ```
   jar -xvf webInstaller.jar web-install/coherence-web.jar
   ```

This will extract the `coherence-web.jar` file into a subdirectory named `web-install`. Use the following commands to move the `coherence-web.jar` file up one level into the library directory:

On Windows:

```
move web-install\coherence-web.jar .
rmdir web-install
```

On UNIX:

```
mv web-install/coherence-web.jar .
rmdir web-install
```

2. For each Resin installation that will be running in the server cluster, update the libraries using the following command (note that it is broken up into multiple lines only for formatting purposes; this is a single command entered on one line):

```
java -cp coherence.jar;coherence-web.jar
 com.tangosol.coherence.servlet.WebPluginInstaller <resin-home-path> -install
```

For example, on Windows:

```
java -cp coherence.jar;coherence-web.jar
com.tangosol.coherence.servlet.WebPluginInstaller C:\opt\resin30 -install
```

3. Follow the instructions described in "General Instructions for Installing Coherence*Web Session Management Module" on page 3-4 to complete the installation. Use the value `Resin/3.0.x` for the *server type*.

Installing on Caucho Resin 3.1.x

Complete the following steps to install the Coherence*Web Session Management Module into a Caucho Resin 3.1.x server:

1. From within the Coherence library directory, extract the `coherence-web.jar` from the `webInstaller.jar`:

```
jar -xvf webInstaller.jar web-install/coherence-web.jar
```

This will extract the `coherence-web.jar` file into a subdirectory named `web-install`. Use the following commands to move the `coherence-web.jar` file up one level into the library directory:

On Windows:

```
move web-install\coherence-web.jar .
rmdir web-install
```

On UNIX:

```
mv web-install/coherence-web.jar .
rmdir web-install
```

2. For each Resin installation that will be running in the server cluster, update the libraries using the following command (note that it is broken up into multiple lines only for formatting purposes; this is a single command entered on one line):

```
java -cp coherence.jar;coherence-web.jar
com.tangosol.coherence.servlet.WebPluginInstaller <resin-home-path> -install
```

For example, on Windows:

```
java -cp coherence.jar;coherence-web.jar
```

```
com.tangosol.coherence.servlet.WebPluginInstaller C:\opt\resin31 -install
```

3. Follow the instructions described in "General Instructions for Installing Coherence*Web Session Management Module" on page 3-4 to complete the installation. Use the value `Resin/3.1.x` for the *server type*.

Installing on Oracle OC4J 10.1.2.x

Complete the following steps to install the Coherence*Web Session Management Module into an Oracle OC4J 10.1.2.x server:

1. From within the Coherence library directory, extract the `coherence-web.jar` from the `webInstaller.jar`:

   ```
   jar -xvf webInstaller.jar web-install/coherence-web.jar
   ```

   This will extract the `coherence-web.jar` file into a subdirectory named `web-install`. Use the following commands to move the `coherence-web.jar` file up one level into the `library` directory:

   On Windows:

   ```
   move web-install\coherence-web.jar .
   rmdir web-install
   ```

   On UNIX:

   ```
   mv web-install/coherence-web.jar .
   rmdir web-install
   ```

2. For each OC4J installation that will be running in the server cluster, update the libraries using the following command (note that it is broken up into multiple lines only for formatting purposes; this is a single command entered on one line):

   ```
   java -cp coherence.jar;coherence-web.jar
   com.tangosol.coherence.servlet.WebPluginInstaller <oc4j-home-path> -install
   ```

   For example, on Windows:

   ```
   java -cp coherence.jar;coherence-web.jar
   com.tangosol.coherence.servlet.WebPluginInstaller C:\opt\oracle1012\j2ee\home
   ```

3. Follow the instructions described in "General Instructions for Installing Coherence*Web Session Management Module" on page 3-4 to complete the installation. Use the value `Oracle/10.1.2.x` for the *server type*.

General Instructions for Installing Coherence*Web Session Management Module

You must complete the following steps to install Coherence*Web for a Java EE application on any of the Web containers listed under "Supported Web Containers" on page 1-2.

If you are installing Coherence*Web Session Management Module on an Oracle OC4J, Caucho, or WebLogic container, then you must complete certain container-specific installation steps before you start the general installation instructions. These container-specific installation steps are described in "Application Server-Specific Installation Instructions" on page 3-1.

To install Coherence*Web for the Java EE application you are deploying:

1. Make sure that the application directory and the `.ear` file or `.war` file are not being used or accessed by another process.

2. Change the current directory to the Coherence library directory (*%COHERENCE_HOME%*`\lib` on Windows and *$COHERENCE_HOME*`/lib` on UNIX).
3. Make sure that the paths are configured so that Java commands will run.
4. Complete the application inspection step by running the following command. Specify the full path to your application and the name of your server found in Table 1–1 (replacing the `<app-path>` and `<server-type>` with them in the command line below):

   ```
   java -jar webInstaller.jar <app-path> -inspect -server:<server-type>
   ```

   The system will create (or update, if it already exists), the `coherence-web.xml` configuration descriptor file for your Java EE application in the directory where the application is located. This configuration descriptor contains the default Coherence*Web settings for your application recommended by the installer.
5. You may proceed to the install step (Step 6) or review and modify the Coherence*Web settings based on your requirements, before running the install step.

   You modify the Coherence*Web settings by editing the `coherence-web.xml` descriptor. Appendix A, "Coherence*Web Configuration Parameters," describes the Coherence*Web settings that can be modified. Use the `param-name` and `param-value` subelements of `context-param` to enable the features you want.

   For example:

   – The setting inTable 3–1 will cluster all `ServletContext` ("global") attributes so that servers in a cluster will share the same values for those attributes, and will also receive the events specified by the Servlet Specification when those attributes change:

Table 3–1 Settings to Cluster ServletContext Attributes

Parameter	Value
param-name	coherence-servletcontext-clustered
param-value	true

   – The setting in Table 3–2 allows an application to enumerate all of the sessions that exist within the application, or to obtain any one of those sessions to examine or manipulate:

Table 3–2 Settings to Enumerate All Sessions in the Application

Parameter	Value
param-name	coherence-enable-sessioncontext
param-value	true

   – The setting in Table 3–3 enables you to increase the length of the `HttpSession` ID, which is generated using a `SecureRandom` algorithm; the length can be any value, although in practice it should be small enough to fit into a cookie or a URL (depending on how session IDs are maintained.) Increasing the length can decrease the chance of a session being purposefully hijacked:

Table 3–3 Settings to Increase Length of HTTPSession ID

Parameter	Value
param-name	coherence-session-id-length
param-value	32

– By default, the `HttpSession` ID is managed in a cookie. If the application supports URL encoding, set the option described in Table 3–4 to enable it:

Table 3–4 Settings to Support URI Encoding

Parameter	Value
param-name	coherence-session-urlencode-enabled
param-value	true

After double-checking that these changes have been made, save the file and exit the editor. Remember to return back to the Coherence library directory if you are working from a shell or command line.

6. Perform the Coherence*Web application installation step by running the following command, replacing `<app-path>` with the full path to your application:

```
java -jar webInstaller.jar <app-path> -install
```

The installer requires a valid `coherence-web.xml` configuration descriptor for its use in the same directory in which the application is located.

7. Deploy the updated application and verify that everything functions as expected, using the load balancer if necessary. Remember that the load balancer is intended only for testing and should not be used in a production environment.

Testing HTTP Session Management

Coherence comes with a light-weight software load balancer; it is intended only for testing purposes. The load balancer is very useful when testing functionality such as session management and is very easy to use.

1. Start multiple application server processes on one or more server machines, each running your application on a unique IP address and port combination.
2. Open a command (or shell) window.
3. Change the current directory to the Coherence library directory (*%COHERENCE_HOME%*`\lib` on Windows and *$COHERENCE_HOME*`/lib` on UNIX).
4. Make sure that paths are configured so that Java commands will run.
5. Start the software load balancer with the following command lines (each of these command lines makes the application available on the default HTTP port, which is port 80).

   For example, to test load-balancing locally on one machine with two application server instances on ports `7001` and `7002`:

```
java -jar coherence-loadbalancer.jar localhost:80 localhost:7001 localhost:7002
```

   To run the load-balancer locally on a machine named `server1` that load balances to port `7001` on `server1`, `server2`, and `server3`:

```
java -jar coherence-loadbalancer.jar server1:80 server1:7001 server2:7001
```

```
server3:7001
```

Assuming the above command line, an application that previously was accessed with the URL `http://server1:7001/my.jsp` would now be accessed with the URL `http://server1:80/my.jsp` or just `http://server1/my.jsp`.

Note: Make sure that your application uses only relative re-directs or the address of the load-balancer.

Table 3–5 describes the command line options for the load balancer:

Table 3–5 Load Balancer Command Line Options

Option	Description
backlog	Sets the TCP/ IP accept backlog option to the specified value, for example: `-backlog=64`
random	Specifies the use of a random load-balancing algorithm (default).
roundrobin	Specifies the use of a round-robin load-balancing algorithm
threads	Uses the specified number of request/ response thread pairs (so the total number of additional daemon threads will be two times the specified value), for example: `-threads=64`

How the Coherence*Web Installer Instruments a Java EE Application

During the inspect step, the Coherence*Web WebInstaller performs the following tasks:

1. Generates a template `coherence-web.xml` configuration file that contains basic information about the application and target Web container along with a set of default Coherence*Web configuration context parameters appropriate for the target Web container. See Appendix A, "Coherence*Web Configuration Parameters" for descriptions of all possible parameters.

   If an existing `coherence-web.xml` configuration file exists (for example, from a previous run of the Coherence*Web Installer), the context parameters in the existing file are merged with those in the generated template.

2. Enumerates the JSPs from each Web application in the target Java EE application and add information about each JSP to the `coherence-web.xml` configuration file.

3. Enumerates the TLDs from each Web application in the target Java EE application and adds information about each TLD to the `coherence-web.xml` configuration file.

During the install step, the Coherence*Web WebInstaller performs the following tasks:

1. Creates a backup of the original Java EE application so that it can be restored during the uninstall step.

2. Adds the Coherence*Web configuration context parameters generated in Step (1) of the inspect step to the `web.xml` descriptor of each Web application contained in the target Java EE application.

3. Unregisters any application-specific `ServletContextListener`, `ServletContextAttributeListener`, `ServletRequestListener`, `ServletRequestAttributeListener`, `HttpSessionListener`, and

`HttpSessionAttributeListener` classes (including those registered by TLDs) from each Web application.

4. Registers a Coherence*Web `ServletContextListener` in each `web.xml` descriptor. At run time, the Coherence*Web `ServletContextListener` will propagate each `ServletContextEvent` to each application-specific `ServletContextListener`.
5. Registers a Coherence*Web `ServletContextAttributeListener` in each `web.xml` descriptor. At run time, the Coherence*Web `ServletContextAttributeListener` will propagate each `ServletContextAttributeEvent` to each application-specific `ServletContextAttributeListener`.
6. Wraps each application-specific `Servlet` declared in each `web.xml` descriptor with a Coherence*Web `SessionServlet`. At run time, each Coherence*Web `SessionServlet` will delegate to the wrapped `Servlet`.
7. Adds the following directive to each JSP enumerated in Step (2) of the inspect step:

```
<%@ page extends="com.tangosol.coherence.servlet.api22.JspServlet" %>
```

During the uninstall step, the Coherence*Web WebInstaller replaces the instrumented Java EE application with the backup of the original version created in Step (1) of the install process.

Installing Coherence*Web into Applications using Java EE Security

> **Note:** This section does not apply to the native WebLogic SPI implementation of Coherence*Web. It applies only if you are using the WebInstaller to install Coherence*Web into an application that uses Java EE security.

If you want to install Coherence*Web into an application that uses Java EE security, you must follow these additional steps during installation:

1. Enable Coherence*Web session cookies.

   See the `coherence-session-cookies-enabled` configuration element in Table A–1 for additional details.
2. Change the Coherence*Web session cookie name to a name which is different from the one used by the target Web container.

   By default, most containers use `JSESSIONID` for the session cookie name, so a good choice for the Coherence*Web session cookie name is `CSESSIONID`. See the `coherence-session-cookie-name` configuration element in Table A–1 for additional details.
3. Enable session replication for the target Web container.

   If session replication is not enabled, or the container does not support a form of session replication, then you will be forced to re-authenticate to the Web application during failover. See your Web container's documentation for instructions on enabling session replication.

This configuration will cause two sessions to be associated with a given authenticated user:

- A Coherence*Web session which will contain all session data created by the Web application.
- A session created by the Web container during authentication which will only store information necessary to identify the user.

Coherence*Web WebInstaller Ant Task

The Coherence*Web WebInstaller Ant task allows you to run the installer from within your existing Ant build files.

This section contains the following information:

- Using the WebInstaller Ant task
- Configuring the WebInstaller Ant Task
- WebInstaller Ant Task Examples

Using the WebInstaller Ant task

To use the Coherence*Web WebInstaller Ant task, add the task import statement illustrated in Example 3–1 to your Ant build file. In this example, `${coherence.home}` refers to the root directory of your Coherence installation.

Example 3–1 Task Import Statement for Coherence*Web WebInstaller

```
<taskdef name="cwi" classname="com.tangosol.coherence.misc.CoherenceWebAntTask">
    <classpath>
        <pathelement location="${coherence.home}/lib/webInstaller.jar"/>
    </classpath>
</taskdef>
```

The following procedure describes the basic process of installing Coherence*Web into a Java EE application from an Ant build.

1. Build your Java EE application as you normally would.
2. Run the Coherence*Web Ant task with the `operations` attribute set to `inspect`.
3. Make any necessary changes to the generated Coherence*Web XML descriptor.
4. Run the Coherence*Web Ant task with the `operations` attribute set to `install`.

If you are performing iterative development on your application (such as modifying JSPs, Servlets, static resources, and so on), use the following installation process:

1. Run the Coherence*Web Ant task with the `operations` attribute set to `uninstall`, the `failonerror` attribute set to `false`, and the `descriptor` attribute set to the location of the previously generated Coherence*Web XML descriptor (from Step 2 above).
2. Build your Java EE application as you normally would.
3. Run the Coherence*Web Ant task with the `operations` attribute set to `inspect`, `install` and the `descriptor` attribute set to the location of the previously generated Coherence*Web XML descriptor (from Step 2 above).

To change the Coherence*Web configuration settings of a Java EE application that already has Coherence*Web installed, use this procedure:

1. Run the Coherence*Web Ant task with the `operations` attribute set to `uninstall` and the `descriptor` attribute set to the location of the Coherence*Web XML descriptor for the Java EE application.

2. Change the necessary configuration parameters in the Coherence*Web XML descriptor.
3. Run the Coherence*Web Ant task with the `operations` attribute set to `install` and the `descriptor` attribute set to the location of the modified Coherence*Web XML descriptor (from Step 2).

Configuring the WebInstaller Ant Task

Table 3–6 describes the attributes that can be used with the Coherence*Web WebInstaller Ant Task.

Table 3–6 Coherence*Web WebInstaller Ant Task Attributes

Attribute	Description	Required?
app	Path to the target Java EE application. This can be a path to a WAR file, an EAR file, an expanded WAR directory, or an expanded EAR directory.	Yes, if the `operations` attribute is set to any value other than `version`.
backup	Path to a directory that will hold a backup of the original target Java EE application. This attribute defaults to the directory that contains the Java EE application.	No
descriptor	Path to the Coherence*Web XML descriptor. This attribute defaults to `coherence-web.xml` in the directory that contains the target Java EE application.	No
failonerror	Stop the Ant build if the Coherence*Web installer exits with a status other than 0. The default is `true`.	No
nowarn	Suppress warning messages. This attribute can be either `true` or `false`. The default is `false`.	No
operations	A comma- or space-separated list of operations to perform; each operation must be one of `inspect`, `install`, `uninstall`, or `version`.	Yes
server	The alias of the target Java EE application server.	No
touch	Touch JSPs and TLDs that are modified by the Coherence*Web installer. This attribute can be either `true`, `false`, or `M/d/y h:mm a`' The default is `false`.	No
verbose	Show verbose output. This attribute can be either `true` or `false`. The default is `false`.	No

WebInstaller Ant Task Examples

- Inspect the `myWebApp.war` Web application and generate a Coherence*Web XML descriptor called `my-coherence-web.xml` in the current working directory:

```
<cwi app="myWebApp.war" operations="inspect" descriptor="my-coherence-web.
xml"/>
```

- Install Coherence*Web into the `myWebApp.war` Web application using the Coherence*Web XML descriptor called `my-coherence-web.xml` found in the current working directory:

```
<cwi app="myWebApp.war" operations="install" descriptor="my-coherence-web.
xml"/>
```

- Uninstall Coherence*Web from the `myWebApp.war` Web application:

```
<cwi app="myWebApp.war" operations="uninstall">
```

- Install Coherence*Web into the `myWebApp.war` Web application located in the `/dev/myWebApp/build` directory using the Coherence*Web XML descriptor called `my-coherence-web.xml` found in the `/dev/myWebApp/src` directory, and place a backup of the original Web application in the `/dev/myWebApp/work` directory:

  ```
  <cwi app="/dev/myWebApp/build/myWebApp.war" operations="install"
  descriptor="/dev/myWebApp/src/my-coherence-web.xml"
  backup="/dev/myWebApp/work"/>
  ```

- Install Coherence*Web into the `myWebApp.war` Web application located in the `/dev/myWebApp/build` directory using the Coherence*Web XML descriptor called `coherence-web.xml` found in the `/dev/myWebApp/build` directory. If the Web application has not already been inspected (that is, `/dev/myWebApp/build/coherence-web.xml` does not exists), inspect the Web application prior to installing Coherence*Web:

  ```
  <cwi app="/dev/myWebApp/build/myWebApp.war" operations="inspect,install"/>
  ```

- Reinstall Coherence*Web into the `myWebApp.war` Web application located in the `/dev/myWebApp/build` directory using the Coherence*Web XML descriptor called `my-coherence-web.xml` found in the `/dev/myWebApp/src` directory:

  ```
  <cwi app="/dev/myWebApp/build/myWebApp.war" operations="uninstall,install"
  descriptor="/dev/myWebApp/src/my-coherence-web.xml"/>
  ```

4

Coherence*Web Session Management Features

Coherence*Web can be configured in a number of ways to meet the demands of your environment. Consequently, you may need to change some of the default configuration options. The purpose of this chapter is to provide an in-depth look at the features that Coherence*Web supports so that you can make the appropriate configuration and deployment decisions.

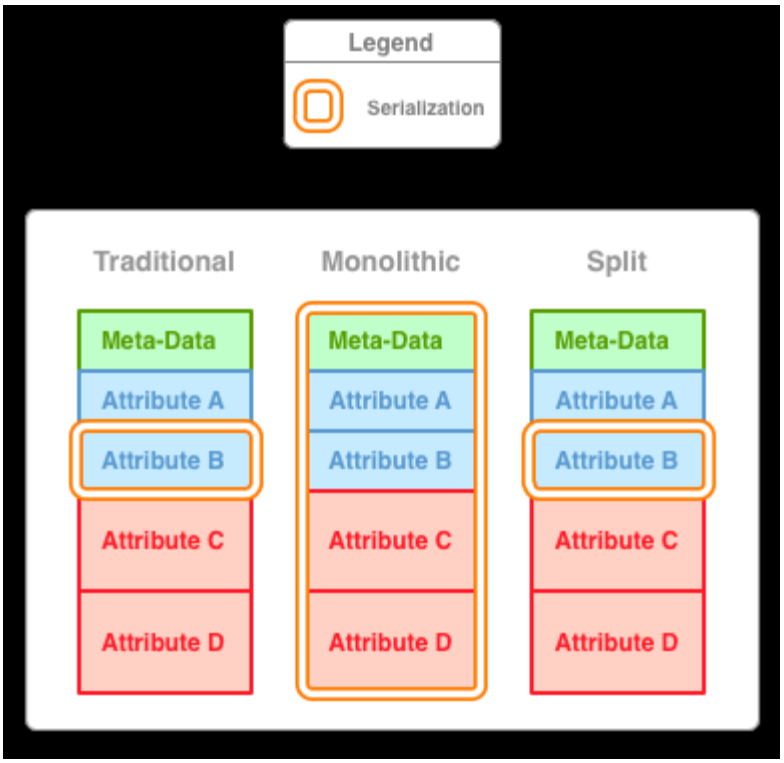
- Session Models
- Session and Session Attribute Scoping
- Cluster Node Isolation
- Session Locking Modes
- Deployment Topologies
- Managing and Monitoring Applications with JMX

Session Models

A session model describes how Coherence*Web physically represents and stores session state in Coherence. Coherence*Web supports a flexible data management model for session state. The session state is managed by an `HttpSessionModel` object, and the list of all sessions is managed by an `HttpSessionCollection` object. Coherence*Web includes these different session model implementations out of the box:

- Traditional Model—Stores all session state as a single entity but serializes and deserializes attributes individually.
- Monolithic Model—Stores all session state as a single entity, serializing and deserializing all attributes as a single operation.
- Split Model—Extends the Traditional Model but separates the larger session attributes into independent physical entities.

Figure 4–1 Traditional, Monolithic, and Split Session Models

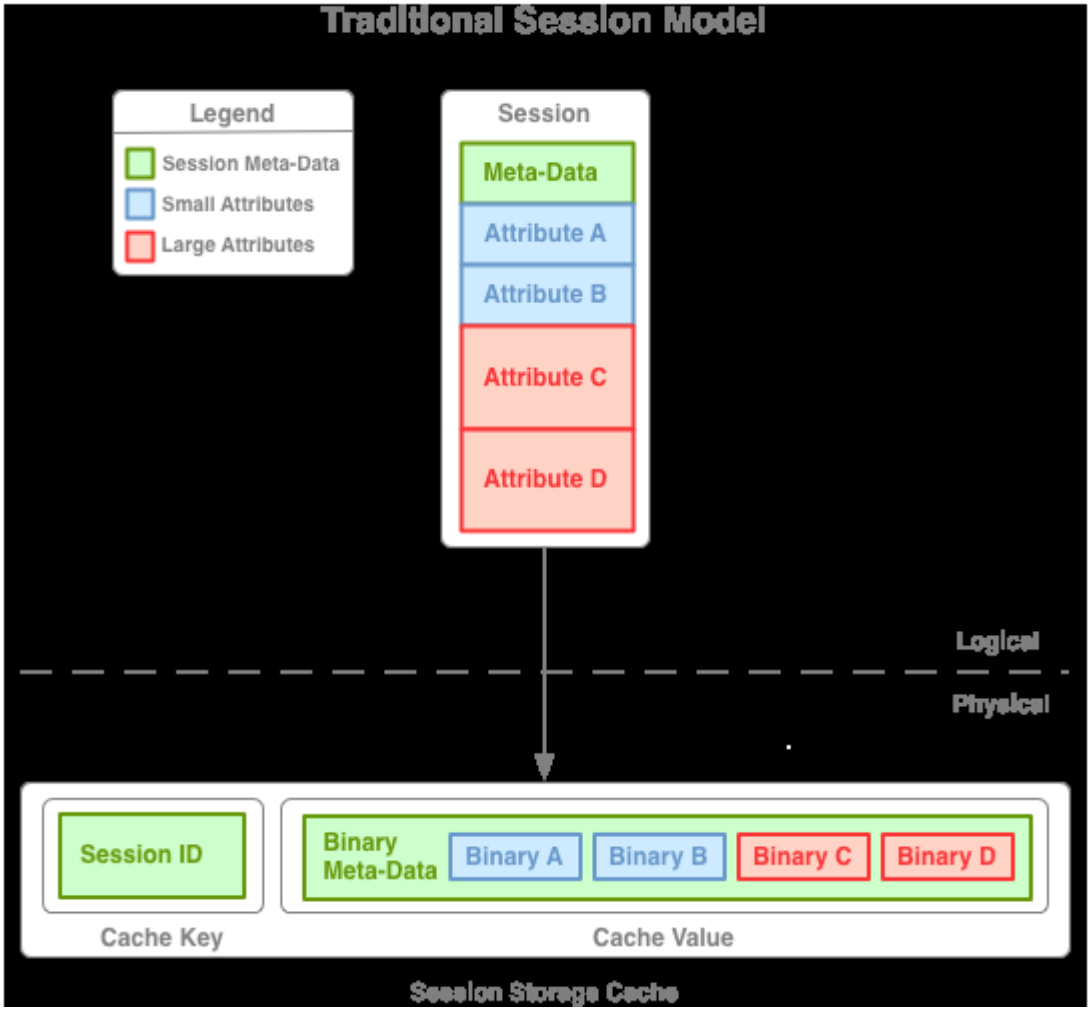


Traditional Model

`TraditionalHttpSessionModel` and `TraditionalHttpSessionCollection` manage all of the HTTP session data for a particular session in a single Coherence cache entry, but manage each HTTP session attribute (particularly, its serialization and deserialization) separately.

This model is suggested for applications with relatively small HTTP session objects (10KB or less) that do not have issues with object-sharing between session attributes. (Object-sharing between session attributes occurs when multiple attributes of a session have references to the same exact object, meaning that separate serialization and deserialization of those attributes will cause multiple instances of that shared object to exist when the HTTP session is later deserialized.)

Figure 4–2 Traditional Session Model

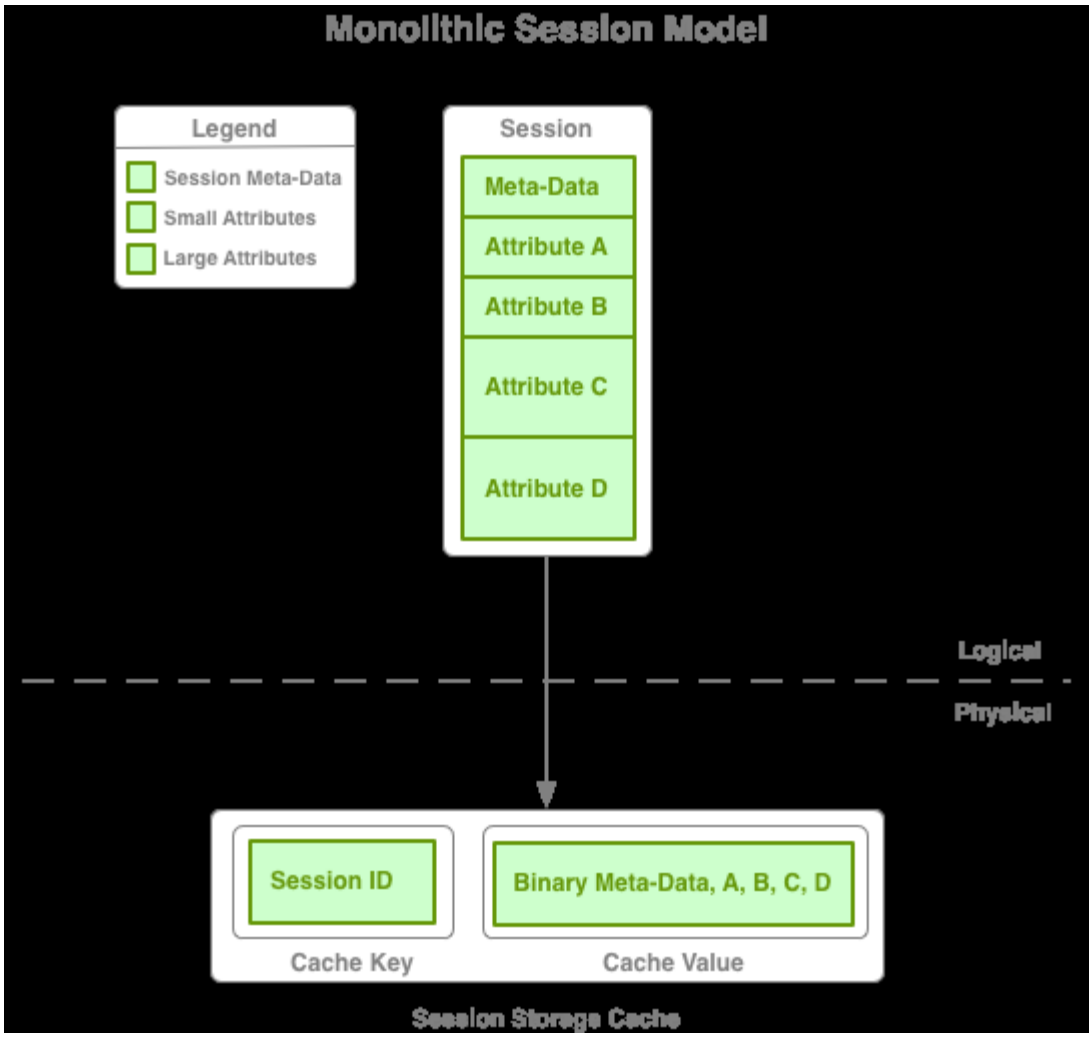


Monolithic Model

`MonolithicHttpSessionModel` and `MonolithicHttpSessionCollection` are similar to the Traditional Model, except that they solve the shared object issue by serializing and deserializing all attributes together in a single object stream.

As a result, the Monolithic Model is often less performant than the Traditional Model.

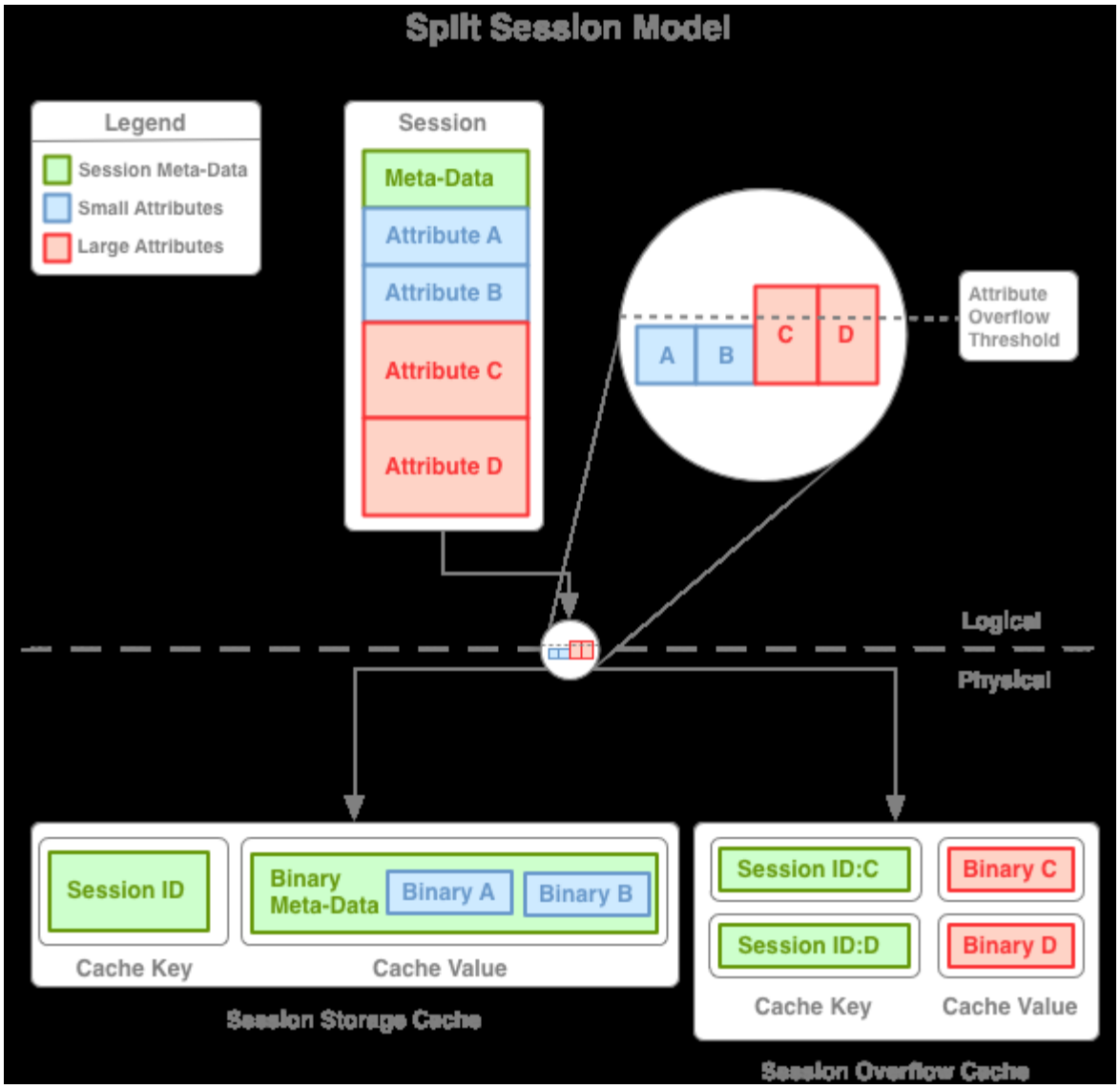
Figure 4–3 Monolithic Session Model



Split Model

`SplitHttpSessionModel` and `SplitHttpSessionCollection` manage the core HTTP session data such as the session ID, creation time, last access time, and so on, with all of the small session attributes in the same manner as the Traditional Model, thus ensuring high performance by keeping that block of session data small. All large attributes are split out into separate cache entries to be managed individually, thus supporting very large HTTP session objects without unduly increasing the amount of data that must be accessed and updated within the cluster on each request. In other words, only the large attributes that are modified within a particular request will incur any network overhead for their updates, and (because it uses Near Caching) the Split Model generally does not incur any network overhead for accessing either the core HTTP session data or any of the session attributes.

Figure 4–4 Split Session Model



Session Model Recommendations

- The Split Model is the recommended session model for most applications.
- The Traditional Model may be more optimal for applications that are known to have small HTTP session objects.
- The Monolithic Model is designed to solve a specific class of problems related to multiple session attributes that have references to the same shared object, and that must maintain that object as a shared object.

Session Management for Clustered Applications in *Getting Started with Oracle Coherence*, provides information on the behavior of these models in a clustered environment.

> **Note:** For configuration information, see Appendix A, "Coherence*Web Configuration Parameters."

Session and Session Attribute Scoping

Coherence*Web allows fine-grained control over how both session data and session attributes are scoped (or "shared") across application boundaries:

Session Scoping

Coherence*Web allows session data to be shared by different Web applications deployed in the same or different Web containers. To do so, you must correctly configure the Coherence*Web cookie context parameters and make the classes of objects stored in session attributes available to each Web application.

If you are using cookies to store session IDs (that is, you are not using URL rewriting), you must set the `coherence-session-cookie-path` context parameter to a common context path of all Web applications that share session data. For example, if you want to share session data between two Web applications registered under the contexts paths `/web/HRPortal` and `/web/InWeb`, you should set the `coherence-session-cookie-path` parameter to `/web`. On the other hand, if the two Web applications are registered under the context paths `/HRPortal` and `/InWeb`, you should set the `coherence-session-cookie-path` parameter to `/`.

If the Web applications that you would like to share session data are deployed on different Web containers running on different machines (that are not behind a common load balancer), you must also set the `coherence-session-cookie-domain` parameter to a domain shared by the machines. For example, if you would like to share session data between two Web applications running on `server1.mydomain.com` and `server2.mydomain.com`, you must set the `coherence-session-cookie-domain` parameter to `.mydomain.com`.

To correctly serialize or deserialize objects stored in shared sessions, the classes of all objects stored in session attributes must be available to Web applications that share session data. For Web applications deployed on different containers, the classes may be placed in either the Web container or Web application classpath; however, for applications deployed in the same Web container, the classes must be placed in the Web container classpath. This is due to the fact that most containers load each Web application using a separate `ClassLoader`.

Note: For advanced use cases where EAR cluster node-scoping or application server JVM cluster scoping is employed *and* you do not want session data shared across individual Web applications see "Preventing Web Applications from Sharing Session Data".

Preventing Web Applications from Sharing Session Data

Sometimes you may want to explicitly prevent HTTP session data from being shared by different Java EE applications that participate in the same Coherence cluster. For example, assume you have two applications `HRPortal` and `InWeb` that share cached data in their EJB tiers but utilize different session data. In this case, it is desirable for both applications to be part of the same Coherence cluster, but undesirable for both applications to use the same clustered service for session data.

To prevent different Java EE applications from sharing session data, specify a unique session cache service name for each application:

1. Locate the `<service-name/>` parameters in each `session-cache-config.xml` file found in your application.
2. Set the parameters to a unique value for each application.

This will force each application to use a separate clustered service for session data.

3. Save the modified `session-cache-config.xml` files.

Example 4–1 illustrates a sample `session-cache-config.xml` file for an `HRPortal` application. To prevent the `HRPortal` application from sharing session data with the `InWeb` application, rename the `<service-name>` parameter for the replicated scheme to be `ReplicationSessionsMiscHRP`. Rename the `<service-name>` parameter for the distributed schemes to be `DistributedSessionsHRP`.

Example 4–1 Configuration to Prevent Applications from Sharing Session Data

```
<replicated-scheme>
  <scheme-name>default-replicated</scheme-name>
  <service-name>ReplicatedSessionsMisc</service-name> // rename this to
ReplicatedSessionsMiscHRP
  <backing-map-scheme>
    <class-scheme>
      <scheme-ref>default-backing-map</scheme-ref>
    </class-scheme>
  </backing-map-scheme>
</replicated-scheme>

<distributed-scheme>
  <scheme-name>session-distributed</scheme-name>
  <service-name>DistributedSessions</service-name> // rename this to
DistributedSessionsHRP
  <lease-granularity>member</lease-granularity>
  <backing-map-scheme>
    <class-scheme>
      <scheme-ref>default-backing-map</scheme-ref>
    </class-scheme>
  </backing-map-scheme>
</distributed-scheme>

<distributed-scheme>
  <scheme-name>session-certificate</scheme-name>
  <service-name>DistributedSessions</service-name> // rename this to
DistributedSessionsHRP
  <lease-granularity>member</lease-granularity>
  <backing-map-scheme>
    <local-scheme>
      <scheme-ref>session-certificate-autoexpiring</scheme-ref>
    </local-scheme>
  </backing-map-scheme>
</distributed-scheme>
```

Keeping Session Cookies Separate

If you are using cookies to store session IDs, you must make sure that session cookies created by one application are not propagated to another application. To do this, you must set each application's session cookie domain and path in their `web.xml` file. The context parameter `coherence-session-cookie-path` sets the context path for a Web application. To prevent cookies from being propagated, be sure that no two applications share the same context path.

For example, assume you have two Web applications registered under the contexts paths `/web/HRPortal` and `/web/InWeb`. To prevent the Web applications from sharing session data through cookies, set the `coherence-session-cookie-path`

parameter in one application's `web.xml` file to `/web/HRPortal`; set the parameter in the other application's `web.xml` file to `/web/InWeb`.

If your applications are deployed on different Web containers running on separate machines, then you can set the context parameter `coherence-session-cookie-domain` to ensure that they are not in the same domain.

For example, assume you have two Web applications running on `server1.mydomain.com` and `server2.mydomain.com`. To prevent session cookies from being shared between them, then set the `coherence-session-cookie-domain` parameter in one application's `web.xml` file to `server1.mydomain.com`; set the parameter in the other application's `web.xml` file to `server2.mydomain.com`.

Session Attribute Scoping

In the case where sessions are shared across Web applications there are many instances where the application may want to scope individual session attributes so that they are either globally visible (that is, all Web applications can see and modify these attributes) or scoped to an individual Web application (that is, not visible to any instance of another application).

Coherence*Web provides the ability to control this behavior by using the `AttributeScopeController` interface. This optional interface is used to selectively scope attributes in cases when a session may be shared across more than one application. This enables different applications to potentially use the same attribute names for application-scope state without accidentally reading, updating, or removing other applications' attributes. In addition to having application-scoped information in the session, it allows the session to contain global (unscoped) information that is readable, updatable, and removable by any of the applications that share the session.

There are two implementations of this interface available out of the box: the `ApplicationScopeController` and the `GlobalScopeController`.

Cluster Node Isolation

When using Coherence*Web there are a number of deployment options to consider, one of which is the concept of cluster node isolation.

This option determines:

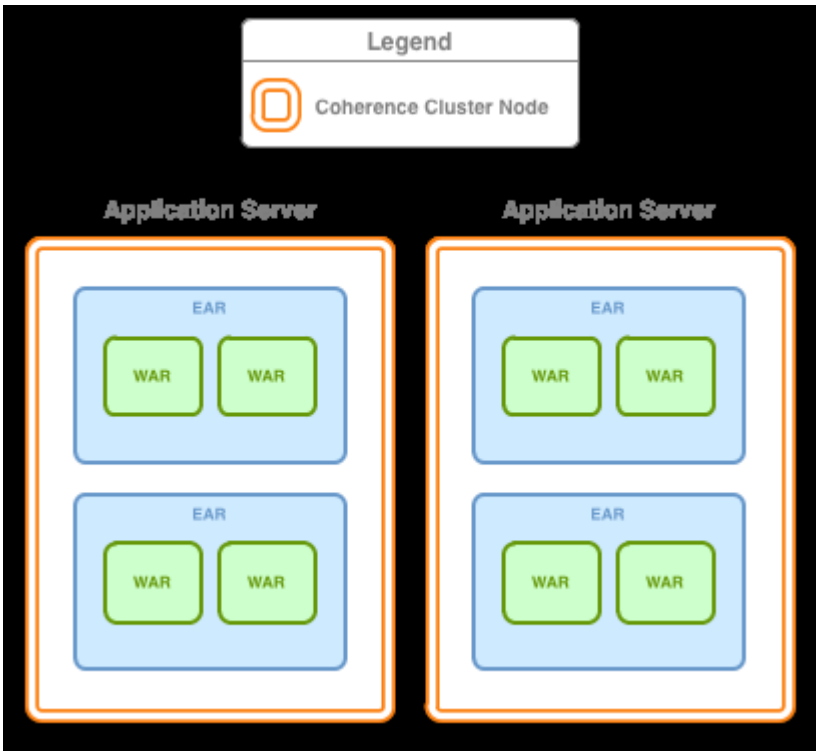
- The number of Coherence nodes that are created within an application server JVM.
- Where the Coherence library is deployed.

Applications can be application server-scoped, EAR-scoped, or WAR-scoped. This section describes these options. For detailed information on the XML configuration for each of these options, see "Packaging Applications and Configuring Cluster Nodes" on page 2-4.

Application Server-Scoped Cluster Nodes

With this configuration, all deployed applications in a container using Coherence*Web will be part of one Coherence node. This configuration will produce the smallest number of Coherence nodes in the cluster (one for each Web container JVM) and since the Coherence library (`coherence.jar`) is deployed in the container's classpath, only one copy of the Coherence classes will be loaded into the JVM. This minimizes the use of resources. On the other hand, since all applications are using the same cluster node, all applications will be affected if one application misbehaves.

Figure 4–5 Application Server-Scoped Cluster



Requirements for using this configuration are:

- Each deployed application must use the same version of Coherence and participate in the same cluster.
- Objects placed in the HTTP session must have their classes in the container's classpath.

The XML configuration for application server-scoped cluster nodes is described in "Packaging and Configuring Application Server-Scoped Cluster Nodes" on page 2-5.

Note: The application server-scoped cluster node configuration should be considered very carefully and *never* used in environments where the interaction between applications is unknown or unpredictable.

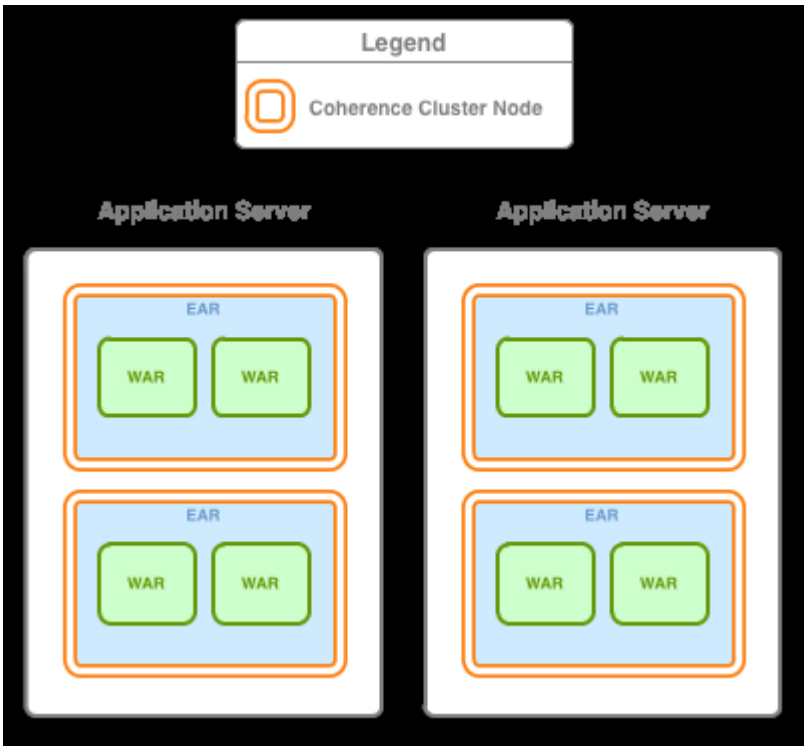
An example of such an environment may be a deployment where multiple application groups are deploying applications written independently, without carefully coordinating and enforcing their conventions and naming standards. With this configuration, all applications are part of the same cluster and the likelihood of collisions between namespaces for caches, services and other configuration settings is quite high and may lead to unexpected results.

For these reasons, Oracle Coherence strongly recommends that you use EAR-scoped and WAR-scoped cluster node configurations. If you are in doubt as to which deployment topology to choose, or if this warning applies to your deployment, then ***do not*** choose the application server-scoped cluster node configuration.

EAR-Scoped Cluster Nodes

With this configuration, all deployed applications within each EAR will be part of one Coherence node. This configuration will produce the next smallest number of Coherence nodes in the cluster (one for each deployed EAR that uses Coherence*Web). Since the Coherence library (`coherence.jar`) is deployed in the application's classpath, only one copy of the Coherence classes will be loaded for each EAR. Since all Web applications in the EAR use the same cluster node, all Web applications in the EAR will be affected if one of the Web applications misbehaves.

Figure 4–6 EAR-Scoped Cluster



EAR-scoped cluster nodes reduce the deployment effort as no changes to the application server classpath are required. This option is also ideal if you plan on deploying only one EAR to an application server.

Requirements for using this configuration are:

- The Coherence library (`coherence.jar`) must be deployed as part of the EAR file and listed as a Java module in `META-INF/application.xml`.
- Objects placed into the HTTP session must have their classes deployed as a Java EAR module in a similar fashion.

The XML configuration for EAR-scoped cluster nodes is described in"Packaging and Configuring EAR-Scoped Cluster Nodes" on page 2-5.

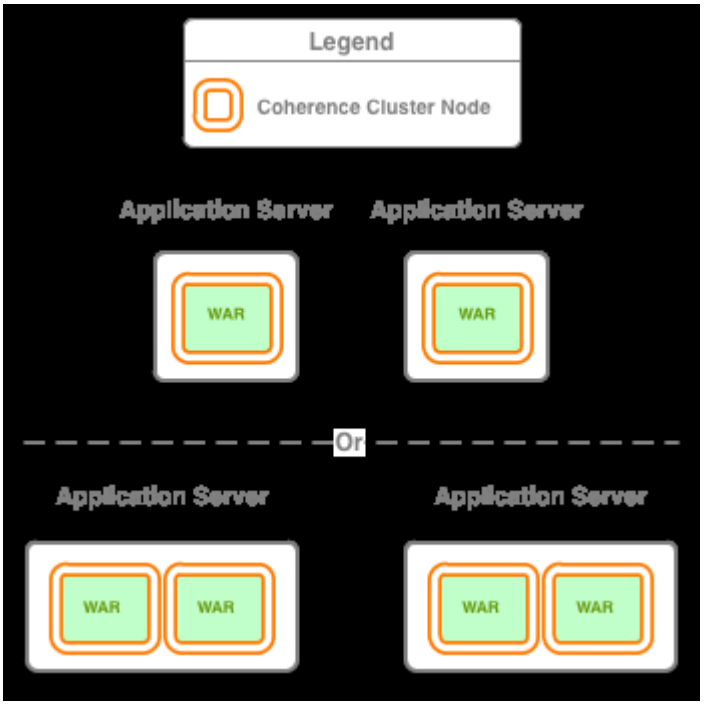
WAR-Scoped Cluster Nodes

With this configuration, each deployed Web application will be its own Coherence node. This configuration will produce the largest number of Coherence nodes in the cluster (one for each deployed WAR that uses Coherence*Web) and since the Coherence library (`coherence.jar`) is deployed in the Web application's classpath, there will be as many copies of the Coherence classes loaded as there are deployed

WARs. This results in the largest resource utilization out of the three options. However, since each deployed Web application is its own cluster node, Web applications are completely isolated from other potentially misbehaving Web applications.

WAR scoped cluster nodes reduce the deployment effort as no changes to the application server classpath are required. This option is also ideal if you plan on deploying only one WAR to an application server.

Figure 4–7 WAR-Scoped Clusters



Requirements for using this configuration are:

- The Coherence library (`coherence.jar`) must be deployed as part of the WAR file (usually in `WEB-INF/lib`).
- Objects placed into the HTTP session must have their classes deployed as part of the WAR file (in `WEB-INF/lib` or `WEB-INF/classes`).

The XML configuration for WAR-scoped cluster nodes is described in "Packaging and Configuring WAR-Scoped Cluster Nodes" on page 2-6.

Session Locking Modes

Oracle Coherence provides these configuration options for concurrent access to HTTP sessions.

- Optimistic Locking (Default)—Allows concurrent access to a session by multiple threads in a single JVM or multiple JVMs while prohibiting concurrent modification
- Member Locking—Allows concurrent access and modification of a session by multiple threads in the same JVM while prohibiting concurrent access by threads in different JVMs.

- Thread Locking—Prohibits concurrent access and modification of a session by multiple threads in a single JVM or multiple JVMs.

For more information on the parameters described in this section, see Appendix A, "Coherence*Web Configuration Parameters."

Optimistic Locking (Default)

The Optimistic Locking mode allows multiple Web container threads in one or more JVMs to access the same session concurrently. This setting does not use explicit locking; rather an optimistic approach is used to detect and prevent concurrent updates upon completion of an HTTP request that modifies the session. When Coherence*Web detects a concurrent modification, a `ConcurrentModificationException` is thrown to the application; therefore an application must be prepared to handle this exception in an appropriate manner.

This mode can be configured by setting the `coherence-session-member-locking` parameter to `false`.

Member Locking

The Member Locking mode allows multiple Web container threads in the same JVM to access and modify the same session concurrently, but prohibits concurrent access by threads in different JVMs. This is accomplished by acquiring a member-level lock for an HTTP session at the beginning of a request and releasing the lock upon completion of the request. For more information on member-level locks, see `<lease-granularity>` in the *distributed-scheme* section of the *Developer's Guide for Oracle Coherence*.

This mode can be configured by setting the `coherence-session-member-locking` parameter to `true`.

Thread Locking

The Thread Locking mode restricts access (and modification) to a session to a single thread in a single JVM at a time. This is accomplished by acquiring both a member level and thread-level lock for an HTTP session at the beginning of a request and releasing both locks upon completion of the request. For more information on member-level locks, see `<lease-granularity>` in the *distributed-scheme* section of the *Developer's Guide for Oracle Coherence*.

This mode can be configured by setting the `coherence-session-thread-locking` parameter to `true`. Note that setting this to `true` will imply a setting of `true` for `coherence-session-member-locking`.

Using Locking in HTTP Sessions

Enabling Member or Thread Locking for HTTP session access indicates that Coherence*Web will acquire a cluster-wide lock for every HTTP request that requires access to a session. By default, threads that attempt to access a locked session will block until the lock can be acquired. If you want to enable a timeout for lock acquisition, you can configure it by using the `tangosol.coherence.servlet.lock.timeout` system property in the container's startup script (for example `-Dtangosol.coherence.servlet.lock.timeout=30s`).

Many Web applications do not have such a strict concurrency requirement. For these applications, using the Optimistic Locking mode has the following advantages:

- The overhead of obtaining and releasing cluster wide locks for every HTTP request is eliminated.
- Requests can be load balanced away from failing or unresponsive JVMs to healthy JVMs without requiring the unresponsive JVM to release the cluster-wide lock on the session.

Enabling Sticky Session Optimizations

If Member or Thread Locking is a requirement for a Web application that will be behind a sticky load balancer, Coherence*Web provides an optimization for obtaining the cluster-wide lock required for HTTP session access. By definition, a sticky load balancer will attempt to route a request for a given session to the JVM that initially created it. The sticky session optimizations take advantage of this behavior by retaining the cluster-wide lock for a session until the session expires. If a request for the session lands on another JVM, that JVM will ask the JVM that owns the lock to release it as soon as possible. This is implemented using an invocation service. For more information, see the `SessionOwnership` entry in Table B–2.

Sticky session optimization can be enabled by setting the `coherence-sticky-sessions` parameter to `true`.

Deployment Topologies

Coherence*Web supports most of the same deployment topologies that Coherence does including in-process, out-of-process (that is, client/server deployment), and bridging clients and servers over Coherence*Extend. The major supported deployment topologies are described in the following sections.

- Out-of-Process
- Out-of-Process with Coherence*Extend
- In-Process

Out-of-Process

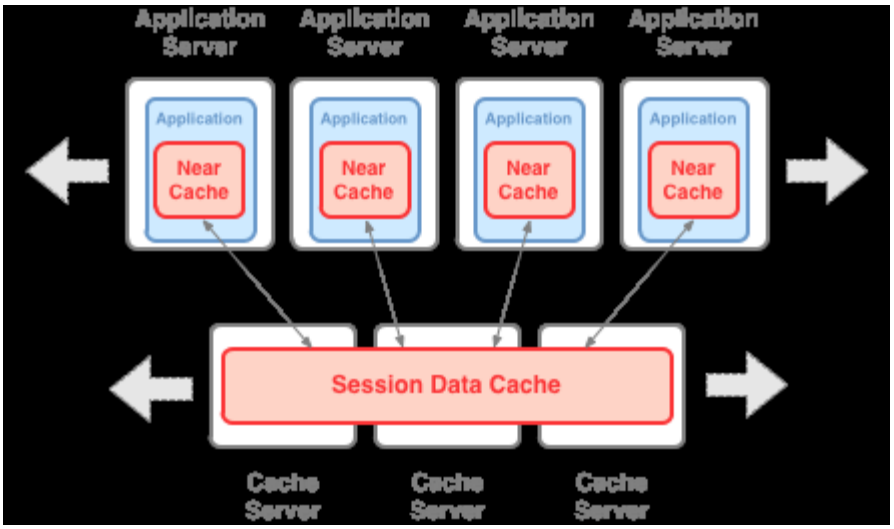
In the Out of Process deployment topology, the application servers (that is, application server tier) are configured as cache clients (that is, `tangosol.coherence.distributed.localstorage=false`) and there are dedicated JVMs running as cache servers, physically storing and managing the clustered data.

This approach has these benefits:

- Session data storage is off-loaded from the application server tier to the cache server tier. This reduces heap usage, garbage collection times, and so on.
- It allows for the two tiers to be scaled independently of one another. If more application processing power is needed, just start more application servers. If more session storage capacity is needed, just start more cache servers.

The Out-of-Process topology is the default recommendation of Oracle Coherence due to its flexibility.

Figure 4–8 Out of Process Deployment Topology

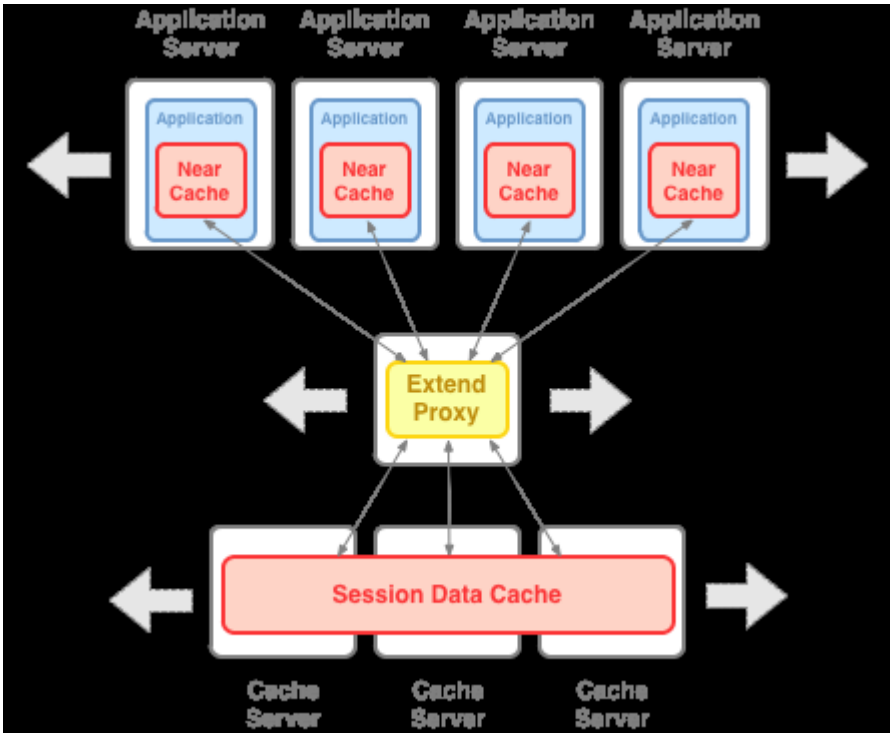


Out-of-Process with Coherence*Extend

The Out-of-Process with Coherence*Extend topology is similar to the Out-of-Process topology except that the communication between the application server tier and the cache server tier are over Coherence*Extend (TCP/IP).

This approach has the same benefits as the Out-of-Process topology and the ability to segment deployment of application servers and cache servers. This is ideal in an environment where application servers are on a network that does not support UDP. The cache servers can be set up in a separate dedicated network, with the application servers connecting to the cluster by using TCP.

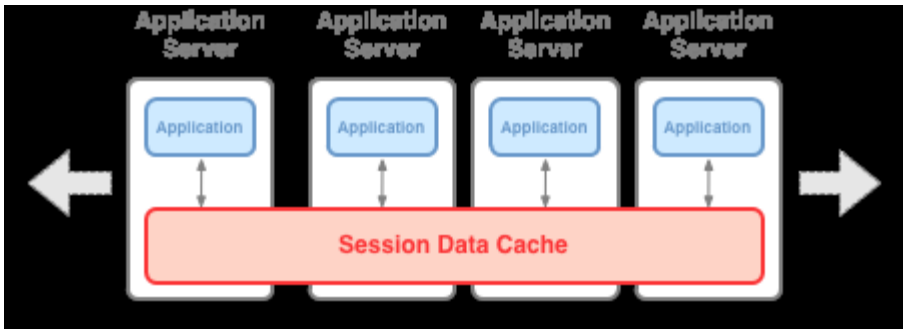
*Figure 4–9 Out-of-Process with Coherence*Extend Deployment Topology*



In-Process

The In-Process topology is not recommended for production use. This topology is supported mainly for development and testing. By storing the session data in-process with the application server, this topology is very easy to get up and running quickly for smoke tests, development and testing.

Figure 4–10 In-Process Deployment Topology



Managing and Monitoring Applications with JMX

> **Note:** To enable Coherence*Web JMX Management and Monitoring, this section assumes that you have first set up the Coherence Clustered JMX Framework. To set up this framework, see the configuration and installation instructions in *How to Manage Coherence with JMX* in the *Developer's Guide for Oracle Coherence*.

The management attributes and operations for Web applications that use Coherence*Web for HTTP session management are exposed through the `HttpSessionManagerMBean` interface (`com.tangosol.coherence.servlet.management.HttpSessionManagerMBean`).

During startup, each Coherence*Web Web application registers a single instance of `HttpSessionManagerMBean`. The MBean is unregistered when the Web application shuts down. Table 4–1 describes the MBean's object name used for registration.

Table 4–1 Object Name for the HttpSessionManagerMBean

Managed Bean	Object Name
HttpSessionManagerMBean	type=HttpSessionManager, nodeId=cluster node id, appId=web application id

Table 4–2 describes the information that is returned by the `HttpSessionManagerMBean`. All of the names represent attributes, with the exception of `resetStatistics`, which is an operation.

Several of the MBean attributes use the following prefixes:

- LocalSession—indicates a session that is not distributed to all members of the cluster. The session remains "local" to the originating server until a later point in the life of the session.
- LocalAttribute—indicates a session attribute that is not distributed to all members of the cluster.

- Overflow—typically, a larger and slower back-end cache that catches entries evicted from a faster front-end cache.

Table 4–2 Information Returned by the HttpSessionManagerMBean

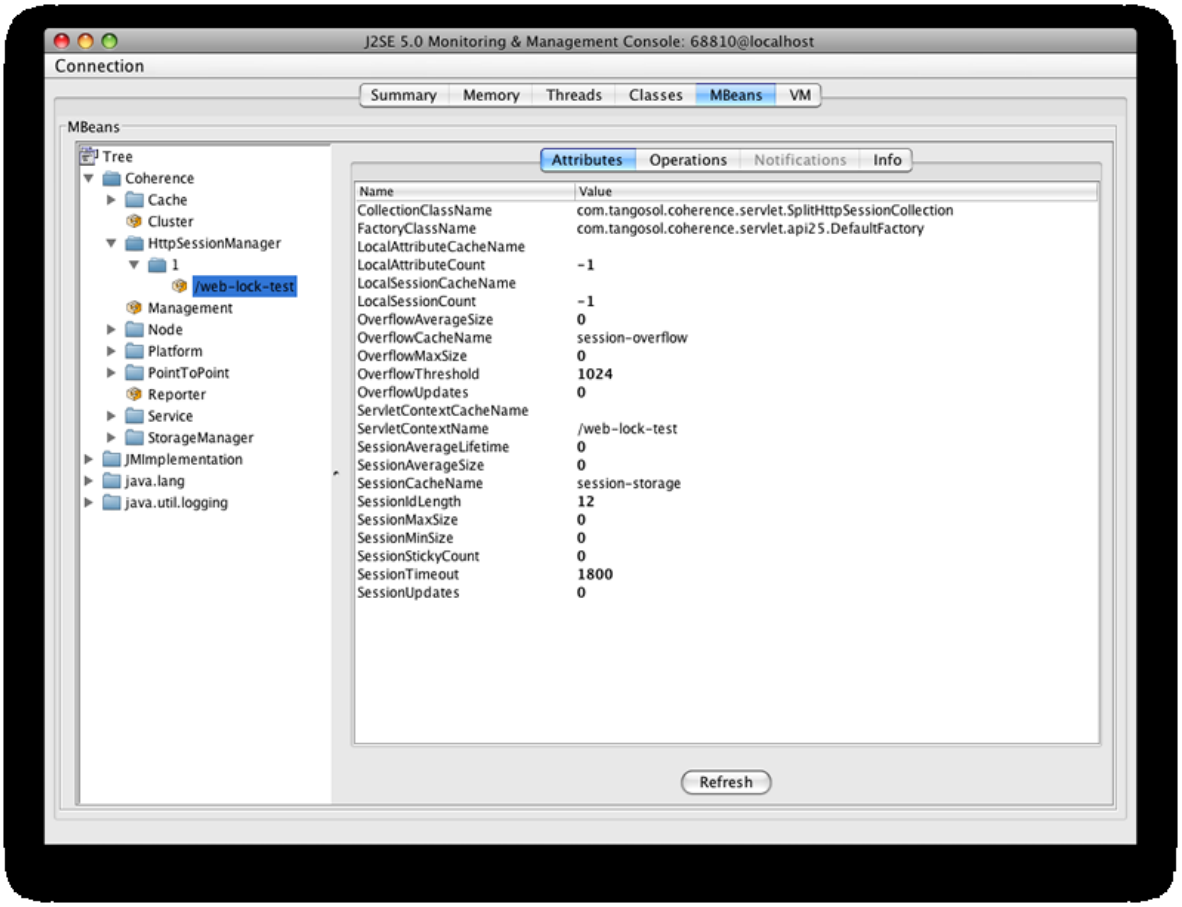
Name	Data Type	Description
CollectionClassName	String	The fully-qualified class name of the `HttpSessionCollection` implementation in use. The `HttpSessionCollection` interface is an abstract model for a collection of `HttpSessionModel` objects. The interface is not at all concerned with how the sessions are communicated between the clients and the servers.
FactoryClassName	String	The fully-qualified class name of the `Factory` implementation in use. The `SessionHelper.Factory` is used by the `SessionHelper` to obtain objects that implement various important parts of the Servlet specification. It can be placed in front of the application in place of the application server's own objects, thus changing the "apparent implementation" of the application server itself (for example, adding clustering.)
LocalAttributeCacheName	String	The name of the local cache that stores non-distributed session attributes. If the attribute displays null then local session attribute storage is disabled.
LocalAttributeCount	Integer	The number of non-distributed session attributes stored in the local session attribute cache. If the attribute displays `-1`, then local session attribute storage is disabled.
LocalSessionCacheName	String	The name of the local cache that stores non-distributed sessions. If the attribute displays `null`, then local session storage is disabled.
LocalSessionCount	Integer	The number of non-distributed sessions stored in the local session cache. If the attribute displays `-1`, then local session storage is disabled.
OverflowAverageSize	Integer	The average size (in bytes) of the session attributes stored in the "overflow" clustered cache since the last time statistics were reset. If the attribute displays `-1`, then a `SplitHttpSessionCollection` is not in use.
OverflowCacheName	String	The name of the clustered cache that stores the "large attributes" that exceed a certain size and thus are determined to be more efficiently managed as separate cache entries and not as part of the serialized session object itself. `Null` is displayed if a `SplitHttpSessionCollection` is not in use.
OverflowMaxSize	Integer	The maximum size (in bytes) of a session attribute stored in the "overflow" clustered cache since the last time statistics were reset. The attribute displays `-1` if a `SplitHttpSessionCollection` is not in use.
OverflowThreshold	Integer	The minimum length (in bytes) that the serialized form of an attribute value must be for that attribute value to be stored in the separate "overflow" cache that is reserved for large attributes. The attribute displays `-1` if a `SplitHttpSessionCollection` is not in use.
OverflowUpdates	Integer	The number of updates to session attributes stored in the "overflow" clustered cache since the last time statistics were reset. The attribute displays `-1` if a `SplitHttpSessionCollection` is not in use.
SessionAverageLifetime	Integer	The average lifetime (in seconds) of session objects invalidated (either due to expiration or to an explicit invalidation) since the last time statistics were reset.
SessionAverageSize	Integer	The average size (in bytes) of session objects placed in the session storage clustered cache since the last time statistics were reset.
SessionCacheName	String	The name of the clustered cache that stores serialized session objects.

Table 4–2 (Cont.) Information Returned by the HttpSessionManagerMBean

Name	Data Type	Description
SessionIdLength	Integer	The length (in characters) of generated session IDs.
SessionMaxSize	Integer	The maximum size (in bytes) of a session object placed in the session storage clustered cache since the last time statistics were reset.
SessionMinSize	Integer	The minimum size (in bytes) of a session object placed in the session storage clustered cache since the last time statistics were reset.
SessionStickyCount	Integer	The number of session objects that are pinned to this instance of the Web application. The attribute displays `-1` if sticky session optimizations are disabled.
SessionTimeout	Integer	The session expiration time (in seconds). The attribute displays `-1` if sessions never expire.
SessionUpdates	Integer	The number of updates of session object stored in the session storage clustered cache since the last time statistics were reset.
ServletContextCacheName	String	The name of the clustered cache that stores javax.servlet. `ServletContext` attributes. The attribute displays `null` if the `ServletContext` is not clustered.
ServletContextName	String	The name of the Web application `ServletContext`.
resetStatistics (operation)	void	Reset the session management statistics.

Figure 4–11 illustrates the `HttpSessionManagerMBean` as it is displayed in the JConsole browser.

Figure 4–11 HttpSessionManagerMBean Displayed in the JConsole Browser



5

Installing Coherence*Web on WebLogic Portal 10.3

Coherence*Web can be installed in a WebLogic Portal environment to provide session state management based on Coherence. Coherence*Web allows for more advanced deployment models, session models, and locking modes in a clustered environment. For more information on these features, see Chapter 4, "Coherence*Web Session Management Features."

Installing Coherence*Web with WebLogic Portal

> **Note:** As with other WebLogic Server applications, all of the information and instructions in Chapter 2, "Installing Coherence*Web on the WebLogic Server 10.3" apply to deploying Coherence*Web with WebLogic Portal. Please read Chapter 2 before proceeding. It provides much needed context for the following information.
>
> However, note that you do not have to install Coherence*Web on WebLogic Server to install it on WebLogic Portal.

Complete these steps to install Coherence*Web with WebLogic Portal applications:

1. Install WebLogic Patch ID N41D according to the instructions described in "Overview Of Configuration and Deployment" on page 2-2.
2. Copy the `coherence.jar` (included in Coherence 3.4.2) into the `APP-INF\lib` directory of the portal enterprise application.
3. (Optional) If you want to use the Coherence P13N `CacheProvider`, then copy the `coherence-wlp.jar` into the `APP-INF\lib` of the portal enterprise application.

   See "Using Coherence*Web and WebLogic Portal" on page 5-2 for more information on the P13N CacheProvider SPI implementation and WSRP-federated portals.
4. (Optional) If you want to use the Coherence P13N Cache as the **default** cache provider, add the following element before the first `<cache>` element to the `META-INF\p13n-cache-config.xml` file in the portal enterprise application:

   ```
   <default-provider-id>com.tangosol.coherence.weblogic</default-provider-id>
   ```
5. Reference `coherence-web-spi.war` by using a library-reference in the `WEB-INF\weblogic.xml` file in the portal Web application:

```
<wls:library-ref>
    <wls:library-name>coherence-web-spi</wls:library-name>
    <wls:specification-version>1.0.0.0</wls:specification-version>
    <wls:implementation-version>1.0.0.0</wls:implementation-version>
    <wls:exact-match>false</wls:exact-match>
</wls:library-ref>
```

6. To enable Coherence*Web sessions, set the application parameter `coherence-web-sessions-enabled` to `true` in the `WEB-INF\web.xml` file in the portal Web application:

```
<context-param>
    <param-name>coherence-web-sessions-enabled</param-name>
    <param-value>true</param-value>
</context-param>
```

7. Create a `.EAR` of the test application using workshop's EAR deployment. This EAR will be deployed to the cluster for testing.
8. In the portal domain, first deploy `coherence-web-spi.war` as a library to the cluster.
9. In the portal domain, deploy the application EAR to the cluster.

Using Coherence*Web and WebLogic Portal

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

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

P13N CacheProvider SPI Implementation

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

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

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

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

To install the Coherence P13N `CacheProvider`:

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

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

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

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

`http://download.oracle.com/docs/cd/E13155_01/wlp/docs103/caches/caches.html`

Sharing Data Between WSRP-Federated Portals Using Coherence

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

See the following document for complete details:

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

A

Coherence*Web Configuration Parameters

Coherence*Web provides a wide variety of configuration options as described in Table A–1. The process for configuring Coherence*Web is slightly different between the WebLogic SPI-based installation case and the generic WebInstaller installation case.

Table A–1 Configuration Parameters for Coherence*Web

Parameter Name	Description
coherence-factory-class	The fully-qualified class name of the `SessionHelper.Factory` to use. Defaults to `com.tangosol.coherence.servlet.api`*XX*`.DefaultFactory` where *XX* is `22`, `23`, `24`, or `25` for Servlet 2.2, 2.3, 2.4, or 2.5 containers respectively.
coherence-sessioncollection-class	The fully-qualified class name of the `HttpSessionCollection` implementation to use. Possible values include: ▪ `com.tangosol.coherence.servlet.MonolithicHttpSessionCollection` ▪ `com.tangosol.coherence.servlet.SplitHttpSessionCollection` ▪ `com.tangosol.coherence.servlet.TraditionalHttpSessionCollection` **A value for this parameter must be specified.**
coherence-cluster-owned	If `true`, Coherence*Web will automatically shut down the Coherence node when the Web application shuts down. You must use the WAR scoped cluster node deployment model in this case. See "WAR-Scoped Cluster Nodes" on page 4-10 for more information. If `false`, the Web application is responsible for shutting down the Coherence node (see `com.tangosol.net.CacheFactory.shutdown()`) in accordance to its own considerations. You must carefully consider a cluster node scoping deployment model in this case and the circumstances under which the application shuts down the Coherence node and the side-effects of doing so. See "Cluster Node Isolation" on page 4-8 for more information on cluster node scoping. **Note:** When using the WebInstaller, a value of `true` instructs the WebInstaller to place the Coherence library in the `WEB-INF/lib` directory of *each* Web application found in your J2EE application. If unspecified, this parameter defaults to `false`.
coherence-servletcontext-clustered (See Note 1)	Either `true` or `false` to indicate whether the attributes of the `ServletContext` will be clustered. If `true`, then all serializable `ServletContext` attribute values will be shared among all cluster nodes. If unspecified, defaults to `false`, primarily because the Servlet specification indicates that the `ServletContext` attributes are local to a JVM and should not be clustered.

*Table A–1 (Cont.) Configuration Parameters for Coherence*Web*

Parameter Name	Description
coherence-servletcontext-cachename (See Note 1)	The name of the Coherence cache that will be used to hold the servlet context data if the servlet context is clustered. If unspecified, defaults to `servletcontext-storage`. This parameter is described in Appendix B, "Session Cache Configuration File."
coherence-eventlisteners (See Note 1)	The comma-delimited list of names of application classes that want to receive events from the Web container. This list comes from the application listeners declared in the `listener` elements of `web.xml`.
coherence-enable-sessioncontext (See Note 1)	When set to `true`, allows the application to iterate sessions from the session context, thus disobeying the deprecation in the servlet specification. If unspecified, defaults to `false`.
coherence-contextless-session-retain-millis	The number of milliseconds that a server will hold a lock on a session while accessing that session without the session being implied by the current request context. A session is implied by the current request context if and only if the current thread is processing a Servlet request, and the request is associated with that session. All other access to a session object is "out of context". For example, if a reference to an arbitrary session is obtained from a `SessionContext` object (if that option is enabled), or if the application has code that holds on to session object references to manage sessions directly. Since session access requires session ownership, "out of context" access to the session object will automatically obtain ownership on behalf of the caller; that ownership will be retained for the number of milliseconds specified by this option so that repeated calls to the session do not individually obtain and release ownership, which is potentially an expensive operation. The legal range is 10 to 10000 (from `1/100`th of a second up to `10` seconds). If unspecified, defaults to `200`.
coherence-session-cookies-enabled (See Note 1)	If unspecified, defaults to `true` to enable session cookies.
coherence-session-cookie-name (See Note 1)	The name of the session cookie. If unspecified, defaults to `JSESSIONID`.
coherence-session-cookie-domain (See Note 1)	The domain of the session cookie as defined by RFC 2109. By default, no domain is set explicitly by the session management implementation.
coherence-session-cookie-path (See Note 1)	The path of the session cookie as defined by RFC 2109. By default, no path is set explicitly by the session management implementation.
coherence-session-cookie-max-age (See Note 1)	The maximum age in seconds of the session cookie as defined by RFC 2109. A value of -1 indicates that the cookie will not be persistent on the client; a positive value gives the maximum age that the cookie will be persisted by the client. Zero is not permitted. If unspecified, defaults to -1.
coherence-session-urlencode-enabled (See Note 1)	When set to true, enables URL encoding of session ids. If unspecified, defaults to `true`.
coherence-session-urlencode-name (See Note 1)	The parameter name to encode the session id into the URL with. On some containers, this value cannot be overridden. If unspecified, defaults to `jsessionid`.
coherence-session-urldecode-bycontainer (See Note 1)	When set to `true`, uses the container's decoding of the URL session ID. If `coherence-session-urlencode-name` has been overridden, this must be set to `false`. Setting this to `false` will not work in some containers. If unspecified, defaults to `true`.

*Table A–1 (Cont.) Configuration Parameters for Coherence*Web*

Parameter Name	Description
coherence-session-urlencode-bycontainer (See Note 1)	When set to `true` to use the container's encoding of the URL session ID. Setting this to `true` may conflict with the setting for `coherence-session-urlencode-name` if it has been specified. If unspecified, defaults to `false`.
coherence-reaperdaemon-cluster-coordinated	When set to `true`, coordinates reaping in the cluster such that only one server will perform reaping within a given reaping cycle, and it will be responsible for checking all of the sessions that are being managed in the cluster. If unspecified, defaults to `false`.
coherence-reaperdaemon-sweep-modulo	The number of times that the reaper reaps the sessions that are being used locally before it will check sessions that may be orphaned or expired elsewhere in the cluster. This setting is only used when coordinate reaping is disabled. Setting it to 1, for example, will cause the daemon to check this server's fair share of the session ids in the cluster every time that it wakes up to check for expired sessions (for example every 60 seconds). Setting it to 10 will cause those sessions to be checked every 600 seconds instead, but the sessions that are being used on this server will still be checked every 60 seconds. If unspecified, defaults to `4`.
coherence-reaperdaemon-assume-locality	This setting allows the reaper to assume that the sessions that are stored on this node (for example, by a distributed cache service) are the only sessions that this node must check for expiry. This value must be set to `false` if the session storage cache is being managed by nodes that are not running a reaper, for example if cache servers are being used to manage the session storage cache. (It is suggested that if cache servers are being used, that the "split" model be selected, and that the session overflow storage be run in a separate distributed cache service that is managed entirely by the cache servers, while the session storage cache itself remain in a distributed cache service that is managed entirely by the application server JVMs to be able to take advantage of this "assume locality" feature.) If unspecified, defaults to `true`.
coherence-reaperdaemon-cycle-seconds	The number of seconds that the daemon rests between reaping. For production clusters with long session timeouts, this can safely be set higher. For testing, particularly with short session timeouts, it can be set much lower. Setting it too low can cause more network traffic and use more processing cycles, and only has benefit if the application requires the sessions to be invalidated quickly when they have expired. If unspecified, defaults to `300`.
coherence-reaperdaemon-priority	The priority for the session reaper daemon. For more information, see the source for the `java.lang.Thread` class. If unspecified, defaults to `5`.
coherence-session-cachename	This name overrides the name of the clustered cache that stores the sessions. If unspecified, defaults to `session-storage`. This parameter is described in Appendix B, "Session Cache Configuration File."
coherence-session-deathcert-cachename	This name overrides the name of the clustered cache that stores the IDs of "recently departed" sessions. If unspecified, defaults to `session-death-certificates`. This parameter is described in Appendix B, "Session Cache Configuration File."
coherence-session-management-cachename	This name overrides the name of the clustered cache that stores the management and configuration information for the session management implementation. Generally, it should be configured as a replicated cache. If unspecified, defaults to `session-management`. This parameter is described in Appendix B, "Session Cache Configuration File."

*Table A–1 (Cont.) Configuration Parameters for Coherence*Web*

Parameter Name	Description
coherence-session-expire-seconds	This value overrides the session expiry time, and is expressed in seconds. Setting it to -1 will cause sessions to never expire. If unspecified, defaults to `1800`.
coherence-session-id-length (See Note 1)	This is the length, in characters, of generated session IDs. The suggested absolute minimum length is 8. If unspecified, defaults to `12`.
coherence-shutdown-delay-seconds	This value determines how long the session management implementation waits before shutting down after receiving the last indication that the application has been stopped, either from `ServletContextListener` events (Servlet 2.3 or later) or by the destruction of `Servlet` and `Filter` objects. This value is expressed in seconds. A value of zero indicates synchronous shut-down; any positive value indicates asynchronous shut-down. If unspecified, defaults to 0, because some servers are not capable of asynchronous shut-down.
coherence-session-member-locking	This value, if set to `true`, will prevent two threads in different JVMs from processing a request for the same session at the same time. A value of `false` is incompatible with sticky session optimizations and thread locking (that is, both `coherence-session-thread-locking` and `coherence-sticky-sessions` should be set to `false` if this value is set to `false`). If unspecified, defaults to `false`.
coherence-session-thread-locking	This value, if set to `true`, will prevent two threads in the same JVM from processing a request for the same session at the same time. If set to `true` the value of the `coherence-session-member-locking` parameter will be ignored, as thread locking implies member locking. If unspecified, defaults to `true`.
coherence-session-strict-spec	This value, if set to `true`, will indicate that the implementation will strictly adhere to the Servlet specification; setting it to `false` will allow the implementation to ignore certain types of exceptions, instead of shutting down the application. If unspecified, defaults to `true`.
coherence-sticky-sessions	This value, if set to `true`, specifies whether sticky sessions optimizations will be used. This should only be enabled if a sticky load balancer is being used. If unspecified, defaults to `false`.
coherence-distributioncontroller-class	This value specifies a class name of the `com.tangosol.coherence.servlet.HttpSessionCollection.SessionDistributionController` interface implementation to use. This feature requires `coherence-sticky-sessions` optimization to be enabled. Legal values include: ■ `com.tangosol.coherence.servlet.AbstractHttpSessionCollection.DistributedController` ■ `com.tangosol.coherence.servlet.AbstractHttpSessionCollection.HybridController` ■ `com.tangosol.coherence.servlet.AbstractHttpSessionCollection.LocalController`

*Table A–1 (Cont.) Configuration Parameters for Coherence*Web*

Parameter Name	Description
coherence-scopecontroller-class	This value specifies a class name of the optional `com.tangosol.coherence.servlet.HttpSessionCollection.AttributeScopeController` interface implementation to use. See "Session Attribute Scoping" on page 4-8 for more information. Legal values include: ▪ `com.tangosol.coherence.servlet.AbstractHttpSessionCollection.ApplicationScopeController` ▪ `com.tangosol.coherence.servlet.AbstractHttpSessionCollection.GlobalScopeController`
coherence-preserve-attributes	This value, if set to `true`, specifies whether non-serializable attributes should be preserved as local ones. This feature requires `coherence-sticky-sessions` optimization to be enabled. If unspecified, defaults to `false`.
coherence-local-session-cachename	This name overrides the name of the local cache that stores non-distributed sessions when `coherence-distributioncontroller-class` parameter is specified. If unspecified, defaults to `local-session-storage`. This parameter is described in Appendix B, "Session Cache Configuration File."
coherence-local-attribute-cachename	This name overrides the name of the local cache that stores non-distributed sessions when either `coherence-sessiondistributioncontroller-class` parameter is specified or `coherence-preserve-attributes` parameter is `true`. If unspecified, defaults to `local-attribute-storage`. This parameter is described in Appendix B, "Session Cache Configuration File."
coherence-session-overflow-cachename	For the split model, this value overrides the name of the clustered cache that stores the "large attributes" that exceed a certain size and thus are determined to be more efficiently managed as separate cache entries and not as part of the serialized session object itself. If unspecified, defaults to `session-overflow`. This parameter is described in Appendix B, "Session Cache Configuration File."
coherence-attribute-overflow-threshold	For the split model, this value specifies the minimum length (in bytes) that the serialized form of an attribute value must be for that attribute value to be stored in the separate "overflow" cache that is reserved for large attributes. If unspecified, defaults to `1024`.

Notes:

1. This parameter does not control Coherence Web behavior when used with the WebLogic SPI implementation. If you are configuring Coherence Web using the WebLogic SPI implementation, see "Configuring Web Applications for Coherence*Web" on page 2-6 for more details on how to configure this attribute.

B

Session Cache Configuration File

Coherence*Web uses the caches and services defined in the `session-cache-config.xml` file to implement HTTP session management. This file is deployed under `WEB-INF/classes` in either the instrumented Web application or shared WebLogic Coherence*Web SPI library. Table B–1 describes the default cache-related values used in the `session-cache-config.xml` file.

Table B–1 Cache-Related Values used in session-cache-config.xml

Value	Description
session-management	This cache is used to store internal configuration and management information for the session management implementation. This information is updated infrequently; therefore, it is a replicated cache by default.
servletcontext-storage	If `ServletContext` attribute clustering (see the `coherence-servletcontext-clustered` parameter in Table A–1) is enabled (it is disabled by default), this cache is used to store `ServletContext` attributes. This cache is replicated by default, as it is expected that there will a few read-mostly attributes.
session-storage	This cache is used to store session models. By default it is mapped to a near cache backed by a distributed cache since it is expected that a container will access and modify a subset of sessions multiple times (assuming that sticky session load balancing is configured.) See "Session Models" on page 4-1 for more information on session models.
session-overflow	If the `coherence-sessioncollection-class` parameter (described in Table A–1) is set to `com.tangosol.coherence.servlet.SplitHttpSessionCollection`, then this cache will hold "large" session attributes. By default, session attributes larger than 1K will be stored in this cache. This is configured as a distributed cache.
session-death-certificates	Recently expired session IDs are stored in this cache to prevent reuse of a recently used session ID. By default, each storage node will hold up to 4000 session IDs, and session IDs will be evicted after 24 hours. This is configured as a distributed cache.
local-session-storage	This local cache is used to store session models that are considered to be "local" by the configured (if any) `coherence-distributioncontroller-class` parameter. This parameter is described in Table A–1.
local-attribute-storage	This local cache is used to store attributes that are not distributed. This can happen under two conditions: ▪ A `coherence-distributioncontroller-class` is configured. Attributes for "local" sessions will be stored in this cache. ▪ A non-serializable attribute is set on a distributed session. If `coherence-sticky-sessions` and `coherence-preserve-attributes` are set to `true`, then this attribute will be placed in this cache. These parameters are described in Table A–1.

Table B–2 describes the services-related values used in the `session-cache-config.xml` file.

Table B–2 Services-Related Values used in session-cache-config.xml

Value	Description
ReplicatedSessionsMisc	This replicated service is used by the `session-management` and `servletcontext-storage` caches.
DistributedSessions	This distributed service is used by the following caches: ▪ `session-storage` ▪ `session-overflow` ▪ `session-death-certificates` The `tangosol.coherence.session.localstorage` system property controls whether a JVM will store and manage data for these caches. Under most circumstances, this should be set to `false` for Web container JVMs. See "Deployment Topologies" on page 4-13 for more details.
SessionOwnership	This invocation service is used by the sticky session optimization feature (if `coherence-sticky-sessions` is set to `true`).

Example B–1 illustrates the contents of the `session-cache-config.xml` file. The cache- and services-related values described in the previous tables appear in **bold**.

Example B–1 Contents of the session-cache-config.xml File

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!--                                                                     -->
<!--      Cache configuration descriptor for Coherence*Web               -->
<!--                                                                     -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<cache-config>
  <caching-scheme-mapping>
    <!--
    The clustered cache used to store Session management data.
    -->
    <cache-mapping>
      <cache-name>session-management</cache-name>
      <scheme-name>replicated</scheme-name>
    </cache-mapping>

    <!--
    The clustered cache used to store ServletContext attributes.
    -->
    <cache-mapping>
      <cache-name>servletcontext-storage</cache-name>
      <scheme-name>replicated</scheme-name>
    </cache-mapping>

    <!--
    The clustered cache used to store Session attributes.
    -->
    <cache-mapping>
      <cache-name>session-storage</cache-name>
      <scheme-name>session-near</scheme-name>
    </cache-mapping>
```

```
  <!--
  The clustered cache used to store the "overflowing" (split-out due to size)
  Session attributes. Only used for the "Split" model.
  -->
  <cache-mapping>
    <cache-name>session-overflow</cache-name>
    <scheme-name>session-distributed</scheme-name>
  </cache-mapping>

  <!--
  The clustered cache used to store IDs of "recently departed" Sessions.
  -->
  <cache-mapping>
    <cache-name>session-death-certificates</cache-name>
    <scheme-name>session-certificate</scheme-name>
  </cache-mapping>

  <!--
  The local cache used to store Sessions that are not yet distributed (if
  there is a distribution controller).
  -->
  <cache-mapping>
    <cache-name>local-session-storage</cache-name>
    <scheme-name>unlimited-local</scheme-name>
  </cache-mapping>

  <!--
  The local cache used to store Session attributes that are not distributed
  (if there is a distribution controller or attributes are allowed to become
  local when serialization fails).
  -->
  <cache-mapping>
    <cache-name>local-attribute-storage</cache-name>
    <scheme-name>unlimited-local</scheme-name>
  </cache-mapping>
</caching-scheme-mapping>

<caching-schemes>
  <!--
  Replicated caching scheme used by the Session management and ServletContext
  attribute caches.
  -->
  <replicated-scheme>
    <scheme-name>replicated</scheme-name>
    <service-name>ReplicatedSessionsMisc</service-name>
    <backing-map-scheme>
      <local-scheme>
        <scheme-ref>unlimited-local</scheme-ref>
      </local-scheme>
    </backing-map-scheme>
    <request-timeout>30s</request-timeout>
    <autostart>true</autostart>
  </replicated-scheme>

  <!--
  Near caching scheme used by the Session attribute cache. The front cache
  uses a Local caching scheme and the back cache uses a Distributed caching
  scheme.
  -->
  <near-scheme>
```

```
  <scheme-name>session-near</scheme-name>
  <front-scheme>
    <local-scheme>
      <scheme-ref>session-front</scheme-ref>
    </local-scheme>
  </front-scheme>
  <back-scheme>
    <distributed-scheme>
      <scheme-ref>session-distributed</scheme-ref>
    </distributed-scheme>
  </back-scheme>
  <invalidation-strategy>present</invalidation-strategy>
</near-scheme>

<local-scheme>
  <scheme-name>session-front</scheme-name>
  <eviction-policy>HYBRID</eviction-policy>
  <high-units>1000</high-units>
  <low-units>750</low-units>
</local-scheme>

<distributed-scheme>
  <scheme-name>session-distributed</scheme-name>
  <scheme-ref>session-base</scheme-ref>
  <backing-map-scheme>
    <local-scheme>
      <scheme-ref>unlimited-local</scheme-ref>
    </local-scheme>
    <!-- for disk overflow use this backing scheme instead:
    <overflow-scheme>
      <scheme-ref>session-paging</scheme-ref>
    </overflow-scheme>
    -->
  </backing-map-scheme>
</distributed-scheme>

<!--
Distributed caching scheme used by the "recently departed" Session cache.
-->
<distributed-scheme>
  <scheme-name>session-certificate</scheme-name>
  <scheme-ref>session-base</scheme-ref>
  <backing-map-scheme>
    <local-scheme>
      <eviction-policy>HYBRID</eviction-policy>
      <high-units>4000</high-units>
      <low-units>3000</low-units>
      <expiry-delay>86400</expiry-delay>
    </local-scheme>
  </backing-map-scheme>
</distributed-scheme>

<!--
"Base" Distributed caching scheme that defines common configuration.
-->
<distributed-scheme>
  <scheme-name>session-base</scheme-name>
  <service-name>DistributedSessions</service-name>
  <thread-count>0</thread-count>
  <lease-granularity>member</lease-granularity>
```

```
      <local-storage system-property="tangosol.coherence.session.
localstorage">false</local-storage>
      <partition-count>257</partition-count>
      <backup-count>1</backup-count>
      <backup-storage>
        <type>on-heap</type>
      </backup-storage>
      <backing-map-scheme>
        <local-scheme>
          <scheme-ref>unlimited-local</scheme-ref>
        </local-scheme>
      </backing-map-scheme>
      <request-timeout>30s</request-timeout>
      <autostart>true</autostart>
    </distributed-scheme>

    <!--
    Disk-based Session attribute overflow caching scheme.
    -->
    <overflow-scheme>
      <scheme-name>session-paging</scheme-name>
      <front-scheme>
        <local-scheme>
          <scheme-ref>session-front</scheme-ref>
        </local-scheme>
      </front-scheme>
      <back-scheme>
        <external-scheme>
          <bdb-store-manager/>
        </external-scheme>
      </back-scheme>
    </overflow-scheme>

    <!--
    Local caching scheme definition used by all caches that do not require an
    eviction policy.
    -->
    <local-scheme>
      <scheme-name>unlimited-local</scheme-name>
      <service-name>LocalSessionCache</service-name>
    </local-scheme>

    <!--
    Clustered invocation service that manages sticky session ownership.
    -->
    <invocation-scheme>
      <service-name>SessionOwnership</service-name>
      <request-timeout>30s</request-timeout>
    </invocation-scheme>
  </caching-schemes>
</cache-config>
```

Index

A

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

W