Oracle® Fusion Middleware

Administering HTTP Session Management with Oracle Coherence*Web

12*c* (12.1.3)

E47888-03

September 2016

Documentation for developers and administrators that describes how to configure, deploy, and use Coherence*Web, the HTTP session management module dedicated to managing session state in clustered environments, on a variiety of applications servers including WebLogic Server and GlassFish Server.

Oracle Fusion Middleware Administering HTTP Session Management with Oracle Coherence*Web, 12*c* (12.1.3)

E47888-03

Primary Author: Tom Pfaeffle

Contributing Author: James Kirsch, Pyounguk Cho, Arun Govindaraju

Contents

List of Examples

List of Figures

List of Tables

Preface

This guide describes how to deploy Oracle Coherence*Web (Coherence*Web), an HTTP session management module, to GlassFish Server, WebLogic Server and other application servers. It also describes the different session management features that you can configure.

This guide also describes how you can integrate Coherence*Web with WebLogic Portal, to provide session state management based on Oracle Coherence caches.

Audience

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

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

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

- *Oracle Fusion Middleware Developing Applications with Oracle Coherence*
- *Oracle Fusion Middleware Developing Remote Clients for Oracle Coherence*
- *Oracle Fusion Middleware Tutorial for Oracle Coherence*
- *Oracle Fusion Middleware Integrating Oracle Coherence*
- *Oracle Fusion Middleware Managing Oracle Coherence*
- *Oracle Fusion Middleware Administering Oracle Coherence*
- *Oracle Fusion Middleware Securing Oracle Coherence*
- *Oracle Fusion Middleware Java API Reference for Oracle Coherence*

- *Oracle Fusion Middleware .NET API Reference for Oracle Coherence*
- *Oracle Fusion Middleware C++ API Reference for Oracle Coherence*

Conventions

The following text conventions are used in this guide:

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.

What's New in This Guide

The following topics introduce the new and changed features of Oracle Coherence and other significant changes that are described in this guide, and provides pointers to additional information. This document is the new edition of the formerly titled *User's Guide for Oracle Coherence*Web*.

New and Changed Features for 12*c* (12.1.3)

Oracle Coherence 12*c* (12.1.3) includes the following new and changed features for this document.

- Coherence*Web for 12.1.3 is compatible with a number of different application servers. In the case of WebLogic Server, however, Weblogic Server 12.1.3 requires you to use Coherence*Web 12.1.3. Weblogic Server 12.1.3 is not compatible with earlier versions of Coherence*Web. For information on the compatibility between earlier versions of Coherence*Web and WebLogic Server, see "Using Coherence*Web with WebLogic Server" in *Administering HTTP Session Management with Oracle Coherence*Web (12.1.2)*.
- The `coherence-web-spi.war` file, which provided an SPI implementation of Coherence*Web, has been removed from the Coherence 12.1.3 release. The sections
- Support for ColdFusion has been removed from Coherence 12.1.3. It can still be used with earlier versions of Coherence. For example, for the Coherence 12.1.2 release, see "Working with ColdFusion Applications" in *Administering HTTP Session Management with Oracle Coherence*Web (12.1.2)*. For Coherence 3.7.1.*x*, see "Working with ColdFusion Applications" in Coherence User's Guide for Oracle Coherence*Web (3.7.1.x)
- Support for WebLogic Portal has been removed from Coherence 12.1.3. The use of WebLogic Portal is still supported by earlier versions of Coherence. For example, for the Coherence 12.1.2 release, see "Using Coherence*Web with WebLogic Portal" in *Administering HTTP Session Management with Oracle Coherence*Web (12.1.2)*. For Coherence 3.7.1.*x*, see "Using Coherence*Web with WebLogic Portal" in *Coherence User's Guide for Oracle Coherence*Web (3.7.1.x)*.

Other Significant Changes in this Document for 12*c* (12.1.3)

For 12*c* (12.1.3), this guide has been updated in several ways. Following are the sections that have been added or changed.

- Because Coherence 12.1.3 does not support ColdFusion, the chapter "Working with ColdFusion Applications" has been removed.

- Because Coherence 12.1.3 does not support Oracle WebLogic Portal, the chapter "Using Coherence*Web with WebLogic Portal" and the appendix "Cache Configuration for WebLogic Portal and Oracle Coherence" have been removed.
- Because `coherence-web-spi.war` file, which provided an SPI implementation of Coherence*Web, has been removed from the Coherence 12.1.3 release, the sections "Configuring for Non-Instrumented Applications that Use MyFaces and Coherence SPI" and "Configuring for Non-Instrumented Applications that use Mojarra and Coherence SPI" sections have bee removed from Chapter 8, "Working with JSF and MyFaces Applications."

1 Introduction to Coherence*Web

This chapter describes the advantages of using Coherence*Web for managing session state in clustered environments. It lists the containers that can use Coherence*Web and provides an installation and deployment roadmap. More detailed information on installation, deployment, and features are provided in following chapters.

This chapter contains the following sections:

Understanding 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), Coherence*Web:

- brings Coherence data grid's data scalability, availability, reliability, and performance to in-memory session management and storage.
- can configure fine-grained session and session attribute scoping by way of pluggable policies (see "Session and Session Attribute Scoping" on page 5-10).
- can be deployed to many mainstream application servers such as Oracle GlassFish Server, Oracle WebLogic Server, IBM WebSphere, Tomcat, and so on (see "Supported Web Containers" on page 1-1).
- 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 5-20).
- enables session sharing and management across different Web applications, domains and heterogeneous application servers (see "Session and Session Attribute Scoping" on page 5-10).
- can be used in advanced session models (that is, Monolithic, Traditional, and Split Session) that define how the session state is serialized or deserialized in the cluster (see "Session Models" on page 5-2).

Supported Web Containers

For WebLogic Server 12*c* (12.1.3), Coherence*Web is integrated with the product. No installation is necessary. For more information, see Chapter 2, "Using Coherence*Web with WebLogic Server."

For third-party application servers, Coherence*Web provides a generic installer, the *WebInstaller*, that transparently instruments your Web applications. Chapter 4, "Using Coherence*Web on Other Application Servers," describes how to use the WebInstaller to install Coherence*Web on these servers.

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. Notice that all of the Web containers (except Oracle WebLogic Server 10.3.4) share the same general installation instructions. A few, such as Caucho Resin, and WebLogic 10.*n*, require extra, container-specific steps that you must complete before starting the general installation instructions.

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 through the `-server` command-line option.

Table 1–1 Web Containers which can use Coherence*Web

Application Server	Server Type Alias	See this Installation Section
Oracle WebLogic 12.1.3	N/A	WebLogic Server 12c (12.1.3), Coherence*Web is integrated with the product. No installation is necessary. See Chapter 2, "Using Coherence*Web with WebLogic Server". **Note**: WebLogic Server and Coherence must be on the same versions when using Coherence*Web.
Apache Tomcat 5.5.*n*	`Tomcat/5.5.`*x*	"General Instructions for Installing Coherence*Web Session Management Module" on page 4-2 and "Enabling Sticky Sessions for Apache Tomcat Servers" on page 4-6
Apache Tomcat 6.0.*n*	`Tomcat/6.0.`*x*	"General Instructions for Installing Coherence*Web Session Management Module" on page 4-2 and "Enabling Sticky Sessions for Apache Tomcat Servers" on page 4-6
Apache Tomcat 7.*n*	`Tomcat/7.`*x*	"General Instructions for Installing Coherence*Web Session Management Module" on page 4-2 and "Enabling Sticky Sessions for Apache Tomcat Servers" on page 4-6
Caucho Resin 3.1.*n*	`Resin/3.1.`*x*	"Installing on Caucho Resin 3.1.n" on page 4-2
IBM WebSphere 5.*n*	`WebSphere/5.`*x*	"General Instructions for Installing Coherence*Web Session Management Module" on page 4-2
IBM WebSphere 6.*n*	`WebSphere/6.`*x*	"General Instructions for Installing Coherence*Web Session Management Module" on page 4-2
IBM WebSphere 7.*n*	`WebSphere/7.`*x*	"General Instructions for Installing Coherence*Web Session Management Module" on page 4-2 and "Decoding URL Session IDs for IBM WebSphere 7.n Servers" on page 4-6
IBM WebSphere 8.*n*	`WebSphere/8.`*x*	"General Instructions for Installing Coherence*Web Session Management Module" on page 4-2
IBM WebSphere 8.5	`WebSphere/8.`*x*	"General Instructions for Installing Coherence*Web Session Management Module" on page 4-2
JBoss Application Server	`Generic` or `Jetty/5.1.`*x*	"General Instructions for Installing Coherence*Web Session Management Module" on page 4-2
Jetty 5.1.*n*	`Jetty/5.1.`*x*	"General Instructions for Installing Coherence*Web Session Management Module" on page 4-2
Jetty 6.1.*n*	`Generic`	"General Instructions for Installing Coherence*Web Session Management Module" on page 4-2

Table 1–1 (Cont.) Web Containers which can use Coherence*Web

Application Server	Server Type Alias	See this Installation Section
Oracle GlassFish 3.*n*	N/A	Chapter 3, "Using Coherence*Web with GlassFish Server"
Oracle OC4J 10.1.2.*n*	`Oracle/10.1.2.x`	"General Instructions for Installing Coherence*Web Session Management Module" on page 4-2
Oracle OC4J 10.1.3.*n*	`Oracle/10.1.3.x`	"General Instructions for Installing Coherence*Web Session Management Module" on page 4-2
Sun Application Server 8.*n*	`Generic`	"General Instructions for Installing Coherence*Web Session Management Module" on page 4-2
Sun GlassFish 2.*n*	`Generic`	"General Instructions for Installing Coherence*Web Session Management Module" on page 4-2

Installation and Deployment Road Map

There are deployment decisions you should make before you configure and deploy Coherence*Web. Coherence*Web is supported on many different application servers. Regardless of which application server you are using, you might have to change some 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 scope of the Coherence nodes that are created within each application server JVM. Several different isolation modes are supported.

For example: you might be deploying multiple applications to the container that require the use of the same cluster (or one Coherence node); you might have multiple Web applications packaged in a single EAR file that use a single cluster; or you might have Web applications that must keep their session data separate and must be deployed to their own individual Coherence cluster. These choices and the deployment descriptors and elements that must be configured are described in "Cluster Node Isolation" on page 5-13.

Choose Your Locking Mode

Locking mode refers to the behavior of HTTP sessions when they are accessed concurrently by multiple Web container threads. Coherence*Web offers several different session locking options. For example, you can allow multiple nodes in a cluster to access an HTTP session simultaneously, or allow only one thread at a time to access an HTTP session. You can also allow multiple threads to access the same Web application instance while prohibiting concurrent access by threads in different Web application instances. These choices, and the deployment descriptors and elements that must be configured, are described in "Session Locking Modes" on page 5-17.

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 and restricts which session attributes are shared across the application boundaries. These choices, and the deployment descriptors and elements that must be configured, are described in "Session and Session Attribute Scoping" on page 5-10.

Choose When to Clean Up Expired HTTP Sessions

Coherence*Web provides a session reaper, which invalidates sessions that have expired. Chapter 7, "Cleaning Up Expired HTTP Sessions," describes the session reaper.

Choose the Installation Method

The installation procedure that you follow depends on your application server. "Supported Web Containers" on page 1-1 provides a list of the application servers and the corresponding instructions for installing Coherence*Web.

If you are using a recent release of WebLogic Server (such as 12*c* (12.1.3)) Coherence and Coherence*Web are installed with the WebLogic Server product. No separate Coherence*Web installation is necessary. See Chapter 2, "Using Coherence*Web with WebLogic Server."

For other application servers, use the generic Java EE WebInstaller described in Chapter 4, "Using Coherence*Web on Other Application Servers."

2

Using Coherence*Web with WebLogic Server

This chapter describes how to configure and deploy Coherence*Web, the session state persistence and management module, for use with WebLogic Server. The functionality that allows Coherence*Web to be used with this application server is contained in the Coherence*Web.

This chapter also provides an overview of Managed Coherence Servers and Grid Archive (GAR) format for packaging Coherence applications for the grid. A detailed discussion of Managed Coherence Servers and the GAR format is beyond the scope of this document. For more information, see *Oracle Fusion Middleware Developing Oracle Coherence Applications for Oracle WebLogic Server*.

This chapter contains the following sections:

Coherence*Web provides session state persistence and management. It is a session management module that uses Coherence caches for storing and managing session data. Coherence*Web is an alternative to the WebLogic Server in-memory HTTP state replication services. Consider using Coherence*Web if you are encountering any of these situations:

- Your application works with large HTTP session state objects
- You run into memory constraints, due to storing HTTP session object data
- You want to offload HTTP session storage to an existing Coherence cluster
- You want to share session state across enterprise applications and Web modules

Overview of Coherence*Web

The classes that define the Coherence*Web are contained in the `coherence-web.jar` file. To use the functionality provided by Coherence*Web, the `coherence.jar` classes must also be available to the Web application. Both of these files can be found in the `coherence\lib` directory.

In WebLogic Server 12*c* (12.1.3) and later, the `coherence-web.jar` and the `coherence.jar` files appear on the WebLogic Server system classpath. The `coherence-web.jar` will load application classes with the appropriate classloader in WebLogic Server. This means that you do not have to include the `coherence.jar` or `coherence-web.jar` files in the web application's classpath.

> **Note:** WebLogic Server and Coherence must be on the same versions when using Coherence*Web.

Coherence cache configurations and services used by Coherence*Web are defined in the `default-session-cache-config.xml` file, which can be found in the `coherence-web.jar` file. The default cache and services configuration defined in the `default-session-cache-config.xml` file should satisfy most Web applications.

You can create your own custom session cache configuration by packaging a file named `session-cache-config.xml` in your Web application. For more information see, "Using a Custom Session Cache Configuration File" on page 2-13.

When Coherence*Web is started on WebLogic Server, it first looks for a file named `session-cache-config.xml`. For example, the file can be placed in a WAR file's `WEB-INF/classes` directory, or packaged in a JAR file and placed in an EAR file's `APP-INF/lib` directory. If no custom session cache configuration XML resource is found, then it will use the `default-session-cache-config.xml` file packaged in `coherence-web.jar`.

In Coherence*Web, the following default cache configurations are defined:

- Coherence*Web for WebLogic Server is configured with local-storage disabled. The server will serve requests and will not be used to host data. 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.RequestTimeoutException` exception is thrown.

All Coherence*Web-enabled applications running on the 12*c* (12.1.3) release of WebLogic Server have application server-scope. In this configuration, all deployed applications become part of one Coherence node. See "Cluster Node Isolation" on page 5-13 for more information about cluster node scope.

Coherence*Web provides several session locking modes to control concurrent access of sessions. Both Coherence*Web employs Last Write Wins locking by default. See "Session Locking Modes" on page 5-17 for more information about locking modes.

By itself, Coherence*Web does not require a load balancer to run in front of the WebLogic Server tier. However, a load balancer will improve performance. It is required if the same session will be used concurrently and locking is not enabled. The default load balancer enforces HTTP session JVM affinity, however, other load balancing alternatives are available. WebLogic Server ships with several different proxy plug-ins which enforce JVM session stickiness. Documentation for configuring the WebLogic Server proxy plug-in is available at this URL:

`http://download.oracle.com/docs/cd/E17904_01/web.1111/e13709/load_balancing.htm`

Overview of Managed Coherence Servers

Oracle WebLogic Server and Coherence have defined Managed Coherence Servers which provide Coherence applications with the same benefits as other Java EE applications that are hosted on WebLogic Server, for example:

- Coherence applications can be deployed in a manner similar to other Java EE applications.
- Coherence applications in the grid can be managed by using WebLogic Server Console and WLST.
- Coherence clusters can be configured by using WebLogic configuration.
- Coherence Grid Archives can be integrated into Enterprise Archives (EAR files).
- Coherence applications can integrate with existing Coherence-based functionality.

Note: Using multiple Coherence clusters in a single WebLogic Server domain is not recommended.

For more information on Managed Coherence Servers, see "Creating Coherence Applications for WebLogic Server" in *Oracle Fusion Middleware Developing Oracle Coherence Applications for Oracle WebLogic Server*.

Configuring and Deploying Coherence*Web: Main Steps

The following steps summarize how to prepare your deployments to use Coherence*Web with applications running on WebLogic Server:

1. Install WebLogic Server and Oracle Coherence. See "Installing WebLogic Server and Oracle Coherence" on page 2-3.
2. (Optional) Modify the `web.xml` file in the deployment if your application requires advanced configuration for Coherence*Web. "Configure Coherence*Web" on page 2-4 describes the parameters that can be configured for Web applications. The entire set of Coherence*Web parameters are described in Appendix A, "Coherence*Web Context Parameters."
3. (Optional) Configure the WebLogic-generated HTTP session cookie parameters in the `weblogic.xml` or `weblogic-application.xml` file. See "Configure the Session Cookies" on page 2-6.
4. (Optional for testing; strongly suggested for production) Start a Cache Server Tier in a separate JVM from the one running WebLogic Server. See "Start a Cache Server" on page 2-9.
5. Set the Coherence*Web storage mode. See "Configure Coherence*Web Storage Mode" on page 2-11.
6. Deploy the application to WebLogic Server. See "Deploying Applications to WebLogic Server" on page 2-11.

Installing WebLogic Server and Oracle Coherence

WebLogic Server is installed by executing the `fmw_`*`version`*`_wls.jar`. This installer provides the full install and allows you to individually select the components to install (bits, examples, Javadoc). The installer supports both a graphical mode using the

Oracle Universal Installer (OUI) and a silent mode. Installing Coherence is an option in the WebLogic Server installer.

Use the `java` command to run the installer on the target computer, for example:

```
java -jar fmw_version_wls.jar
```

WebLogic Server is always installed to the *`ORACLE_HOME`*`/wlserver` directory; Coherence is always installed to the `ORACLE_HOME/coherence` directory.

For detailed instructions on installing WebLogic Server, see *Installing Oracle WebLogic Server*.

Configure Coherence*Web

Coherence*Web provides a default configuration that should satisfy most Web applications. Table 2–1 describes the context parameters configured by Coherence*Web. Table 2–2 describes the compatibility mode context parameter. For complete descriptions of all Coherence*Web parameters, see Appendix A, "Coherence*Web Context Parameters."

You can also configure the context parameters on the command line as system properties. The system properties have the same name as the context parameters, but the dash (`-`) is replaced with a period (`.`). For example, to declare a value for the context parameter `coherence-enable-sessioncontext` on the command line, enter it like this:

```
-Dcoherence.enable.sessioncontext=true
```

If both a system property and the equivalent context parameter are configured, the value from the system property is used.

Table 2–1 Context Parameters Configured by Coherence*Web

Parameter Name	Description
`coherence-application-name`	Coherence*Web uses the value of this parameter to determine the name of the application that uses the `ApplicationScopeController` interface to scope attributes. The value for this parameter should be provided in the following format: *application name* + ! + *Web module name* The *application name* is the name of the application that uses the `ApplicationScopeController` interface and *Web module name* is the name of the Web module in which it appears. For example, if you have an EAR file named `test.ear` and a Web-module named `app1` defined in the EAR file, then the default value for the `coherence-application-name` parameter would be `test!app1`. If this parameter is not configured, then Coherence*Web uses the name of the class loader instead. Also, if the parameter is not configured and the `ApplicationScopeController` interface *is* configured, then a warning is logged saying that the application name was not configured. See "Session Attribute Scoping" on page 5-12 for more information.
`coherence-reaperdaemon-assume-locality`	This setting allows the session 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 expiration. The default is `false`.
`coherence-scopecontroller-class`	This value specifies the class name of the optional `com.tangosol.coherence.servlet.HttpSessionCollection$AttributeScopeController` interface implementation. Valid values include: ■ `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$ApplicationScopeController` (default) ■ `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$GlobalScopeController` The default set by Coherence*Web is `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$ApplicationScopeController`.

Table 2–2 describes the `coherence-session-weblogic-compatibility-mode` context parameter which is specifically provided by Coherence*Web.

Table 2–2 Context Parameter Provided by the Coherence*Web

Parameter Name	Description
`coherence-session-weblogic-compatibility-mode`	This parameter is provided by Coherence*Web. If its value is set to `true`, it determines that a single session ID (with the cookie path set to "`/`") will map to a unique Coherence*Web session instance in each Web application. If it is `false`, then the standard behavior will apply: a single session ID will map to a single session instance using Coherence*Web in WebLogic Server. All other session persistence mechanisms in WebLogic use a single session ID in each Web application to refer to different session instances. This parameter defaults to `true` unless the global scope controller is specified. If this controller is specified, then the parameter defaults to `false`.

Table 2–3 describes the `coherence-factory-class` context parameter. The default value, which is set by Coherence*Web, should not be changed.

Table 2–3 Context Parameter Value that Should Not be Changed

Parameter Name	Description
`coherence-factory-class`	The fully qualified name of the class that implements the `SessionHelper.Factory` factory class. Coherence*Web sets the default value to `weblogic.servlet.internal.session.WebLogicSPIFactory`. This value should not be changed.

Configure the Session Cookies

If you are using Coherence*Web, then WebLogic Server generates and parses the session cookie. In this case, any native Coherence*Web session cookie configuration parameters will be ignored. To configure the session cookies, use the WebLogic-generated HTTP session cookie parameters in the `weblogic.xml` or `weblogic-application.xml` files. Table 2–4 describes these parameters.

In this table, **Updatable?** indicates whether the value of the parameter can be changed while the server is running. **Not applicable** indicates that there is no corresponding Coherence session cookie parameter.

Table 2–4 WebLogic-Generated HTTP Session Cookie Parameters

This Session Cookie Parameter...	Replaces this Coherence*Web Cookie Parameter	Description
`cookie-comment`	**Not applicable**	Specifies the comment that identifies the session tracking cookie in the cookie file. The default is `null`. **Updatable?** Yes
`cookie-domain`	`coherence-session-cookie-domain`	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. The default is `null`. **Updatable?** Yes
`cookie-max-age-secs`	`coherence-session-max-age`	Sets the life span of the session cookie, in seconds, after which it expires on the client. For more information about cookies, see "Using Sessions and Session Persistence" in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*. The default value is `-1` (unlimited). **Updatable?** Yes
`cookie-name`	`coherence-session-cookie-name`	Defines the session-tracking cookie name. Defaults to `JSESSIONID` if not set. You can set this to a more specific name for your application. The default is `JSESSIONID`. **Updatable?** Yes

Table 2–4 (Cont.) WebLogic-Generated HTTP Session Cookie Parameters

This Session Cookie Parameter...	Replaces this Coherence*Web Cookie Parameter	Description
`cookie-path`	`coherence-session-cookie-path`	Defines the session-tracking cookie path. If not set, this attribute defaults to a slash ("`/`") where the browser sends cookies to all URLs served by WebLogic Server. You can set the path to a narrower mapping, to limit the request URLs to which the browser sends cookies. The default is `null`. **Updatable?** Yes
`cookie-secure`	`coherence-session-cookie-secure`	Tells the browser that the cookie can be returned only over an HTTPS connection. This ensures that the cookie ID is secure and should be used only on Web sites that use HTTPS. Session cookies sent over HTTP will not work if this feature is enabled. Disable the `url-rewriting-enabled` element if you intend to use this feature. WebLogic Server generates the session cookie. The default is `false`. **Updatable?** Yes
`cookies-enabled`	`coherence-session-cookies-enabled`	Enables use of session cookies by default and is recommended, but you can disable them by setting this property to `false`. You might turn this option off for testing purposes. The default is `true`. **Updatable?** Yes
`debug-enabled`	**Not applicable**	Enables the debugging feature for HTTP sessions. Support it by enabling `HttpSessionDebug` logging and the WebLogic Server trace logger. The default value is `false`. **Updatable?** Yes
`encode-session-id-in-query-params`	**Not applicable**	Is set to `true` if the latest servlet specification requires containers to encode the session ID in path parameters. Certain Web servers do not work well with path parameters. In such cases, the `encode-session-id-in-query-params` element should be set to `true`. WebLogic Server generates the HTTP response. The default value is `false`. **Updatable?** Yes
`http-proxy-caching-of-cookies`	**Not applicable**	When set to `false`, WebLogic Server adds the following header and response to indicate that the proxy caches are not caching the cookies: "`Cache-control: no-cache=set-cookie`" WebLogic Server generates the HTTP response. The default value is `true.` **Updatable?** Yes

Table 2–4 (Cont.) WebLogic-Generated HTTP Session Cookie Parameters

This Session Cookie Parameter...	Replaces this Coherence*Web Cookie Parameter	Description
`id-length`	`coherence-session-id-length`	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 Wireless Application Protocol (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 rewriting. 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*. The default is 52. **Updatable?** No
`invalidation-interval-secs`	**Not applicable**	Sets the time, in seconds, that Coherence*Web waits between checks for timed-out and invalid sessions, and deleting the old sessions and freeing up memory. Use this element to tune WebLogic Server for best performance on high traffic sites. The default is `60`. **Updatable?** No
`timeout-secs`	**Not applicable**	Sets the time, in seconds, that Coherence*Web waits before timing out a session. On busy sites, you can tune your application by adjusting the timeout of sessions. While you want to give a browser client every opportunity to finish a session, you do not want to tie up the server needlessly if the user has left the site or otherwise abandoned the session. This element can be overridden by the `session-timeout` element (defined in minutes) in `web.xml`. The default is `3600` seconds. **Updatable?** No
`tracking-enabled`	**Not applicable**	Enables session tracking between HTTP requests. WebLogic Server generates the HTTP response. The default is `true`. **Updatable?** No
`url-rewriting-enabled`	`coherence-session-urlencode-enabled`	Enables URL rewriting, which encodes the session ID into the URL and provides session tracking if cookies are disabled in the browser and the `encodeURL` or `encodeRedirectedURL` methods are used when writing out URLs. For more information, see: `http://www.jguru.com/faq/view.jsp?EID=1045` WebLogic Server generates the HTTP response. The default is `true`. **Updatable?** Yes

Start a Cache Server

A Coherence cache server is responsible for storing and managing all cached data. Coherence is integrated within WebLogic Server as a container subsystem. The use of a container aligns the lifecycle of a Coherence cluster member with the lifecycle of a managed server: starting or stopping a managed server JVM starts and stops a Coherence cluster member. Managed servers that are cluster members are referred to as managed Coherence servers.

Coherence clusters are different than WebLogic Server clusters. They use different clustering protocols and are configured separately. Multiple WebLogic Server clusters can be associated with a Coherence cluster and a WebLogic Server domain can contain only a single Coherence cluster. Managed servers that are configured as Coherence cluster members are referred to as managed Coherence servers.

Managed Coherence servers can be explicitly associated with a Coherence cluster or they can be associated with a WebLogic Server cluster that is associated with a Coherence cluster. WebLogic Server-managed servers that are members of a Coherence cluster and are storage-enabled, act as cache servers.

For more information on Coherence clusters in a WebLogic server environment, see "Configuring and Managing Coherence Clusters" in *Administering Clusters for Oracle WebLogic Server*. For more information on Coherence applications in a WebLogic Server environment, see *Oracle Fusion Middleware Developing Oracle Coherence Applications for Oracle WebLogic Server*.

You can start a Coherence cache server or cluster either from the WebLogic Server Administration Console or from the command line, as described in the following sections.

- Starting a Coherence Cache Server from WebLogic Server Administration Console
- Starting a Coherence Cache Server from the Command Line

Starting a Coherence Cache Server from WebLogic Server Administration Console

Using the WebLogic Server Administration Console, you can enable storage for each WebLogic Server that is a member of a Coherence cluster. The Coherence session caches have a separate flag for enabling storage. For more information on this flag, see "Enabling the Coherence Session Cache in Weblogic Server Administration Console" on page 2-12.

> **Note:** If your managed server is a member of a Coherence cluster and is using Coherence*Web, then you can enable session storage by adding the `-Dtangosol.coherence.session.localstorage=true` system property to the startup command.

Coherence session caches automatically start with the WebLogic Server cluster. For an example of working with Coherence cache servers in the WebLogic Server Administration Console, see "Caching Sessions with Managed Coherence Servers" in *Oracle Fusion Middleware Tutorial for Oracle Coherence*.

The following steps summarize how to start a Coherence cluster in the WebLogic Server Administration Console.

1. Configure the Coherence Cluster.

   For detailed information, see "Configuring and Managing Coherence Clusters" in *Administering Clusters for Oracle WebLogic Server*.

2. Configure WebLogic Servers and clusters that will be associated with the Coherence cluster.

   For detailed information, see "Configuring and Managing Coherence Clusters" in *Administering Clusters for Oracle WebLogic Server*.

3. Enable Coherence*Web for the selected WebLogic Servers or clusters.

   For detailed information, see "Enabling the Coherence Session Cache in Weblogic Server Administration Console" on page 2-12.

Starting a Coherence Cache Server from the Command Line

Instead of using the WebLogic Server Administration Console, there may be situations when you might need to start a Coherence cache server or cluster from the command line. You can start the Coherence cache server from the command line either in standalone mode, or as part of a WebLogic Server instance.

- To Start a Standalone Coherence Cache Server
- To Start a Storage-Enabled or -Disabled WebLogic Server Instance

To Start a Standalone Coherence Cache Server Follow these steps to start a standalone Coherence cache server:

1. Create a script for starting a Coherence cache server. The following is a simple example of a script that creates and starts a storage-enabled cache server. This example assumes that you are using a Sun JVM. See "JVM Tuning" in *Oracle Fusion Middleware Developing Applications with Oracle Coherence* for more information.

   ```
   java -server -Xms512m -Xmx512m
   -cp <Coherence installation dir>/lib/coherence-web.jar:<Coherence installation
   dir>/lib/coherence.jar -Dtangosol.coherence.management.remote=true
   -Dtangosol.coherence.cacheconfig=session_cache_configuration_file
   -Dtangosol.coherence.session.localstorage=true
   -Dtangosol.coherence.cluster=Coherence_cluster_name
   com.tangosol.net.DefaultCacheServer
   ```

   You must include `coherence-web.jar` and `coherence.jar` on the classpath. The variable *`session_cache_configuration_file`* represents the absolute path to the cache configuration file on your file system. For Coherence*Web, the default session cache configuration file is named `default-session-cache-config.xml`. Note that the cache configuration defined for the cache server must match the cache configuration defined for the application servers which run on the same Coherence cluster.

   If your application uses additional Coherence caches, then you must merge the cache configuration information with a customized session cache configuration file. This customized session cache configuration file, typically named `session-cache-config.xml`, should contain the contents of `default-session-cache-config.xml` file and the additional caches used by your application.

   The cache and session configuration must be consistent across WebLogic Server and Coherence cache servers.

   For more information on merging these files, see "Merging Coherence Cache and Session Information" in *Oracle Fusion Middleware Integrating Oracle Coherence*.

   The variable *`Coherence_cluster_name`* represents the name of the Coherence cluster. A cluster name check has been added to 10.3.6 and later versions of WebLogic Server. The `tangosol.coherence.cluster` property must be added to the cache server because you are declaring the cluster name in the WebLogic Server application. If the Coherence servers are started in standalone mode, they must pass this property, otherwise the cluster will not form between the WLS servers and the standalone cache server.

2. Start one or more Coherence cache servers using the script described in the previous step.

To Start a Storage-Enabled or -Disabled WebLogic Server Instance By default, a Coherence*Web-enabled WebLogic Server instance starts in storage-disabled mode. To start the WebLogic Server instance in storage-enabled mode, follow these steps:

1. Create a script for starting a Coherence cache server. This can be similar to the script described in the previous section.
2. Include the command-line property to enable local storage, `-Dtangosol.coherence.session.localstorage=true`, in the server startup command. The WebLogic Server instance will start with Coherence*Web-enabled and local storage enabled.

   To start a Coherence*Web-enabled WebLogic Server instance, omit this system property. Local storage will be disabled by default.

For more information about working with WebLogic Server through the command line, see "weblogic.Server Command-Line Reference" in *Oracle Fusion Middleware Command Reference for Oracle WebLogic Server.*

Configure Coherence*Web Storage Mode

You can enable Coherence*Web session storage by specifying `coherence-web` as the value of the `persistent-store-type` attribute in the `weblogic.xml` session configuration. This configuration provides application server-level cluster node scoping for web applications deployed on WebLogic Server. No shared libraries need to be deployed or depended upon.

Coherence*Web is initialized only when a web application that requires session persistence is started in the WebLogic Server instance.

Example 2–1 illustrates a sample `weblogic.xml` file where `coherence-web` is the value of the `persistent-store-type` attribute.

Example 2–1 Coherence Web Storage Mode in weblogic.xml

```
<weblogic-web-app>
  ...
<session-descriptor>
      <persistent-store-type>coherence-web</persistent-store-type>
</session-descriptor>
  ...
</weblogic-web-app>
```

Deploying Applications to WebLogic Server

If you are using the default session cache configuration file with your web application, then you can package and deploy it like any other Java EE application. However, if you are using a custom session cache configuration file, then you must package and deploy the application in a GAR file.

GAR files deploy like any other Java EE application, except that you create a Coherence tier and nodes belonging to the tier. You can configure and deploy a standalone GAR or an embedded GAR.

For information about configuring and deploying standalone GAR files and embedded GAR files, see "Deploying Coherence Applications to WebLogic Server" in *Oracle Fusion Middleware Administering Oracle Coherence* and "Creating Coherence Applications for WebLogic Server" in *Developing Oracle Coherence Applications for Oracle WebLogic Server*.

Coherence MBean Attributes for Coherence*Web

WebLogic Server defines a cluster MBean (`weblogic.management.configuration.ClusterMBean`) which represents a cluster in the domain. The cluster MBean defines a number of attributes, operations, and MBeans related to the management of the cluster. Among the MBeans defined by the cluster MBean are the `CoherenceMemberConfigMBean` and the `CoherenceTierMBean` MBeans.

The `CoherenceMemberConfigMBean` and the `CoherenceTierMBean` MBeans each define an `isCoherenceWebLocalStorageEnabled` attribute that indicates whether a cluster or member is acting as a storage tier for Coherence*Web. This attribute is defined in Table 2–5.

Table 2–5 Coherence MBean Attribute for Coherence*Web

Attribute	Description
isCoherenceWebLocalStorageEnabled	If this attribute is set to `true` in `CoherenceTierMBean`, it indicates that a cluster is acting as a storage tier for Coherence*Web. Coherence*Web cache services will start with storage enabled when the server starts. When deploying a Coherence*Web-enabled application, there must be a running WebLogic cluster in the domain which has this attribute enabled. If this attribute is set to `true` in `CoherenceMemberConfigMBean`, it indicates that this node is acting as a storage node for Coherence*Web. Coherence*Web cache services will start with storage enabled when the server starts. When deploying a Coherence*Web-enabled application, there must be a running WebLogic cluster in the domain which has this attribute enabled. **Default:** `false`

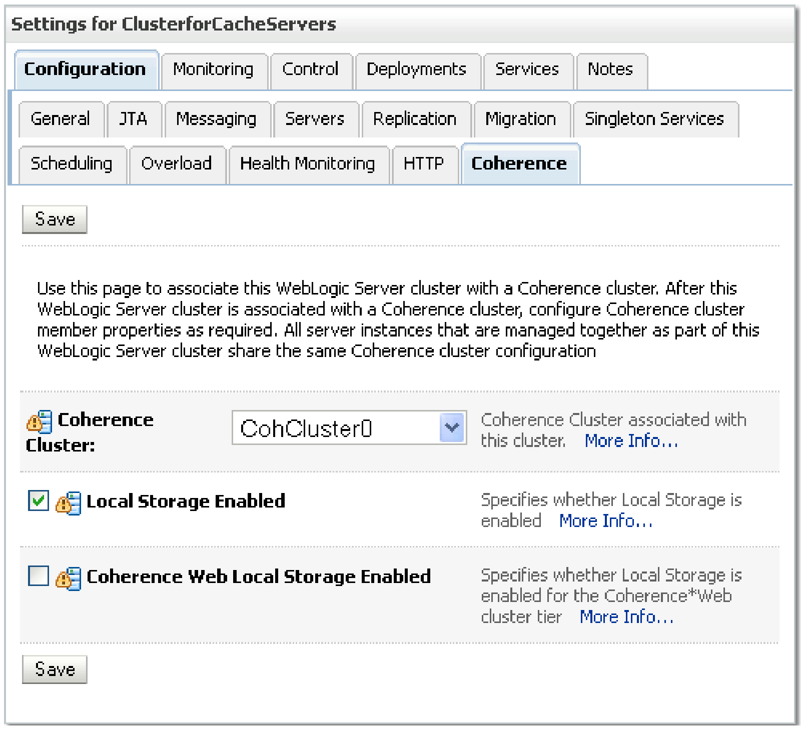
Enabling the Coherence Session Cache in Weblogic Server Administration Console

The **Coherence Web Local Storage Enabled** checkbox in the WebLogic Server Administration Console corresponds to the `isCoherenceWebLocalStorageEnabled` attribute described in the previous section. The checkbox indicates whether the cluster is acting as a storage tier for Coherence*Web. To navigate to this checkbox:

1. Select **Environment**, then **Clusters** in the Domain Structure Window.
2. Select a defined cluster in the **Clusters** table.
3. Select Configuration tab then the Coherence tab in the **Settings for *cluster*** page.

Figure 2–1 illustrates the **Coherence Web Local Storage Enabled** checkbox in the WebLogic Server Administration Console **Settings for *cluster*** page.

Figure 2–1 Coherence Web Local Storage Enabled Checkbox


Settings for ClusterforCacheServers

Use this page to associate this WebLogic Server cluster with a Coherence cluster. After this WebLogic Server cluster is associated with a Coherence cluster, configure Coherence cluster member properties as required. All server instances that are managed together as part of this WebLogic Server cluster share the same Coherence cluster configuration

Coherence Cluster:	CohCluster0	Coherence Cluster associated with this cluster. More Info...
✓ Local Storage Enabled		Specifies whether Local Storage is enabled More Info...
☐ Coherence Web Local Storage Enabled		Specifies whether Local Storage is enabled for the Coherence*Web cluster tier More Info...

Using a Custom Session Cache Configuration File

The `coherence-web.jar` file contains a `default-session-cache-config.xml` cache configuration file which should be sufficient for most applications. However, if you are working with technologies such as Coherence*Extend or Push Replication, or if you have WebLogic Server nodes that are to act as storage-enabled cache servers with a custom session cache configuration, then you must provide a custom session cache configuration file. Custom session cache configuration files must be packaged in a GAR file for deployment.

To use a custom session cache configuration file on WebLogic Server and package it in a GAR file, follow these steps for web applications and for the WebLogic Server nodes acting as cache servers:

For web applications using Coherence*Web:

1. If you are using a custom session cache configuration file (which should be named `session-cache-config.xml`), then package it in your web application:

   - For a WAR file, place the session cache configuration file in the `WEB-INF/classes` folder
   - For an EAR file, package the session cache configuration file in a JAR file and place it in the shared library (the `APP-INF/lib` folder) in an EAR file

   Note that you can customize the session cache configuration file name, but then you must provide the new file name as the value of the `coherence-cache-configuration-path` context parameter in the `web.xml` file.

2. If you do not want the WebLogic Server cluster members running the Coherence*Web application to act as a cache server, then ensure that the **Coherence Web Local Storage Enabled** checkbox in the WebLogic Server

Administration Console "Settings for *cluster*" page is **not** selected for the cluster members. This will cause the custom session cache configuration file to be read.

For WebLogic Server nodes acting as cache servers:

1. If you are using a custom session cache configuration file, then construct a GAR file containing the file and a `coherence-application.xml` file. The GAR file has the following structure:

   ```
   my.gar
   session-cache-config.xml
   META-INF
     coherence-application.xml
     MANIFEST.MF
   ```

   For more information on the packaging requirements for a GAR file, see "Packaging Coherence Applications for WebLogic Server" in *Oracle Fusion Middleware Administering Oracle Coherence* and "Creating Coherence Applications for Oracle WebLogic Server" in *Developing Oracle Coherence Applications for Oracle WebLogic Server*.

   a. Create the custom session cache configuration file and name it `session-cache-config.xml`.

      If you are deploying a GAR file, set the `local-storage` parameter in the custom `session-cache-config.xml` file to `true`, to configure all caches to start with storage enabled, for example:

      ```
      <local-storage>true</local-storage>
      ```

      Note: The `local-storage` parameter specifies whether a cluster node contributes storage to the cluster. In WebLogic Server, the `local-storage` parameter does not enable storage in Coherence*Web for WebLogic Server members that have a GAR file deployed to them.

   b. Create a `coherence-application.xml` file. In the file, use the `cache-configuration-ref` parameter to reference your custom `session-cache-config.xml` file, for example:

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

2. Deploy the GAR file to the WebLogic Server cluster that is to act as the storage-enabled Coherence cluster members.

   Note that storage must be enabled in either of the following ways:

   - Enable storage in the `session-cache-config.xml` file (see Step 1a).
   - Enable storage in the server itself either by selecting the **Coherence Web Local Storage Enabled** checkbox in the WebLogic Server Administration Console or by setting the JVM argument `tangosol.coherence.session.localstorage` to `true`.

   For information on deploying GAR files, see "Deploying Coherence Applications To a WebLogic Server Domain" in *Oracle Fusion Middleware Administering Oracle Coherence*. and "Deploying Coherence Applications in WebLogic Server" in *Developing Oracle Coherence Applications for Oracle WebLogic Server*.

Scoping the Session Cookie Path

WebLogic Server and Coherence*Web handle session scoping and the session lifecycle in different ways. This can impact your decision to implement a single sign-on (SSO) strategy for your applications.

By default, WebLogic Server uses the same session ID in every Web application for a given client, and sets the session cookie path to a forward slash (/). This is a requirement of the WebLogic Server default *thin* SSO implementation, which is enabled by default. By generating a session cookie with a path of "/", clients always return the same session ID in every request to the server. In WebLogic Server, a single session ID can be mapped to multiple session objects. Each Web application will have a different session object instance even though the session ID is identical (unless session sharing is enabled).

In contrast, Coherence*Web maps a session ID to a single session instance. This means that the behavior of having multiple session instances mapped to the same ID is not replicated by default if an application uses Coherence*Web. Because the session cookie is mapped to "/" by default, a single Coherence*Web session is shared across all Web applications. The default configuration in Coherence*Web is that all session attributes are scoped to a Web application. For most purposes, this single session approach is transparent. The major difference of having a single session across all Web applications is the impact of session invalidation. If Coherence*Web is enabled and you invalidate a session in one Web application, then you invalidate that session in all Web applications that use that session instance. If your Web applications do not use *thin* SSO, then you can avoid this issue by scoping the session cookie to the Web application path.

Therefore, you have the following options regarding SSO:

- Enable "WebLogic Server session compatibly mode". This configuration is set with the `coherence-session-weblogic-compatibility-mode` parameter and mirrors all of the native WebLogic Server session persistence types: memory (single-server, non-replicated), file system persistence, JDBC persistence, cookie-based session persistence, and in-memory replication (across a cluster). By default, this mode is enabled. See "Using Sessions and Session Persistence" in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server* for more information.
- Enable *thin* SSO functionality. Clients will use a single session across all Web applications. This means that the session life cycle will be inconsistent with all other session persistence types.
- Disable the *thin* SSO functionality by scoping the session cookie path to the Web application context path. This will allow the session life cycle to be consistent with all other session persistence types.

One advantage of enabling thin SSO with Coherence*Web is that it will work across all Web applications that are using the same Coherence cluster for Coherence*Web. The Coherence cluster is completely independent from the WebLogic Server cluster. The thin SSO functionality can even span multiple domains by enabling cross-domain trust in the WebLogic Server security layer.

Updating the Session ID

When a user successfully authenticates a protected resource, the session ID is changed for security purposes.

In previous releases of WebLogic Server, a new session would be created, all of the session attributes from the old session would be copied into the new session, and then the old session would be invalidated. This would trigger the session listeners (if any were registered), so session lifecycle and session attribute listeners would be executed.

The current release of WebLogic Server implements the `HttpServletRequest.changeSessionId` method from the Java Servlet 3.1 Specification. The implementation of the `changeSessionId` method allows the actual session ID to be updated. This means that no session lifecycle events will be triggered and no listeners will be executed. Most users should not notice any changes in the behavior of their applications.

For more information on the `HttpServletRequest.changeSessionId` method, see the Java Servlet 3.1 Specification and Javadoc available from this URL:

`http://jcp.org/en/jsr/detail?id=340`

Sharing Coherence*Web Sessions with Other Application Servers

If you are running Coherence*Web on WebLogic Server and on other application servers within a single cluster, then the session cookies created by WebLogic Server will not be decoded correctly by Coherence*Web on the other servers. This is because WebLogic Server adds a session affinity suffix to the cookie which is not part of the session ID stored in Coherence*Web. The other application servers must remove the WebLogic session affinity suffix from the session cookie value for Coherence*Web to be able to retrieve the session from the Coherence cache.

To strip the WebLogic session affinity suffix from the session cookie, add the `coherence-session-affinity-token` context parameter to the `web.xml` file used in the other application servers. Set the parameter value to an exclamation point (`!`), as illustrated in Example 2–2. The session affinity suffix will be removed from the session cookie when it is processed by the other application server.

Example 2–2 Removing Session Affinity Suffix

```
...
<context-param>
  <param-name>coherence-session-affinity-token</param-name>
  <param-value>!</param-value>
</context-param>
...
```

See Appendix A, "Coherence*Web Context Parameters" for more information on the `coherence-session-affinity-token` context parameter.

WebLogic Server and Coherence: Compatibility

Coherence*Web for 12.1.3 is compatible with a number of different application servers. In the case of WebLogic Server, WebLogic Server and Coherence must be on the same versions when using Coherence*Web. Weblogic Server 12.1.3 is not compatible with earlier versions of Coherence*Web. For information on the compatibility between earlier versions of Coherence*Web and WebLogic Server, see "Using Coherence*Web with WebLogic Server" in *Administering HTTP Session Management with Oracle Coherence*Web (12.1.2).*

3 Using Coherence*Web with GlassFish Server

This chapter describes how to configure and deploy Coherence*Web, the session state persistence and management module, for use with GlassFish Server.

This chapter contains the following sections:

- Overview of GlassFish Server
- Overview of Coherence*Web on GlassFish
- Configuring And Deploying Coherence*Web on GlassFish Server—Main Steps

Overview of GlassFish Server

Oracle GlassFish Server delivers a flexible, lightweight and extensible Java EE 6 platform. It provides a small footprint, fully featured Java EE application server that is completely supported for commercial deployment and is available as a standalone offering. Oracle GlassFish Server is best suited for applications requiring lightweight infrastructure and the most recent implementations of Java Enterprise Edition. This chapter describes how to set up and deploy Coherence*Web so that it can be used by applications running on GlassFish Server.

Overview of Coherence*Web on GlassFish

Coherence*Web provides functionality in Web applications deployed on Oracle GlassFish Servers. In previous releases, the Coherence*Web WebInstaller was required to pre-process applications before they could use Coherence*Web for session storage. With the current release, the WebInstaller pre-processing step is not required for GlassFish 3.1 applications.

Functionality for GlassFish is provided in the `coherence-web.jar` file which can be found in the `coherence/lib` folder. Coherence cache configurations and services for GlassFish are defined in the `default-session-cache-config.xml` file, which can be found in the `coherence-web.jar` file. The default cache and services configuration defined in the `default-session-cache-config.xml` file should satisfy most Web applications.

Since Coherence*Web uses Coherence caches to store session data, the `coherence.jar` file must also be available to the Web application's classloader. See "Configure Cluster Nodes" on page 3-7 for more information on configuring Coherence*Web to run on EAR-scoped or WAR-scoped cluster nodes.

By default, the Coherence node running on GlassFish Server is configured as storage-disabled. A separate Coherence cache server must be running for the Web application to work.

Coherence*Web provides several session locking modes to control concurrent access of sessions. On GlassFish Servers, Coherence*Web employs Last Write Wins Locking by default. This allows concurrent access to a session by multiple threads in a single JVM or multiple JVMs while prohibiting concurrent modification. See "Session Locking Modes" on page 5-17 for more information about locking modes.

The split session model (`SplitHttpSessionModel`), where small session attributes are stored as a single cache entry, and large attributes are stored as individual cache entries, is the default session model used. See "Session Models" on page 5-2 for more information.

Heap space is made available in GlassFish Server by storing session data in the Coherence data grid. Storing session data outside of GlassFish Server allows the Web application and the server to be restarted without any loss of session data. It also allows sessions to be shared across different web applications. Coherence*Web can run on GlassFish Server in either EAR-scoped or WAR-scoped cluster nodes. Application Server-scope is not supported.

Configuring And Deploying Coherence*Web on GlassFish Server—Main Steps

The following steps summarize how to prepare your deployments to use Coherence*Web with applications running on GlassFish Server:

1. Download Oracle Coherence to your file system. See "Download Oracle Coherence".
2. Set the session persistence type to `coherence-web` for the Web application in `glassfish-web.xml` file. See "Set the Session Persistence Type" on page 3-3.
3. (Optional) If you must override the default Coherence*Web cache or cluster configuration, edit the `default-session-cache-config.xml` file. See "Override the Default Coherence*Web Cache or Cluster Configuration" on page 3-3.
4. Make the `coherence-web.jar` file available to the Web application. See "Copy the Coherence*Web and Session Cache Files to the Application" on page 3-3.
5. Edit the `web.xml` file to make your Web application available to a server cluster. See "Make Your Web Applications Distributable" on page 3-3.
6. (Optional) Edit the `web.xml` file in the WAR deployment if your application requires advanced configuration for Coherence*Web. See "Configure Coherence*Web".
7. (Optional for testing; strongly suggested for production) Start a Cache Server Tier in a JVM that is separate from the one running GlassFish Server. See "Start a Cache Server" on page 3-6.
8. Package the application. Coherence*Web can be configured for EAR-scoped or WAR-scoped cluster nodes on GlassFish Server. See "Configure Cluster Nodes" on page 3-7.

Download Oracle Coherence

All of the files supporting Coherence*Web are included in the Coherence distribution. You can get the latest release of Oracle Coherence at this URL:

`http://www.oracle.com/technetwork/middleware/coherence/overview/index.html`

Set the Session Persistence Type

Create or edit the `glassfish-web.xml` file in the `WEB-INF` directory of the Web application. Set the session persistence type for the Web application to be `coherence-web`, for example:

Example 3–1 Configuring the glassfish-web.xml File for Coherence*Web

```
<glassfish-web-app>
   <session-config>
       <session-manager persistence-type="coherence-web" />
   </session-config>
</glassfish-web-app>
```

Override the Default Coherence*Web Cache or Cluster Configuration

The `default-session-cache-config.xml` file provides the default definition of the session caches and services which Coherence*Web uses to implement HTTP session management. The `default-session-cache-config.xml` file contained in the `coherence-web.jar` file should satisfy most Web applications. If necessary, you can provide an alternate cache and cluster configuration in your own custom session cache configuration file and `tangosol-coherence-override.xml` file. You must include the file in the `WEB-INF/classes` directory in the Web application.

For information on specifying the name of the custom session cache configuration file, see "Customizing the Name of the Session Cache Configuration File" on page 5-27.

Copy the Coherence*Web and Session Cache Files to the Application

The `coherence-web.jar` file provides the functionality for Coherence*Web on GlassFish. You must make the `coherence-web.jar` file available to the applications that you want to run on GlassFish Server.

Copy the `coherence-web.jar` file to the `/WEB-INF/lib/` directory of each Web application that you intend to deploy on GlassFish Server.

Make Your Web Applications Distributable

Your Web application must be configured to run in a distributed environment, such as a server cluster. Add the `<distributable/>` element to the `web.xml` deployment descriptor of your Web application. The `<distributable/>` element is a child of the root `<web-app>` element. The `web.xml` file is located in the `WEB-INF` directory of your Web application.

Configure Coherence*Web

Coherence*Web provides a default configuration that should satisfy most Web applications. The Coherence*Web configuration is defined using `<context-param>` elements in the `web.xml` file. The default values can be overridden by adding or editing `<context-params>` elements in the `web.xml` file.

Note: If you make any changes to the Coherence*Web configuration after a Web application has been started, then you must restart the Web application. There are no dynamically updatable configuration options in Coherence*Web.

Table 3–1 describes the default values provided for Coherence*Web on GlassFish Servers. See Appendix A, "Coherence*Web Context Parameters" for more information about these parameters.

Table 3–1 Default Context Parameter Values Provided by Coherence*Web

Parameter Name	Description and Default Value
`coherence-application-name`	The default is *ServletContext path + ServletContext name* A consistent and unique string to represent the Web application name.
`coherence-cluster-owned`	The default is `false` Because all Coherence nodes start when the Web application starts, they should also shutdown the Coherence node when the Web application stops.
`coherence-reaperdaemon-assume-locality`	The default is `false` Sessions can be stored on standalone cache servers.
`coherence-scopecontroller-class`	The default is `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$ApplicationScopeController`
`coherence-session-locking`	The default is `false` Session locking is disabled by default. This configuration enables a "last writer wins" policy.
`coherence-session-locking-mode`	The default is `none`. Session locking is disabled by default. This configuration enables a "last writer wins" policy.
`coherence-sessioncollection-class`	The default is `com.tangosol.coherence.servlet.SplitHttpSessionCollection` `SplitHttpSessionCollection` is the recommended default model.

Coherence*Web context parameters that configure the session cookie are not honored because GlassFish Server generates and parses the session cookie. Even though Coherence*Web can be configured to enable servlet contexts to be clustered, GlassFish does not support this functionality.

Table 3–2 describes the Coherence*Web context parameters that are not valid when used with GlassFish. GlassFish will return a warning log if these context parameters are present. See Appendix A, "Coherence*Web Context Parameters" for more information about these parameters.

Table 3–2 Coherence*Web Context Parameters that are not Valid for the GlassFish Server

Parameter Name	Reason Why it is Not Valid
`coherence-distributioncontroller-class`	The value for GlassFish Server is `com.tangosol.coherence.servlet.glassfish31.GlassFishHybridController`.
`coherence-preserve-attributes`	GlassFish Server caches nonserializable user data in the session.
`coherence-session-cookie-domain`	This parameter is not valid because GlassFish Server is responsible for generating and parsing the session cookie.
`coherence-session-cookie-max-age`	This parameter is not valid because GlassFish Server is responsible for generating and parsing the session cookie.
`coherence-session-cookie-name`	This parameter is not valid because GlassFish Server is responsible for generating and parsing the session cookie.

Table 3–2 (Cont.) Coherence*Web Context Parameters that are not Valid for the GlassFish Server

Parameter Name	Reason Why it is Not Valid
`coherence-session-cookie-path`	This parameter is not valid because GlassFish Server is responsible for generating and parsing the session cookie.
`coherence-session-cookies-enabled`	This parameter is not valid because GlassFish Server is responsible for generating and parsing the session cookie, as well as encoding URLs with session id.
`coherence-eventlisteners`	This parameter is not valid because Coherence*Web on GlassFish automatically registers session event listeners registered in `web.xml`.
`coherence-servletcontext-clustered`	This parameter is not valid because Coherence*Web does not support a clustered ServletContext in Coherence*Web for GlassFish.
`coherence-session-id-length`	This parameter is not valid because GlassFish Server is responsible for generating and parsing the session cookie.
`coherence-session-urldecode-bycontainer`	This parameter is not valid because GlassFish Server is responsible for generating and parsing the session cookie, as well as encoding and decoding URLs.
`coherence-session-urlencode-enabled`	This parameter is not valid because GlassFish Server is responsible for generating and parsing the session cookie, as well as encoding URLs with session id.
`coherence-session-urlencode-name`	This parameter is not valid because GlassFish Server is responsible for generating and parsing the session cookie, as well as encoding URLs with session id.

Table 3–3 describes the valid session configuration parameters in the `glassfish-web.xml` file. They are valid because GlassFish creates the session cookie and performs URL encoding in servlets.

Table 3–3 Valid GlassFish Session Configuration Parameters in glassfish-web.xml

Parameter Name	Reason Why it is Valid
`session-config/cookie-properties/cookieComment`	GlassFish generates the session cookie.
`session-config/cookie-properties/cookieMaxAgeSeconds`	GlassFish generates the session cookie.
`session-config/cookie-properties/cookiePath`	GlassFish generates the session cookie.
`session-config/cookie-properties/cookieSecure`	GlassFish generates the session cookie.
`session-config/cookie-properties/cookieDomain`	GlassFish generates the session cookie.
`session-config/session-properties/enableCookies`	GlassFish Server generates the session cookie.
`session-config/session-properties/enableURLRewriting`	GlassFish Server generates the session cookie.

Table 3–3 (Cont.) Valid GlassFish Session Configuration Parameters in glassfish-web.xml

Parameter Name	Reason Why it is Valid
`session-config/session-manager/persistence-type`	This parameter must be set to `coherence-web` to enable Coherence*Web.
`session-config/session-manager/manager-properties/reap-interval-in-seconds`	This value sets the `coherence-reaperdaemon-cycle-seconds` Coherence*Web configuration parameter. The default for the GlassFish Server is 60 seconds.
`session-config/session-properties/timeoutSeconds`	This value sets the `coherence-session-expire-seconds` Coherence*Web configuration parameter. It overrides the equivalent parameter in the `web.xml` file.

Table 3–4 describes the configuration parameters in the `glassfish-web.xml` file which are not valid when using Coherence*Web on GlassFish. If these parameters are configured in the `glassfish-web.xml` file, then they are ignored.

*Table 3–4 GlassFish Context Parameters that are not Valid for Coherence*Web in glassfish.web.xml*

Parameter Name	Reason Why it is Not Valid
`session-config/session-manager/manager-properties/maxSessions`	Coherence*Web controls session management. Because sessions can be shared across applications, it is not possible to count the number of sessions for a specific application.
`session-config/session-manager/manager-properties/ persistenceFrequency`	Coherence*Web always flushes session data to the cache at the end of a request (although it might do so asynchronously).
`session-config/session-manager/manager-properties/sessionFilename`	This parameter is relevant only for the `file` session persistence type.
`session-config/session-manager/store-properties/ directory`	This parameter is relevant only for the `file` session persistent type.
`session-config/session-manager/store-properties/ persistenceScope`	This parameter is relevant only for the GlassFish Server `native replicated` session persistent type.

Start a Cache Server

A Coherence cache server is responsible for storing and managing all cached data. It can be run either in a dedicated JVM (out-of-process) or within a GlassFish Server instance (in-process). The senior node (which is the first node) in a Coherence data cluster can take several seconds to start up; the start-up time required by subsequent nodes is minimal.

If you are using an out-of-process topology (storage-disabled GlassFish Server instances and stand alone Coherence cache servers), then start the cache servers first, followed by the GlassFish Server instances. This will ensure that there is minimal (measured in milliseconds) startup time for applications using Coherence. Any additional Web applications that use Coherence are guaranteed not to be the senior data member, so they will have minimal impact on GlassFish Server startup.

To Start a Stand-Alone Coherence Cache Server

1. Create a script for starting a Coherence cache server.

   The following is a sample script that starts a cache server. This example assumes that you are using a Sun JVM. See "JVM Tuning" in *Oracle Fusion Middleware Developing Applications with Oracle Coherence* for more information.

   ```
   java -server -Xms512m -Xmx512m
   -cp <Coherence installation dir>/lib/coherence-web.jar:<Coherence installation
   dir>/lib/coherence.jar -Dtangosol.coherence.management.remote=true
   ```

```
-Dtangosol.coherence.cacheconfig=default-session-cache-config.xml
-Dtangosol.coherence.session.localstorage=true
com.tangosol.net.DefaultCacheServer
```

You must include `coherence-web.jar` and `coherence.jar` on the classpath. For Coherence*Web, use the `default-session-cache-config.xml` file as the cache configuration file. This is the file you obtained in "Copy the Coherence*Web and Session Cache Files to the Application" on page 3-3. Note that the cache configuration defined for the cache server must match the cache configuration defined for the application servers which run on the same Coherence cluster.

If you have additional Coherence caches running on Coherence*Web, then you must merge the cache configuration information (typically defined in the `coherence-cache-config.xml` file) with the session configuration contained in the `default-session-cache-config.xml` file. The cache and session configuration must be consistent for the GlassFish Server and the Coherence cache servers.

For more information on merging these files, see "Merging Coherence Cache and Session Information" in *Oracle Fusion Middleware Integrating Oracle Coherence*.

2. Start one or more Coherence cache servers using the script described in the previous step.

To Start a Storage-Enabled or -Disabled GlassFish Server Instance

By default, a Coherence*Web-enabled GlassFish Server instance starts in storage-disabled mode.

There are several ways to start the GlassFish Server instance in storage-enabled mode. One way is to include the command line property `-Dtangosol.coherence.session.localstorage=true` in the server startup command.

Another way is to set the `local-storage` element in the `default-session-cache-config.xml` file to `true`, for example:

```
...
<local-storage>true</local-storage>
...
```

Configure Cluster Nodes

On the GlassFish Server, Coherence*Web can be configured only for EAR- or WAR-scoped cluster nodes. Because of the way that GlassFish Server class loaders work, it is not possible to configure application server-scoped cluster nodes. See "Cluster Node Isolation" on page 5-13 for more information on application server-, EAR-, and WAR-scoped cluster nodes.

Configuring EAR-Scoped Cluster Nodes

To use Coherence*Web in EAR-scoped cluster nodes:

1. Copy the `coherence.jar` and `coherence-web.jar` files to the `WEB-INF/lib` directory of each WAR in the EAR file.
2. Set the session persistence type to `coherence-web` in the `glassfish-web.xml` file, as illustrated in Example 3–2.

Example 3–2 Setting the Persistence Type in the glassfish-web.xml File

```
<glassfish-web-app>
   ...
   <session-config>
```

```
      <session-manager persistence-type="coherence-web"/>
   </session-config>
...
</glassfish-web-app>
```

3. Copy the session cache configuration file `default-session-cache-config.xml` to the `/WEB-INF/classes/` directory of each WAR file in the EAR file.

Configuring WAR-Scoped Cluster Nodes

To use Coherence*Web in WAR-scoped cluster nodes:

1. Copy the `coherence.jar` and `coherence-web.jar` files to the `/WEB-INF/lib/` directory of each Web application that you intend to deploy on GlassFish Server.
2. Set the session persistence type to `coherence-web` in the `glassfish-web.xml` file. See Example 3–2.
3. Copy the session cache configuration file `default-session-cache-config.xml` to the `/WEB-INF/classes/` directory of each Web application that you intend to deploy on GlassFish Server.

This packaging means that each deployed WAR file will create a Coherence node in the cluster. If you package multiple WAR files in an EAR file, then each WAR file will create a Coherence node in the cluster.

4 Using Coherence*Web on Other Application Servers

This chapter describes how to configure and deploy Coherence*Web, the session state persistence and management module, for use with a variety of application servers. The functionality that allows Coherence*Web to be used with these application servers is provided by running the automated Coherence*Web WebInstaller.

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

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.

This chapter contains the following sections:

- Installing Coherence*Web Using the WebInstaller
- Coherence*Web WebInstaller Ant Task
- Testing HTTP Session Management
- How the Coherence*Web WebInstaller Instruments a Java EE Application
- Installing Coherence*Web into Applications Using Java EE Security
- Preventing Cross-Site Scripting Attacks

Installing Coherence*Web Using the WebInstaller

Coherence*Web can be enabled for Java EE applications on several different Web containers. To do this, 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 inspection step and an installation step. For more information about what the installer does during these steps, see "How the Coherence*Web WebInstaller Instruments a Java EE Application" on page 4-10.

The installer can be run either from the Java command line or from Ant tasks. The following sections describe the Java command-line method. For Ant task-based installation, see "Coherence*Web WebInstaller Ant Task" on page 4-6.

Application Server-Specific Installation Instructions

All of the Web containers listed in "Supported Web Containers" on page 1-1 that can be installed with the WebInstaller share the same general installation instructions. These instructions are described in "General Instructions for Installing Coherence*Web Session Management Module" on page 4-2.

A few of the Web containers, such as Caucho Resin, and WebLogic 10.*n*, require extra, container-specific steps that you must complete before starting the general installation procedure. The following sections describe application server-specific installation steps:

- Installing on Oracle WebLogic Server 10.n
- Installing on Caucho Resin 3.1.n

Installing on Oracle WebLogic Server 10.*n*

Complete the following steps to install the Coherence*Web Session Management Module into Oracle WebLogic Server release 10 to 10.2:

1. Obtain the `coherence-web.jar` file from the `coherence/lib` directory.
2. For each WebLogic Server 10.*n* installation that will be running in the server cluster, update the libraries using the following command:

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

3. Follow the instructions described in "General Instructions for Installing Coherence*Web Session Management Module" on page 4-2 to complete the installation. Use the value `WebLogic/10.x` for the *server type*.

Installing on Caucho Resin 3.1.*n*

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

1. Obtain the `coherence-web.jar` file from the `coherence/lib` directory.
2. For each Caucho Resin installation that will be running in the server cluster, update the libraries using the following command:

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

3. Follow the instructions described in "General Instructions for Installing Coherence*Web Session Management Module" on page 4-2 to complete the installation. Use the value `Resin/3.1.x` for the *server type*.

General Instructions for Installing Coherence*Web Session Management Module

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 1-1.

If you are installing Coherence*Web for a Java EE application on an Apache Tomcat Server, see also "Enabling Sticky Sessions for Apache Tomcat Servers" on page 4-6 for additional instructions.

If you are installing Coherence*Web for a Java EE application on IBM WebSphere Server, see also "Decoding URL Session IDs for IBM WebSphere 7.n Servers" on page 4-6 for additional instructions.

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

1. Ensure 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. Ensure 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 following command line):

   ```
   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 file contains the default Coherence*Web settings for your application as recommended by the installer.

5. If necessary, review and modify the Coherence*Web settings based on your requirements.

   You can modify the Coherence*Web settings by editing the `coherence-web.xml` descriptor file. Appendix A, "Coherence*Web Context Parameters," describes the Coherence*Web settings that can be modified. Use the `param-name` and `param-value` subelements of the `context-param` parameter to enable the features you want. Table 4–1 describes some examples of different settings.

Table 4–1 Example Context Parameter Settings for Coherence*Web

Parameter	Name	Description
`coherence-servletcontext-clustered`	true	Clusters all `ServletContext` (global) attributes so that servers in a cluster share the same values for those attributes, and also receive the events specified by the Servlet Specification when those attributes change.
`coherence-enable-sessioncontext`	true	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.
`coherence-session-id-length`	32	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 purposely hijacked.
`coherence-session-urlencode-enabled`	true	By default, the `HttpSession` ID is managed in a cookie. If the application supports URL encoding, this option enables it.

6. Complete 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 file to reside in the same directory as the application. The command creates a `default-session-cache-config.xml` file in the `WEB-INF\classes` directory of the application archive file. This file contains the session and cache configuration information.

7. Deploy the updated application and verify that everything functions as expected, using the lightweight load balancer provided with the Coherence distribution. Remember that the lightweight load balancer is not a production-ready utility, in contrast to the load balancer provided by WebLogic Server.

The application can be deployed and run in any of the deployment topologies supported by Coherence: in-process, out-of-process, or out-of-process with Coherence*Extend. See the following sections for information on deploying and running your applications under these topologies. For more information on the topologies themselves, see "Deployment Topologies" on page 5-20.

Deploying and Running Applications In Process

Coherence*Web can be run *in-process* with the application server. This is where session data is stored with the application server. See "In-Process Topology" on page 5-20 for more information on this topology.

For the application server:

1. Start the application server in storage-enabled mode. Add the system property `tangosol.coherence.session.localstorage=true` to the Java options of your application server startup script.
2. Deploy the `coherence.jar` and `coherence-web.jar` files as shared libraries.
3. Deploy and run your application.

Deploying and Running Applications Out-of-Process

In the out-of-process deployment topology, a stand-alone cache server stores the session data and the application server is configured as a cache client. See "Out-of-Process Topology" on page 5-20 for more information on this topology.

The cache server and the application server must use the same cache and session configuration. This configuration is generated in the `default-session-cache-config.xml` file by the Coherence*Web WebInstaller. The WebInstaller generates the file in the `WEB-INF\classes` directory of the instrumented application.

For the cache server:

1. Add the `tangosol.coherence.cacheconfig` system property to the cache server startup script to locate the file configuration file. You must also include the system property `tangosol.coherence.session.localstorage=true` to enable storage for the cache server.
2. Add the `coherence.jar` and `coherence-web.jar` files to the classpath in the cache server startup script.

   Following is a sample startup script:

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

For the application server (cache client):

1. Deploy the `coherence.jar` and `coherence-web.jar` files as shared libraries.

2. The `default-session-cache-config.xml` file should already be present in the `WEB-INF\classes` directory of the instrumented application.

   By default, the file should specify that local storage is disabled (if you are not sure, you can either inspect the file to confirm that the `local-storage` element is set to `false` or add the system property `tangosol.coherence.session.localstorage=false` to the startup script).

3. Deploy the application to the server.

Migrating to Out-of-Process Topology

If you have been running and testing your application with Coherence*Web in-process, you can easily migrate to the out-of-process topology. Simply set up your cache server and application server as described in "Deploying and Running Applications Out-of-Process" on page 4-4.

Deploying and Running Applications 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 is over Coherence*Extend (TCP/IP). Coherence*Extend consists of two components: an extend client (or *proxy*) running outside the cluster and an extend proxy service running in the cluster hosted by one or more cache servers. See "Out-of-Process with Coherence*Extend Topology" on page 5-21 for more information on this topology.

In these deployments, there are three types of participants:

- Cache servers (storage servers), which are used to store the actual session data in memory.
- Web (application) servers, which are the Extend clients in this topology. They are not members of the cluster; instead, they connect to a proxy node in the cluster that will issue requests to the cluster on their behalf.
- Proxy servers, which are storage-disabled members (nodes) of the cluster that accept and manage TCP/IP connections from Extend clients. Requests that arrive from clients will be sent into the cluster, and responses will be sent back through the TCP/IP connections.

For the cache server:

Follow the instructions for configuring the cache server in "Deploying and Running Applications Out-of-Process" on page 4-4. Also, edit the cache server's copy of the `default-session-cache-config.xml` file to add the system properties `tangosol.coherence.session.proxy=false` and `tangosol.coherence.session.localstorage=true`.

See "Configure the Cache for Proxy and Storage JVMs" on page 5-22 for more information and an example of a `default-session-cache-config.xml` file with these context parameters.

For the Web tier (application) server:

Follow the instructions for configuring the application server in "Deploying and Running Applications Out-of-Process" on page 4-4. Also, complete these steps:

1. Ensure that Coherence*Web is configured to use the Optimistic Locking mode. Optimistic locking is the default locking mechanism for Coherence*Web (see"Optimistic Locking" on page 5-17).

2. Edit the application server's copy of the `default-session-cache-config.xml` file to add the proxy JVM host names, IP addresses and ports. To do this, add a `<remote-addresses>` section to the file. In most cases, you should include the host name and IP address, and port of all proxy JVMs for load balancing and failover.

See "Configure the Cache for Web Tier JVMs" on page 5-23 for more information and an example of a `default-session-cache-config.xml` file with a `<remote-addresses>` section.

For the proxy server:

With a few changes, the proxy server can use the same cache and session configuration as the application server and the cache server. Edit the `default-session-cache-config.xml` file to add these system properties:

- `tangosol.coherence.session.localstorage=false` to disable local storage.
- `tangosol.coherence.session.proxy=true` to indicate that a proxy service is being used.
- `tangosol.coherence.session.proxy.localhost` to indicate the host name or IP address of the NIC to which the proxy will bind.
- `tangosol.coherence.session.proxy.localport` to indicate a unique port number to which the proxy will bind.

See "Configure the Cache for Proxy and Storage JVMs" on page 5-22 for more information and an example of a `default-session-cache-config.xml` file with these context parameters.

Enabling Sticky Sessions for Apache Tomcat Servers

If you want to employ sticky sessions for the Apache Tomcat Server, you must configure the `jvmRoute` attribute in the server's `server.xml` file. You can find more information on this attribute at this URL:

`http://tomcat.apache.org/connectors-doc/reference/workers.html`

Decoding URL Session IDs for IBM WebSphere 7.*n* Servers

If set to `true`, the `coherence-session-urldecode-bycontainer` context parameter allows the container to decode the URL. This context parameter must be set to `false` if you are installing Coherence*Web for a Java EE application on release 7.*n* of the IBM WebSphere application server. Instead of the WebSphere application server, Coherence*Web will handle the decoding of session IDs.

The Coherence*Web WebInstaller, when run for the WebSphere 7.*n* application server type, will automatically set this parameter to `false` unless you explicitly set it to `true`.

Coherence*Web WebInstaller Ant Task

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

This section contains the following information:

- Using the Coherence*Web WebInstaller Ant Task
- Configuring the WebInstaller Ant Task
- WebInstaller Ant Task Examples

Using the Coherence*Web WebInstaller Ant Task

To use the Coherence*Web WebInstaller Ant task, add the task import statement illustrated in Example 4–1 to your Ant build file. In this example, `${coherence.home}` refers to the root directory of your Coherence installation.

Example 4–1 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 ordinarily 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 file.
4. Run the Coherence*Web Ant task with the `operations` attribute set to `install`.

Performing Iterative Development

If you are performing iterative development on your application, such as modifying JavaServer Pages (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 file (from Step 2 of "Using the Coherence*Web WebInstaller Ant Task").
2. Build your Java EE application as you ordinarily would.
3. Run the Coherence*Web Ant task with the `operations` attribute set to `inspect`, and the `install` and `descriptor` attributes set to the location of the previously generated Coherence*Web XML descriptor file (from Step 2 of "Using the Coherence*Web WebInstaller Ant Task" on page 4-7).

Changing the Coherence*Web Configuration Settings of a Java EE Application

If you must change the Coherence*Web configuration settings of a Java EE application that is using Coherence*Web, follow these steps:

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 file for the Java EE application.
2. Change the necessary configuration parameters in the Coherence*Web XML descriptor file.
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 file (from Step 2 of "Using the Coherence*Web WebInstaller Ant Task" on page 4-7).

Configuring the WebInstaller Ant Task

Table 4–2 describes the attributes that can be used with the Coherence*Web WebInstaller Ant task.

Table 4–2 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 holds 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 file. This attribute defaults to the `coherence-web.xml` file in the directory that contains the target Java EE application.	No
`failonerror`	Stops the Ant build if the Coherence*Web installer exits with a status other than 0. The default is `true`.	No
`nowarn`	Suppresses 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`	Touches 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`	Displays verbose output. This attribute can be either `true` or `false`. The default is `false`.	No

WebInstaller Ant Task Examples

The following list provides sample commands for the WebInstaller Ant task.

- Inspect the `myWebApp.war` Web application and generate a Coherence*Web XML descriptor file 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 file 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 file 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 file 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 before 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 file 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"/>
```

Testing HTTP Session Management

Coherence comes with a lightweight software load balancer; it is intended only for testing purposes. The load balancer is very easy to use and is very useful when testing functionality such as session management. Follow these steps to test HTTP session management with the lightweight load balancer:

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. Ensure 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 `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 that you use the preceding 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: Ensure that your application uses only relative redirections or the address of the load balancer.

   Table 4–3 describes the command-line options for the load balancer:

Table 4–3 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 or 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 WebInstaller Instruments a Java EE Application

During the inspection 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 Context Parameters" for descriptions of all possible parameters.

   The WebInstaller sets the servlet container to start in storage-disabled mode (that is, it sets `tangosol.coherence.session.localstorage` to `false`).

   If an existing `coherence-web.xml` configuration file exists (for example, from a previous run of the Coherence*Web WebInstaller), the context parameters in the existing file are merged with those in the generated template.
2. Enumerates the JSP from each Web application in the target Java EE application and adds 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 installation 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 uninstallation step.
2. Adds the Coherence*Web configuration context parameters generated in Step 1 of the inspection step to the `web.xml` descriptor file 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` class in each `web.xml` descriptor file. At run time, the Coherence*Web `ServletContextListener` class propagates each `ServletContextEvent` event to each application-specific `ServletContextListener` listener.
5. Registers a Coherence*Web `ServletContextAttributeListener` listener in each `web.xml` descriptor file. At run time, the Coherence*Web `ServletContextAttributeListener` propagates each

`ServletContextAttributeEvent` event to each application-specific `ServletContextAttributeListener` listener.

6. Wraps each application-specific `Servlet` declared in each `web.xml` descriptor file with a Coherence*Web `SessionServlet`. At run time, each Coherence*Web `SessionServlet` delegates to the wrapped `Servlet`.
7. Adds the following directive to each JSP enumerated in Step 2 of the inspection step:

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

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

Installing Coherence*Web into Applications Using Java EE Security

To install Coherence*Web into an application that uses Java EE security, 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 that 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 causes two sessions to be associated with a given authenticated user:

- A Coherence*Web session that contains all session data created by the Web application
- A session created by the Web container during authentication that stores only information necessary to identify the user

Preventing Cross-Site Scripting Attacks

Use the `coherence-session-cookie-httponly` context parameter to append the `HttpOnly` attribute to the session cookie. The `HttpOnly` attribute is used to help prevent attacks such as cross-site scripting, since it does not allow the cookie to be accessed by a client-side script such as JavaScript. Note that not all browsers support this functionality. This context parameter is available for instrumented applications only.

5 Coherence*Web Session Management Features

This chapter describes the features of Coherence*Web, including session models, session scoping, session locking, deployment topologies, and logging. You can configure Coherence*Web in many ways to meet the demands of your environment. Consequently, you might have to change some default configuration options. This chapter provides an in-depth look at the features that Coherence*Web supports so that you can make the appropriate configuration and deployment decisions.

- Session Models, which describes how Coherence*Web stores session state
- Session and Session Attribute Scoping, which allows fine-grained control over how both session data and session attributes are scoped (or *shared*) across application boundaries
- Cluster Node Isolation, which determines the number of Coherence nodes that are created within an application server JVM and where the Coherence library is deployed in the application's classpath
- Session Locking Modes, which determines how applications will obtain concurrent access to HTTP sessions
- Deployment Topologies, which determines how the session data is stored and managed between the cache servers and application servers
- Accessing Sessions with Lazy Acquisition, which describes how to save processing time and power by directing Coherence*Web to acquire sessions only when the servlet or filter attempts to access it
- Overriding the Distribution of HTTP Sessions and Attributes, which describes how you can control whether a session or its attributes remain local (stored on the originating server's heap and accessible only by that server) or distributed (stored within the Coherence grid, and thus, accessible to other server JVMs)
- Detecting Changed Attribute Values, which describes how Coherence*Web tracks attributes retrieved from the session that may have changed during the course of processing a request
- Saving Non-Serializable Attributes Locally, which describes how Coherence*Web can handle session attributes that are not serializable.
- Securing Coherence*Web Deployments, which describes how to prevent unauthorized Coherence TCMP cluster members from accessing HTTP session cache servers by enabling Secure Socket Layer (SSL).
- Customizing the Name of the Session Cache Configuration File, which describes how you can choose a custom name for your session cache configuration file.

- Configuring Logging for Coherence*Web, which describes the types of logging which are supported for Coherence*Web.
- Getting Concurrent Access to the Same Session Instance, which describes how you can use a cache delegator to ensure that the local cache should be used for storing and retrieving the session instance before attempting to use the distributed cache.

Session Models

A session model describes how Coherence*Web stores the session state in Coherence. Session data is managed by an `HttpSessionModel` object while the session collection in a Web application is managed by an `HttpSessionCollection` object. You must configure only the collection type in the `web.xml` file—the model is implicitly derived from the collection type. Coherence*Web includes these different session model implementations:

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

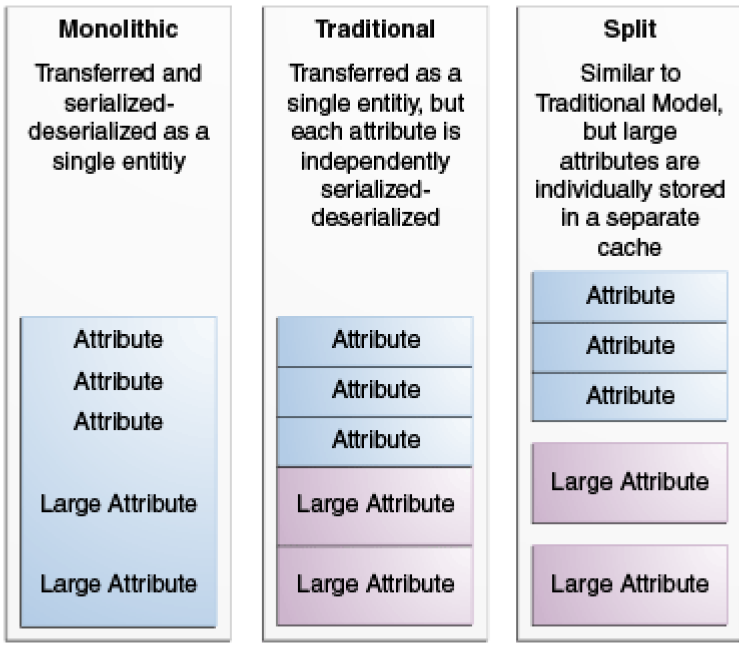
These sections provide additional information on session models:

- Session Model Recommendations, provides recommendations on which session model to choose for your applications
- Configuring a Session Model, describes how to change the session model by using a system property or a context parameter
- Sharing Data in a Clustered Environment, describes how data is shared between and within JVMs
- Scalability and Performance. describes the impact of session models on scalability and performance

Note: In general, Web applications that are part of the same Coherence cluster must use the same session model type. Inconsistent configurations could result in deserialization errors.

Figure 5–1 illustrates the three session models.

Figure 5–1 Traditional, Monolithic, and Split Session Models

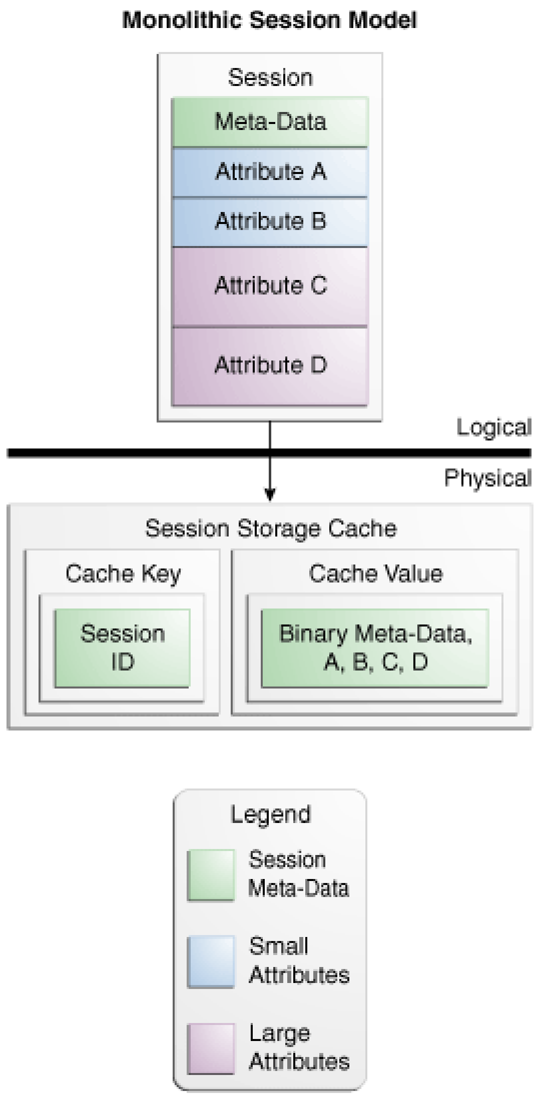

Monolithic
Transferred and serialized-deserialized as a single entitiy
Attribute
Attribute
Attribute
Large Attribute
Large Attribute
Traditional
Transferred as a single entitiy, but each attribute is independently serialized-deserialized
Attribute
Attribute
Attribute
Large Attribute
Large Attribute
Split
Similar to Traditional Model, but large attributes are individually stored in a separate cache
Attribute
Attribute
Attribute
Large Attribute
Large Attribute

Monolithic Model

The Monolithic model is represented by the `MonolithicHttpSessionModel` and `MonolithicHttpSessionCollection` objects. These are similar to the Traditional model, except that they solve the shared object issue by serializing and deserializing all attributes into a single object stream. As a result, the Monolithic model often does not perform as well as the Traditional model.

Figure 5–2 illustrates the relationship between the logical representation of data and its physical representation in the session storage cache. In its logical representation session data consists of metadata, and various attributes. In its physical representation in the session storage cache, the metadata and attributes are serialized into a single stream. A session ID is associated with the metadata and attributes.

Figure 5–2 Monolithic Session Model



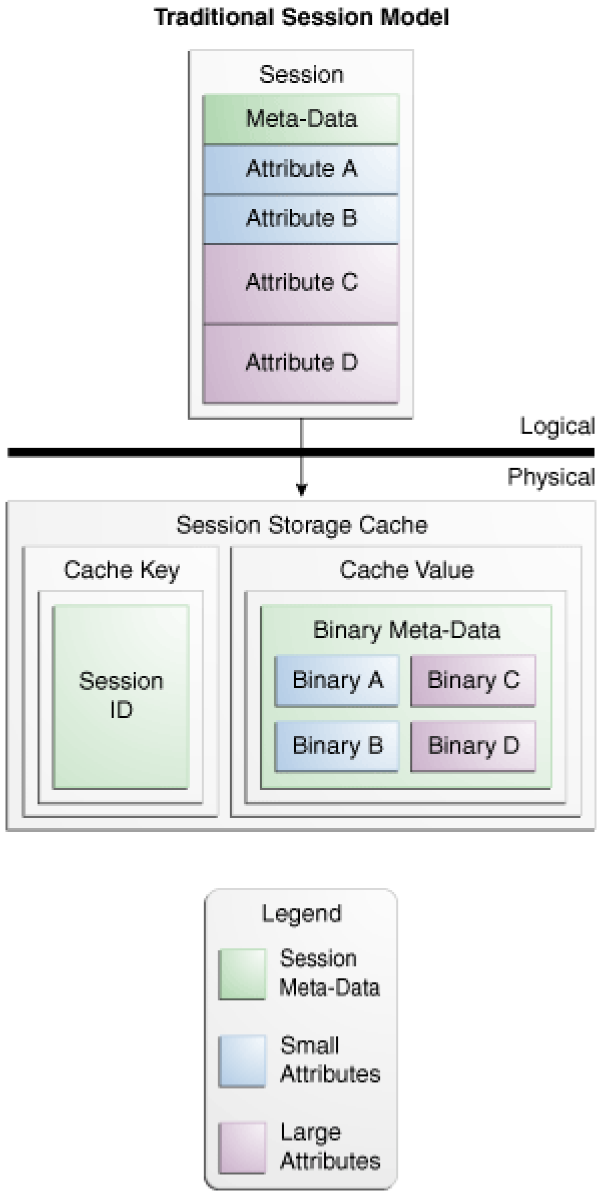
Traditional Model

The Traditional model is represented by the `TraditionalHttpSessionModel` and `TraditionalHttpSessionCollection` objects. The `TraditionalHttpSessionCollection` object stores an HTTP session object in a single cache, but serializes each attribute independently.

This model is suggested for applications with relatively small HTTP session objects (10 KB 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 cause multiple instances of that shared object to exist when the HTTP session is later deserialized.

Figure 5–3 illustrates the relationship between the logical representation of data and its physical representation in the session storage cache. In its logical representation session data consists of metadata, and various attributes. In its physical representation in the session storage cache, the metadata and attributes are converted to binaries, and a session ID is associated with them. Note that the attributes are serialized individually instead of as a single binary BLOB (such as in the Monolithic case).

Figure 5–3 Traditional Session Model



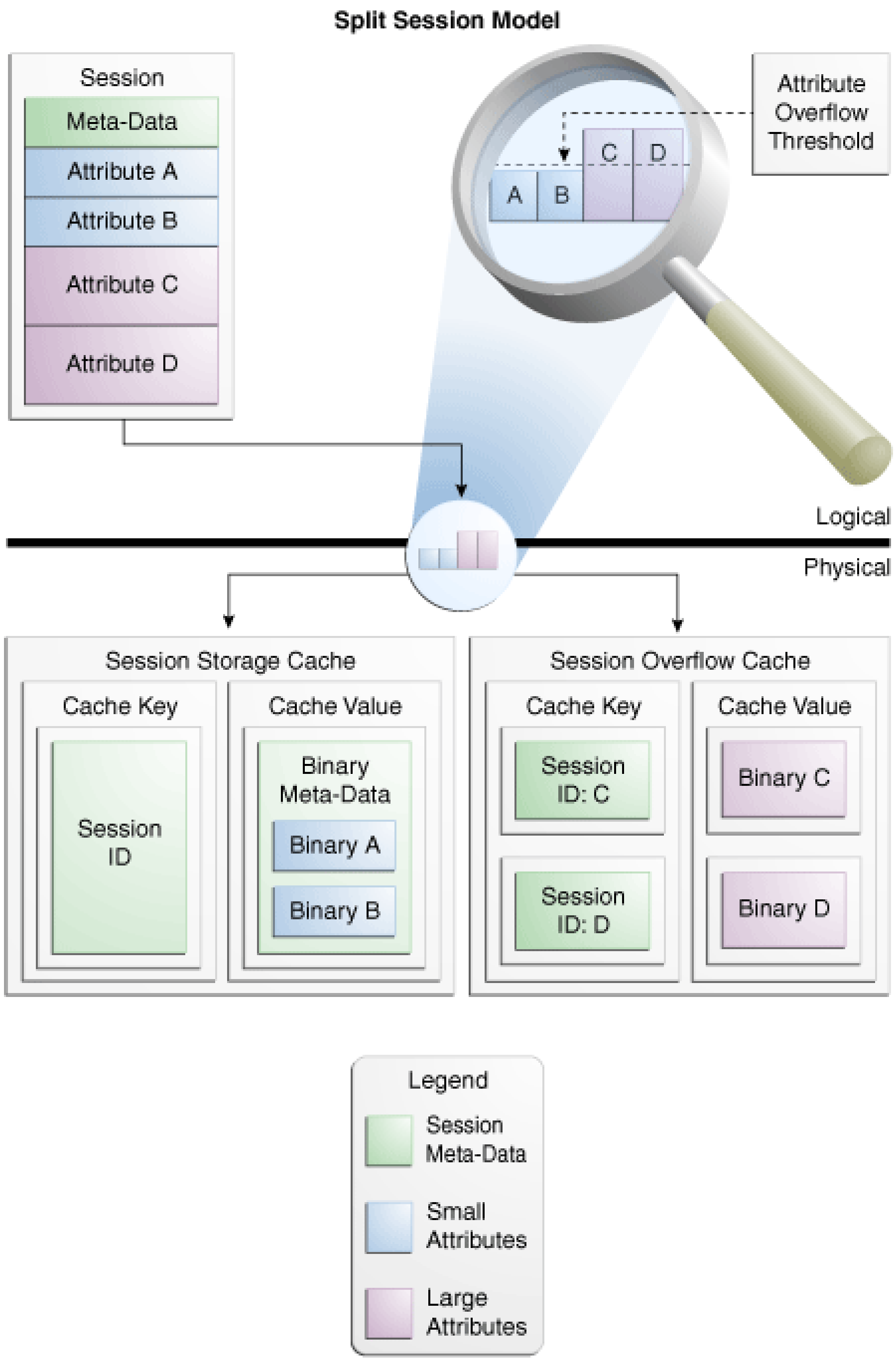
Split Model

The Split model is represented by the `SplitHttpSessionModel` and `SplitHttpSessionCollection` objects. `SplitHttpSessionCollection` is the default used by Coherence*Web.

These models store the core HTTP session metadata and all of the small session attributes in the same manner as the Traditional model, thus ensuring high performance by keeping that block of binary session data small. All large attributes are split 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 for each request. In other words, only the large attributes that are modified within a particular request 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 5–4 illustrates the relationship between the logical representation of data and its physical representation in the session storage cache. In this model, large objects are stored as separate cache entries with their own session ID.

Figure 5–4 Split Session Model



Session Model Recommendations

The following are recommendations on which session model to choose for your applications:

- The Split model is the recommended session model for most applications.

- The Traditional model might 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.

Note: See Appendix A, "Coherence*Web Context Parameters" for descriptions of the parameters used to configure session models.

Configuring a Session Model

By default, Coherence*Web uses the split session model, where large attributes are split into separate cache entries to be managed individually. You can change the session model used by Coherence*Web by configuring the `-Dcoherence.sessioncollection.class` system property or by setting the equivalent `coherence-sessioncollection-class` context parameter in the Web application's `web.xml` file. As the value of the context parameter (or system property), use the fully-qualified class name of the `HttpSessionCollection` implementation.

- `com.tangosol.coherence.servlet.SplitHttpSessionCollection` (default) configures the Split model.
- `com.tangosol.coherence.servlet.MonolithicHttpSessionCollection` configures the Monolithic model.
- `com.tangosol.coherence.servlet.TraditionalHttpSessionCollection` configures the Traditional model.

Example 5–1 illustrates a `web.xml` entry to configure the Monolithic model.

Example 5–1 Configuring the Session Model

```
...
<context-param>
   <param-name>coherence-sessioncollection-class</param-name>

<param-value>com.tangosol.coherence.servlet.MonolithicHttpSessionCollection</param
-value>
</context-param>
...
```

Sharing Data in a Clustered Environment

Clustering can boost scalability and availability for applications. Clustering solutions such as Coherence*Web solve many problems for developers, but successful developers must be aware of the limitations of the underlying technology, and how to manage those limitations. Understanding what the platform provides, and what users require, gives developers the ability to eliminate the gap between the two.

Session attributes must be serializable if they are to be processed across multiple JVMs, which is a requirement for clustering. It is possible to make some fields of a session attribute non-clustered by declaring those fields as transient. While this eliminates the requirement for all fields of the session attributes to be serializable, it also means that these attributes are not fully replicated to the backup server(s). Developers who follow this approach should be very careful to ensure that their applications are capable of operating in a consistent manner even if these attribute fields are lost. In most cases, this approach ends up being more difficult than simply converting all session attributes to serializable objects. However, it can be a useful pattern when very large amounts of user-specific data are cached in a session.

The Java EE Servlet specification (versions 2.2, 2.3, and 2.4) states that the servlet context should not be shared across the cluster. Non-clustered applications that rely on the servlet context as a singleton data structure have porting issues when moving to a clustered environment.

A more subtle issue that arises in clustered environments is the issue of object sharing. In a non-clustered application, if two session attributes reference a common object, changes to the shared object are visible as part of both session attributes. However, this is not the case in most clustered applications. To avoid unnecessary use of compute resources, most session management implementations serialize and deserialize session attributes individually on demand. Coherence*Web (Traditional and Split session models) normally operates in this manner. If two session attributes that reference a common object are separately deserialized, the shared common object is instantiated twice. For applications that depend on shared object behavior and cannot be readily corrected, Coherence*Web provides the option of a Monolithic session model, which serializes and deserializes the entire session object as a single operation. This provides compatibility for applications that were not originally designed with clustering in mind.

Many projects require sharing session data between different Web applications. The challenge that arises is that each Web application typically has its own class loader. Consequently, objects cannot readily be shared between separate Web applications. There are two general methods used as a work around, each with its own set of trade-offs.

- Place common classes in the Java CLASSPATH, allowing multiple applications to share instances of those classes at the expense of a slightly more complicated configuration.
- Use Coherence*Web to share session data across class loader boundaries. Each Web application is treated as a separate cluster member, even if they run within the same JVM. This approach provides looser coupling between Web applications (assuming serialized classes share a common serial Version UID), but suffers from a performance impact because objects must be serialized-deserialized for transfer between cluster members.

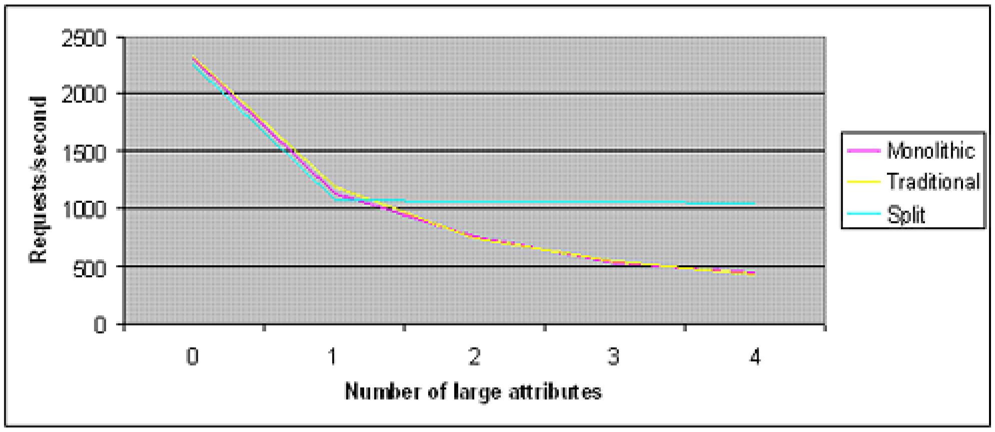
Scalability and Performance

Moving to a clustered environment makes session size a critical consideration. Memory usage is a factor regardless of whether an application is clustered or not, but clustered applications must also consider the increased CPU and network load that larger sessions introduce. While non-clustered applications using in-memory sessions are not required to serialize-deserialize session state, clustered applications must do this every time session state is updated. Serializing session state and then transmitting it over the network becomes a critical factor in application performance. For this reason and others, a server should generally limit session size to no more than a few kilobytes.

While the Traditional and Monolithic session models for Coherence*Web have the same limiting factor, the Split session model was explicitly designed to efficiently support large HTTP sessions. Using a single clustered cache entry to contain all of the small session attributes means that network traffic is minimized when accessing and updating the session or any of its smaller attributes. Independently deserializing each attribute means that CPU usage is minimized. By splitting out larger session attributes into separate clustered cache entries, Coherence*Web ensures that the application only pays the cost for those attributes when they are actually accessed or updated. Additionally, because Coherence*Web leverages the data management features of Coherence, all of the underlying features are available for managing session attributes, such as near caching, NIO buffer caching, and disk-based overflow.

Figure 5–5 illustrates performance as a function of session size. Each session consists of ten 10-character Strings and from zero to four 10,000-character Strings. Each HTTP request reads a single small attribute and a single large attribute (for cases where there are any in the session), and 50 percent of requests update those attributes. Tests were performed on a two-server cluster. Note the similar performance between the Traditional and Monolithic models; serializing-deserializing Strings consumes minimal CPU resources, so there is little performance gain from deserializing only the attributes that are actually used. The performance gain of the Split model increases to over 37:1 by the time session size reaches one megabyte (100 large Strings). In a clustered environment, it is particularly true that application requests that access only essential data have the opportunity to scale and perform better; this is part of the reason that sessions should be kept to a reasonable size.

Figure 5–5 Performance as a Function of Session Size



Another optimization is the use of transient data members in session attribute classes. Because Java serialization routines ignore transient fields, they provide a very convenient means of controlling whether session attributes are clustered or isolated to a single cluster member. These are useful in situations where data can be "lazy loaded" from other data sources (and therefore recalculated during a server failover process), and also in scenarios where absolute reliability is not critical. If an application can withstand the loss of a portion of its session state with zero (or acceptably minimal) impact on the user, then the performance benefit may be worth considering. In a similar vein, it is not uncommon for high-scale applications to treat session loss as a session timeout, requiring the user to log back in to the application (which has the implicit benefit of properly setting user expectations regarding the state of their application session).

Sticky load balancing plays a critical role because session state is not globally visible across the cluster. For high-scale clusters, user requests normally enter the application tier through a set of stateless load balancers, which redistribute (more or less randomly) these requests across a set of sticky load balancers, such as Microsoft IIS or Apache HTTP Server. These sticky load balancers are responsible for the more computationally intense act of parsing the HTTP headers to determine which server instance is processing the request (based on the server ID specified by the session cookie). If requests are misrouted for any reason, session integrity is lost. For example, some load balancers may not parse HTTP headers for requests with large amounts of

POST data (for example, more than 64KB), so these requests are not routed to the appropriate server instance. Other causes of routing failure include corrupted or malformed server IDs in the session cookie. Most of these issues can be handled with proper selection of a load balancer and designing tolerance into the application whenever possible (for example, ensuring that all large POST requests avoid accessing or modifying session state).

Sticky load balancing aids the performance of Coherence*Web but is not required. Because Coherence*Web is built on the Coherence data management platform, all session data is globally visible across the cluster. A typical Coherence*Web deployment places session data in a near cache topology, which uses a partitioned cache to manage huge amounts of data in a scalable and fault-tolerant manner, combined with local caches in each application server JVM to provide instant access to commonly used session state. While a sticky load balancer is not required when Coherence*Web is used, there are two key benefits to using one. Due to the use of near cache technology, read access to session attributes is instant if user requests are consistently routed to the same server, as using the local cache avoids the cost of deserialization and network transfer of session attributes. Additionally, sticky load balancing allows Coherence to manage concurrency locally, transferring session locks only when a user request is rebalanced to another server.

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 session 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 session cookie path to a common context path for all Web applications that share session data. For example, to share session data between two Web applications registered under the context 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 a slash (`/`).

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 configure the session cookie domain to a domain shared by the machines. For example, 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` context 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.

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 might 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 Enterprise JavaBeans (EJB) tiers but use 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. One way to do this is to use the `ApplicationScopeController` interface to define the scope of an application's attributes. "Session Attribute Scoping" on page 5-12 describes this technique. Another way is to specify a unique session cache service name for each application.

Follow these steps to specify a unique session cache service name for each application:

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

   This forces each application to use a separate clustered service for session data.
3. Include the modified `default-session-cache-config.xml` file with the application.

Example 5–2 illustrates a sample `default-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>` element for the replicated scheme to `ReplicationSessionsMiscHRP`. Rename the `<service-name>` element for the distributed schemes to `DistributedSessionsHRP`.

Example 5–2 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>
```

Working with Multiple Cache Configurations

If you are working with two or more applications running under Coherence*Web, then they could have multiple different cache configurations. In this case, the cache configuration on the cache server must contain the union of these cache configurations regardless of whether you run in storage-enabled or storage-disabled mode. This will allow the applications to be supported in the same cache cluster.

Keeping Session Cookies Separate

If you are using cookies to store session IDs, you must ensure 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. To prevent cookies from being propagated, ensure that no two applications share the same context path.

For example, assume you have two Web applications registered under the context paths `/web/HRPortal` and `/web/InWeb`. To prevent the Web applications from sharing session data through cookies, set the cookie path to `/web/HRPortal` in one application, and set the cookie path to `/web/InWeb` in the other application.

If your applications are deployed on different Web containers running on separate machines, then you can configure the 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, set the cookie domain in one application to `server1.mydomain.com`, and set the cookie domain in the other application to `server2.mydomain.com`.

Session Attribute Scoping

In the case where sessions are shared across Web applications there are many instances where the application might 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 can selectively scope attributes in cases when a session might be shared across multiple applications. This allows different applications to potentially use the same attribute names for the application-scope state without accidentally reading, updating, or removing other applications' attributes. In addition to having application-scoped information in the session, this interface allows the session to contain global (unscoped) information that can be read, updated, and removed by any of the applications that shares the session.

Two implementations of the `AttributeScopeController` interface are available: `ApplicationScopeController` and `GlobalScopeController`. The `GlobalScopeController` implementation does not scope attributes, while `ApplicationScopeController` scopes all attributes to the application by prefixing the name of the application to all attribute names.

Use the `coherence-application-name` context parameter to specify the name of the application (and the Web module in which the application appears). The `ApplicationScopeController` interface will use the name of the application to scope the attributes. If you do not configure this parameter, then Coherence*Web uses the name of the class loader instead. For more information, see the description of `coherence-application-name` in Table 2–1.

Note: After a configured `AttributeScopeController` implementation is created, it is initialized with the name of the Web application, which it can use to qualify attribute names. Use the `coherence-application-name` context parameter to configure the name of your Web application.

Sharing Session Information Between Multiple Applications

Coherence*Web allows multiple applications to share the same session object. To do this, the session attributes must be visible to all applications. You must also specify which URLs served by WebLogic Server will be able to receive cookies.

To allow the applications to share and modify the session attributes, reference the `GlobalScopeController` (`com.tangosol.coherence.servlet.AbstractHttpSessionCollection$GlobalScopeController`) interface as the value of the `coherence-scopecontroller-class` context parameter in the `web.xml` file. `GlobalScopeController` is an implementation of the `com.tangosol.coherence.servlet.HttpSessionCollection$AttributeScopeController` interface that allows individual session attributes to be globally visible.

Example 5–3 illustrates the `GlobalScopeController` interface specified in the `web.xml` file.

Example 5–3 GlobalScopeController Specified in the web.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
  <web-app>
    ...
    <context-param>
      <param-name>coherence-scopecontroller-class</param-name>
      <param-value>com.tangosol.coherence.servlet.
AbstractHttpSessionCollection$GlobalScopeController</param-value>
    </context-param>
    ...
  </web-app>
```

Cluster Node Isolation

There are several different ways in which you can deploy Coherence*Web. One of the things to consider when deciding on a deployment option is cluster node isolation. Cluster node isolation considers:

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

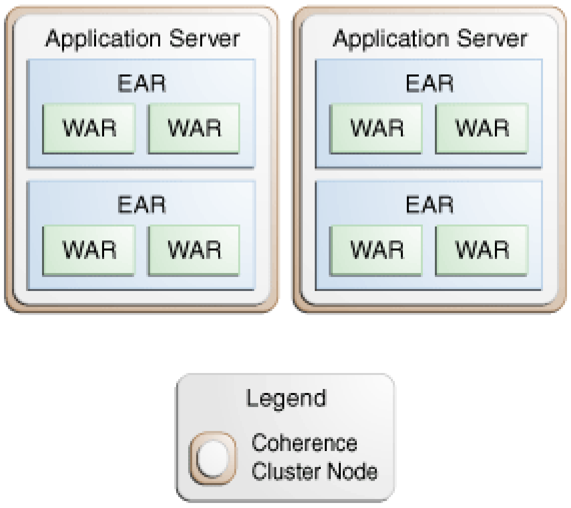
Applications can be application server-scoped, EAR-scoped, or WAR-scoped. This section describes these considerations. For detailed information about the XML configuration for each of these options, see "Configure Coherence*Web Storage Mode" on page 2-11.

Application Server-Scoped Cluster Nodes

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

Figure 5–6 illustrates an application server-scoped cluster with two cluster nodes (application server instances). Because Coherence*Web has been deployed to each instance's class path, each instance can be considered to be a Coherence node. Each node contains two EAR files; each EAR file contains two WAR files. All of the application running in each instance share the same Coherence library and classes.

Figure 5–6 Application Server-Scoped Cluster



For WebLogic Server, all Coherence*Web-enabled applications have application server scope. "Configure Coherence*Web Storage Mode" on page 2-11 describes the XML configuration requirements for application server-scoped cluster nodes for WebLogic Server.

All Coherence*Web-enabled applications have application server scope. Application server scope is not available for GlassFish Server.

Note: Consider the use of the application server-scoped cluster configuration very carefully. Do not use it in environments where application interaction is unknown or unpredictable.

An example of such an environment might 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—the likelihood of collisions between namespaces for caches, services, and other configuration settings is quite high and could lead to unexpected results.

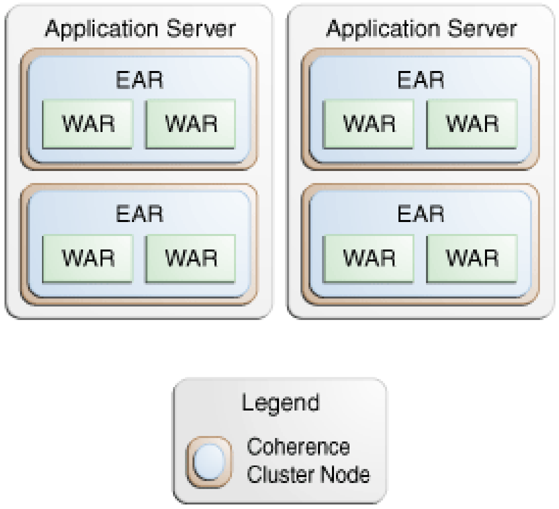
For these reasons, Oracle Coherence strongly recommends that you use EAR-scoped and WAR-scoped cluster node configurations. If you are in doubt regarding which deployment topology to choose, or if this warning applies to your deployment, then *do not* choose the application server-scoped cluster node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

EAR-Scoped Cluster Nodes

With this configuration, all deployed applications within each EAR file become part of one Coherence node. This configuration produces one Coherence node for each deployed EAR file that uses Coherence*Web. Because the Coherence library (`coherence.jar`) is deployed in the application's classpath, only one copy of the Coherence classes is loaded for each EAR file. Since all Web applications in the EAR file use the same cluster node, all Web applications in the EAR file are affected if one of the Web applications malfunctions.

Figure 5–7 illustrates four EAR-scoped cluster nodes. Since Coherence*Web has been deployed to each EAR file, each EAR file becomes a cluster node. All applications running inside each EAR file have access to the same Coherence libraries and classes.

Figure 5–7 EAR-Scoped Cluster



EAR-scoped cluster nodes reduce the deployment effort because no changes to the application server class path are required. This option is also ideal if you plan to deploy only one EAR file to an application server.

For more information on XML configuration requirements for EAR-scoped cluster nodes, see "Configuring EAR-Scoped Cluster Nodes" on page 3-7.

Note: This configuration is not available for Coherence*Web applications running on the WebLogic Server platform. Applications running on the WebLogic Server platform can be only application server-scoped.

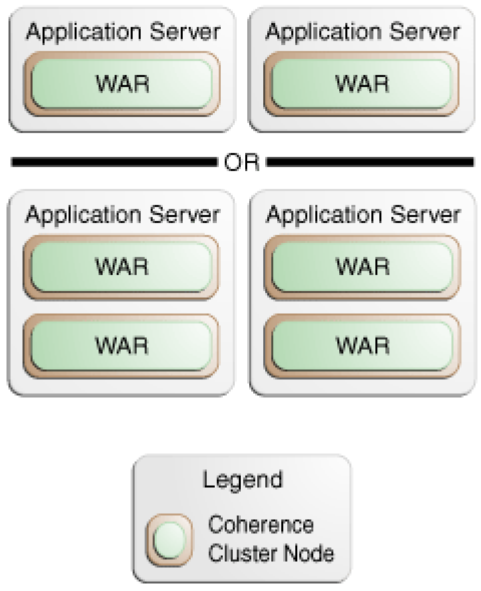
WAR-Scoped Cluster Nodes

With this configuration, each deployed Web application becomes its own Coherence node. This configuration produces the largest number of Coherence nodes in the cluster (one for each deployed WAR file that uses Coherence*Web) and because the Coherence library (`coherence.jar`) is deployed in the Web application's class path, there will be as many copies of the Coherence classes loaded as there are deployed WAR files. This results in the largest resource utilization of the three options. However, because each deployed Web application is its own cluster node, Web applications are completely isolated from other potentially malfunctioning Web applications.

WAR scoped cluster nodes reduce the deployment effort because no changes to the application server class path are required. This option is also ideal if you plan to deploy only one WAR file to an application server.

Figure 5–8 illustrates two different configurations of WAR files in application servers. Because each WAR file contains a copy of Coherence*Web (and Coherence), it can be considered a cluster node.

Figure 5–8 WAR-Scoped Clusters



For more information on XML configuration requirements for WAR-scoped cluster nodes, see "Configuring WAR-Scoped Cluster Nodes" on page 3-8.

Note: This configuration is not available for Coherence*Web applications running on the WebLogic Server platform. Applications running on the WebLogic Server platform can be only application server-scoped.

Session Locking Modes

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

- Optimistic Locking, which allows concurrent access to a session by multiple threads in a single member or multiple members, while prohibiting concurrent modification.
- Last-Write-Wins Locking, which is a variation of Optimistic Locking. This allows concurrent access to a session by multiple threads in a single member or multiple members. In this case, the last write is saved. This is the default locking mode.
- Member Locking, which allows concurrent access and modification of a session by multiple threads in the same member, while prohibiting concurrent access by threads in different members.
- Application Locking, which allows concurrent access and modification of a session by multiple threads in the same Web application instance, while prohibiting concurrent access by threads in different Web application instances.
- Thread Locking, which prohibits concurrent access and modification of a session by multiple threads in a single member.

Note: Generally, Web applications that are part of the same cluster must use the same locking mode and sticky session optimizations setting. Inconsistent configurations could result in deadlock.

You can specify the session locking mode used by your Web applications by setting the `coherence-session-locking-mode` context parameter. Table 5–1 lists the context parameter values and the corresponding session locking modes they specify. For more information about the `coherence-session-locking-mode` context parameter, see the following sections and Appendix A, "Coherence*Web Context Parameters."

Table 5–1 Summary of coherence-session-locking-mode Context Parameter Values

Locking Mode	coherence-session-locking-mode Values
Optimistic Locking	`optimistic`
Last-Write-Wins Locking	`none`
Member Locking	`member`
Application Locking	`app`
Thread Locking	`thread`

Optimistic Locking

Optimistic Locking mode allows multiple Web container threads in one or more members 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. The exception

`ConcurrentModificationException` is thrown when the session is flushed to the cache, which is after the Servlet request has finished processing. To view the exception, set the `weblogic.debug.DebugHttpSessions` system property to `true` in the container's startup script (for example: `-Dweblogic.debug.DebugHttpSessions=true`).

The Optimistic Locking mode can be configured by setting the `coherence-session-locking-mode` parameter to `optimistic`.

Last-Write-Wins Locking

Coherence*Web is configured with Last-Write Wins Locking by default. Last-Write-Wins Locking mode is a variation on the Optimistic Locking mode. It allows multiple Web container threads in one or more members to access the same session concurrently. This setting does not use explicit locking; it does not prevent concurrent updates upon completion of an HTTP request that modifies the session. Instead, the last write, that is, the last modification made, is allowed to modify the session.

The Last-Write-Wins Locking mode can be configured by setting the `coherence-session-locking-mode` parameter to `none`. This value will allow concurrent modification to sessions with the last update being applied.

Member Locking

The Member Locking mode allows multiple Web container threads in the same cluster node to access and modify the same session concurrently, but prohibits concurrent access by threads in different members. This is accomplished by acquiring a member-level lock for an HTTP session when the session is acquired. The lock is released on completion of the of the HTTP request. For more information about member-level locks, see `<lease-granularity>` in the "distributed-scheme" section of *Oracle Fusion Middleware Developing Applications with Oracle Coherence Oracle*.

The Member Locking mode can be configured by setting the `coherence-session-locking-mode` parameter to `member`.

Application Locking

The Application Locking mode restricts session access (and modification) to threads in a single Web application instance at a time. This is accomplished by acquiring both a member-level and application-level lock for an HTTP session when the session is acquired, and releasing both locks upon completion of the HTTP request. For more information about member-level locks, see `<lease-granularity>` in the "distributed-scheme" section of *Oracle Fusion Middleware Developing Applications with Oracle Coherence*.

The Application Locking mode can be configured by setting the `coherence-session-locking-mode` parameter to `app`.

Thread Locking

Thread Locking mode restricts session access (and modification) to a single thread in a single member at a time. This is accomplished by acquiring both a member level, application-level, and thread-level lock for an HTTP session when the session is acquired, and releasing all three locks upon completion of the request. For more information about member-level locks, see `<lease-granularity>` in the "distributed-scheme" section of the *Oracle Fusion Middleware Developing Applications with Oracle Coherence*.

The Thread Locking mode can be configured by setting the `coherence-session-locking-mode` parameter to `thread`.

Troubleshooting Locking in HTTP Sessions

Enabling Member, Application, or Thread Locking for HTTP session access indicates that Coherence*Web will acquire a clusterwide lock for every HTTP request that requires access to a session. By default, threads that attempt to access a locked session (locked by a thread in a different member) block access until the lock can be acquired. If you want to enable a timeout for lock acquisition, configure it with the `coherence-session-get-lock-timeout` context parameter, for example:

```
...
<context-param>
    <param-name>coherence-session-get-lock-timeout</param-name>
    <param-value>30</param-value>
  </context-param>
...
```

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 clusterwide locks for every HTTP request is eliminated.
- Requests can be load-balanced away from failing or unresponsive members to active members without requiring the unresponsive member to release the clusterwide lock on the session.

Coherence*Web provides a diagnostic invocation service that is executed when a member cannot acquire the cluster lock for a session. You can control if this service is enabled by setting the `coherence-session-log-threads-holding-lock` context parameter. If this context parameter is set to `true` (default), then the invocation service will cause the member that has ownership of the session to log the stack trace of the threads that are currently holding the lock.

Note that the `coherence-session-log-threads-holding-lock` context parameter is available only when the `coherence-sticky-sessions` context parameter is set to `true`. This requirement exists because Coherence Web will acquire a cluster-wide lock for every session access request unless sticky session optimization is enabled. By enabling sticky session optimization, frequent lock-holding, and the subsequent production of numerous log files, can be avoided.

Like all Coherence*Web messages, the Coherence `logging-config` operational configuration element controls how the message is logged. For more information on how to configure logging in Coherence, see the description of `logging-config`, in "Operation Configuration Elements" in *Oracle Fusion Middleware Developing Applications with Oracle Coherence*.

Enabling Sticky Session Optimizations

If Member, Application, or Thread Locking is a requirement for a Web application that resides behind a sticky load balancer, Coherence*Web provides an optimization for obtaining the clusterwide lock required for HTTP session access. By definition, a sticky load balancer attempts to route each request for a given session to the same application server JVM that it previously routed requests to for that same session. This should be the same application server JVM that created the session. The sticky session optimization takes advantage of this behavior by retaining the clusterwide lock for a session until the session expires or until it is asked to release it. If, for whatever reason, the sticky load balancer sends a request for the same session to another application server JVM, that JVM will ask the JVM that owns the lock on the session to release the lock as soon as possible. For more information, see the `SessionOwnership` entry in Table C–2.

Sticky session optimization can be enabled by setting the `coherence-sticky-sessions` context parameter to `true`. This setting requires that Member, Application, or Thread Locking is enabled.

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.

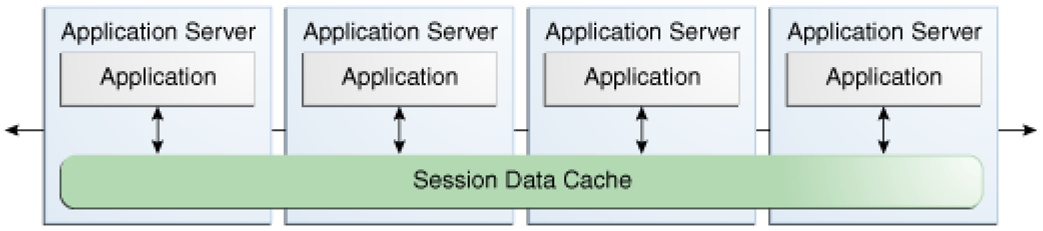
- In-Process Topology, also known as *local storage enabled*, is where session data is stored *in-process* with the application server
- Out-of-Process Topology, also known as *local storage disabled*, is where the application servers are configured as cache clients and dedicated JVMs run as cache servers, physically storing and managing the clustered data.
- Out-of-Process with Coherence*Extend Topology, means communication between the application server tier and the cache server tier are over Coherence*Extend (TCP/IP).

In-Process Topology

The in-process topology is not recommended for production use and 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, developing and testing. In this topology, local storage is enabled (that is, `tangosol.coherence.distributed.localstorage=true`).

Figure 5–9 illustrates the in-process topology. All of the application servers communicate with the same session data cache.

Figure 5–9 In-Process Deployment Topology



Out-of-Process Topology

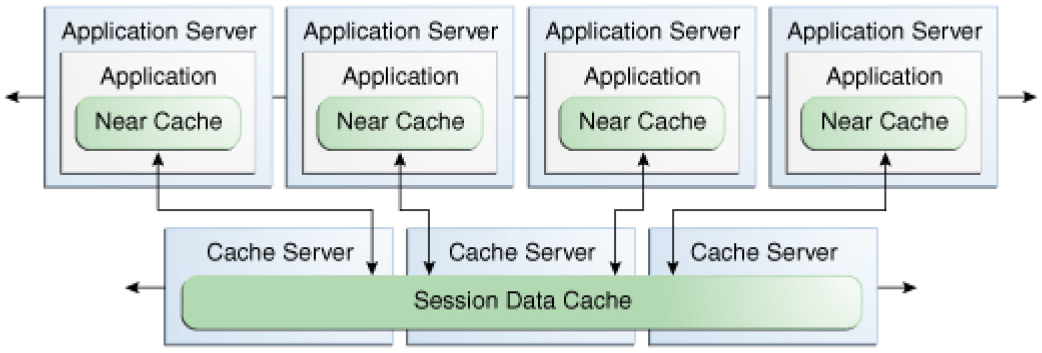
For the out-of-process deployment topology, the application servers (that is, application server tier) are configured as cache clients (that is, `tangosol.coherence.distributed.localstorage=false`) and there are dedicated JVMs running as cache servers, physically storing and managing the clustered data.

This approach has these benefits:

- Session data storage is offloaded from the application server tier to the cache server tier. This reduces heap usage, garbage collection times, and so on.
- The application and cache server tiers can be scaled independently. 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 5–10 illustrates the out-of-process topology. Each of the servers in the application tier maintain their own near cache. These near caches communicate with the session data cache which runs in a separate cache server tier.

Figure 5–10 Out-of-Process Deployment Topology



Migrating from In-Process to Out-of-Process Topology

You can easily migrate your application from an in-process to an out of process topology. To do this, you must run a cache server in addition to the application server. Start the cache server in storage-enabled mode and ensure that it references the same session and cache configuration file (`default-session-cache-config.xml`) that the application server uses. Start the application server in storage-disabled mode. See "Migrating to Out-of-Process Topology" on page 4-5 for detailed information.

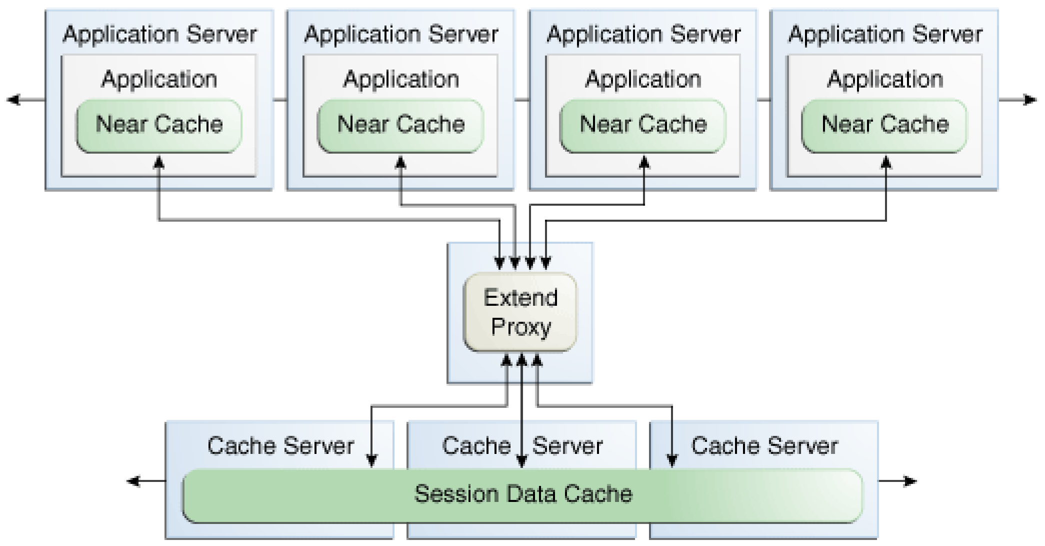
Out-of-Process with Coherence*Extend Topology

Coherence*Extend consists of two components: an extend client (or *proxy*) running outside the cluster and an extend proxy service running in the cluster hosted by one or more cache servers. 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 is over Coherence*Extend (TCP/IP). For information about configuring this scenario, see "Configuring Coherence*Web with Coherence*Extend" on page 5-22. For information about Coherence*Extend, see *Oracle Fusion Middleware Developing Remote Clients for Oracle Coherence*.

This approach has the same benefits as the out-of-process topology and the ability to divide the deployment of application servers and cache servers into segments. 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 5–11 illustrates the out-of-process with Coherence*Extend topology. Near caches in the servers in the application server tier use an extend proxy to communicate with the session data cache in the cache server tier.

Figure 5–11 Out-of-Process with Coherence*Extend Deployment Topology



Configuring Coherence*Web with Coherence*Extend

One of the deployment options for Coherence*Web is to use Coherence*Extend to connect Web container JVMs to the cluster by using TCP/IP. This configuration should be considered if any of the following situations applies:

- The Web tier JVMs are in a DMZ while the Coherence cluster is behind a firewall.
- The Web tier is in an environment that does not support UDP.
- Web tier JVMs experience long or frequent garbage collection (GC) pauses.
- Web tier JVMs are restarted frequently.

In these deployments, there are three types of participants:

- Web tier JVMs, which are the Extend clients in this topology. They are not members of the cluster; instead, they connect to a proxy node in the cluster that will issue requests to the cluster on their behalf.
- Proxy JVMs, which are storage-disabled members (nodes) of the cluster that accept and manage TCP/IP connections from Extend clients. Requests that arrive from clients will be sent into the cluster, and responses will be sent back through the TCP/IP connections.
- Storage JVMs, which are used to store the actual session data in memory.

To Configure Coherence*Web to Use Coherence*Extend

1. Configure Coherence*Web to use the Optimistic Locking mode (see "Optimistic Locking" on page 5-17).
2. Configure a cache configuration file for the proxy and storage JVMs (see "Configure the Cache for Proxy and Storage JVMs" on page 5-22).
3. Modify the Web tier cache configuration file to point to one or more of the proxy JVMs (see "Configure the Cache for Web Tier JVMs" on page 5-23).

Configure the Cache for Proxy and Storage JVMs

The session cache configuration file (`WEB-INF/classes/default-session-cache-config.xml`) is an example

Coherence*Web session cache configuration file that uses Coherence*Extend. Use this file for the proxy and server JVMs. It contains system property overrides that allow the same file to be used for both proxy and storage JVMs.

When used by a proxy JVM, the system properties described in Table 5–2 should be specified.

> **Note:** If you are writing applications for the WebLogic Server platform and you are using a customized session cache configuration file, then the file must be packaged in a GAR file for deployment. For more information, see "Using a Custom Session Cache Configuration File" on page 2-13.
>
> For more information on the packaging requirements for a GAR file, see also "Packaging Coherence Applications for WebLogic Server" in *Oracle Fusion Middleware Administering Oracle Coherence* and "Creating Coherence Applications for Oracle WebLogic Server" in *Developing Oracle Coherence Applications for Oracle WebLogic Server*.

Table 5–2 System Property Values for Proxy JVMs

System Property Name	Value
`tangosol.coherence.session.localstorage`	`false`
`tangosol.coherence.session.proxy`	`true`
`tangosol.coherence.session.proxy.localhost`	The host name or IP address of the NIC to which the proxy will bind.
`tangosol.coherence.session.proxy.localport`	A unique port number to which the proxy will bind.

When used by a cache server, specify the system properties described in Table 5–3.

Table 5–3 System Property Values for Storage JVMs

System Property Name	Value
`tangosol.coherence.session.localstorage`	`true`
`tangosol.coherence.session.proxy`	`false`

Configure the Cache for Web Tier JVMs

Coherence*Extend clients must also include a session cache configuration file. The file can be based on the `default-session-cache-config.xml` file that is found in the `coherence-web.jar` file.

To Install the Session Cache Configuration File for the Web Tier:

1. Extract the `default-session-cache-config.xml` file from the `coherence-web.jar` file.
2. Add proxy JVM host names and IP addresses and ports within the `<remote-addresses>` element. In most cases, you should include the host name and IP address, and port of all proxy JVMs for load balancing and failover. For example:

```
<remote-cache-scheme>
   <scheme-name>session-remote</scheme-name>
   <initiator-config>
```

```
        <serializer>
           <instance>
              <class-name>com.tangosol.io.DefaultSerializer</class-name>
           </instance>
        </serializer>
        <tcp-initiator>
           <remote-addresses>
              <socket-address>
                 <address>localhost</address>
                 <port>9099</port>
              </socket-address>
           </remote-addresses>
        </tcp-initiator>
     </initiator-config>
  </remote-cache-scheme>
```

Note: The `<remote-addresses>` element contains the proxy server(s) to which the Web container connects. By default, the Web container will pick an address at random (if there is more than one address in the configuration). If the connection between the Web container and the proxy is broken, the container will connect to another proxy in the list.

3. Rename the file to `default-session-cache-config-web-tier.xml`.
4. Place the file in the `WEB-INF/classes` directory of your Web application. If you used the WebInstaller to install Coherence*Web, replace the existing file that was added by the WebInstaller.

Accessing Sessions with Lazy Acquisition

By default, Web applications instrumented with the WebInstaller will always acquire a session whenever a servlet or filter is called. The session is acquired regardless of whether the servlet or filter actually needs a session. This can be expensive in terms of time and processing power if you run many servlets or filters that do not require a session.

To avoid this behavior, enable lazy acquisition by setting the `coherence-session-lazy-access` context parameter to `true` in the `web.xml` file. The session will be acquired only when the servlet or filter attempts to access it.

Overriding the Distribution of HTTP Sessions and Attributes

The Coherence*Web Session Distribution Controller, described by the `HttpSessionCollection.SessionDistributionController` interface, enables you to override the default distribution of HTTP sessions and attributes in a Web application. You override the default distribution by setting the `coherence-distributioncontroller-class` context parameter (see "Registering a Session Distribution Controller Implementation" on page 5-26). The value of the context parameter indicates an implementation of the `SessionDistributionController` interface.

An implementation of the `SessionDistributionController` interface can identify sessions or attributes in any of the following ways:

- Distributed, where a *distributed* session or attribute is stored within the Coherence data grid, and thus, accessible to other server JVMs. All sessions (and their attributes) are managed in a distributed manner. This is the default behavior and is provided by the `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$DistributedController` implementation of the `SessionDistributionController` interface.

- Local, where a *local* session or attribute is stored on the originating server's heap, and thus, only accessible by that server. The `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$LocalController` class provides this behavior. This option is not recommended for production purposes, but it can be useful for testing the difference in scalable performance between local-only and fully distributed implementations.
- Hybrid, which is similar to distributed in that all sessions and serializable attributes are managed in a distributed manner. However, unlike distributed, session attributes that do not implement the `Serializable` interface will be kept local. The `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$HybridController` class provides this behavior.

At any point during the life of a session, the session or attributes for that session can change from local or distributed. However, when a session or attribute is distributed it cannot change back to local.

You can use the Session Distribution Controller in any of the following ways:

- You can allow new sessions to remain local until you add an attribute (for example, when you add the first item to an online shopping cart); the idea is that a session must be fault-tolerant only when it contains valuable data.
- Some Web frameworks use session attributes to store the UI rendering state. Often, this data cannot be distributed because it is not serializable. Using the Session Distribution Controller, these attributes can be kept local while allowing the rest of the session attributes to be distributed.
- The Session Distribution Controller can assist in the conversion from nondistributed to distributed systems, especially when the cost of distributing all sessions and all attributes is a consideration.

Implementing a Session Distribution Controller

Example 5–4 illustrates a sample implementation of the `HttpSessionCollection.SessionDistributionController` interface. In the sample, sessions are tested to see if they have a shopping cart attached (only these sessions will be distributed). Next, the session is tested whether it contains a certain attribute. If the attribute is found, then it is not distributed.

Example 5–4 Sample Session Distribution Controller Implementation

```
import com.tangosol.coherence.servlet.HttpSessionCollection;
import com.tangosol.coherence.servlet.HttpSessionModel;

/**
* Sample implementation of SessionDistributionController
*/
public class CustomSessionDistributionController
        implements HttpSessionCollection.SessionDistributionController
    {
    public void init(HttpSessionCollection collection)
        {
        }
/**
* Only distribute sessions that have a shopping cart.
*
* @param model Coherence representation of the HTTP session
*
* @return true if the session should be distributed
*/
public boolean isSessionDistributed(HttpSessionModel model)
    {
    return model.getAttribute("shopping-cart") != null;
    }

/**
* If a session is "distributed", then distribute all attributes with the
* exception of the "ui-rendering" attribute.
*
* @param model Coherence representation of the HTTP session
* @param sName name of the attribute to check
*
* @return true if the attribute should be distributed
*/
public boolean isSessionAttributeDistributed(HttpSessionModel model,
        String sName)
    {
    return !"ui-rendering".equals(sName);
    }
}
```

Registering a Session Distribution Controller Implementation

After you have written your `SessionDistributionController` implementation, you can register it with your application by using the `coherence-distributioncontroller-class` context parameter. See Appendix A, "Coherence*Web Context Parameters" for more information about this parameter.

Detecting Changed Attribute Values

By default, Coherence*Web tracks if attributes retrieved from the session have changed during the course of processing a request. This is done by caching the initial serialized binary form of the attribute when it is retrieved from the session. At the end of processing a request, Coherence*Web will compare the current binary value of the attribute with the "old" version of the binary. If the values do not match, then the current value is written to the cache.

If you know that your application does not mutate session attributes without doing a corresponding `set`, then you should set the `coherence-enable-suspect-attributes` context parameter to `false`. This will improve memory use and near-cache optimization.

Saving Non-Serializable Attributes Locally

By default, Coherence*Web attempts to serialize all session attributes. If you are working with any session attributes that are not serializable, you can store them locally by setting the `coherence-preserve-attributes` parameter to `true`. This parameter requires you to use a load balancer to retrieve non-serializable attributes for a session.

Note that if the client (application server) fails, then the attributes will be lost. Your application must be able to recover from this.

The default for this parameter is `false`. If you are using Coherence*Web on GlassFish, then this value will be set to `true` because the GlassFish Server requires local sessions to be available.

See Appendix A, "Coherence*Web Context Parameters" for more information about the `coherence-preserve-attributes` parameter.

Securing Coherence*Web Deployments

To prevent unauthorized Coherence TCMP cluster members from accessing HTTP session cache servers, Coherence provides a Secure Socket Layer (SSL) implementation. This implementation can be used to secure TCMP communication between cluster nodes and TCP communication between Coherence*Extend clients and proxies. Coherence allows you to use the Transport Layer Security (TLS) 1.0 protocol which is the next version of the SSL 3.0 protocol; however, the term SSL is used since it is the more widely recognized term.

This section provides only an overview of using SSL in a Coherence environment. For more information and sample configurations, see "Using SSL to Secure Communication" in *Oracle Fusion Middleware Securing Oracle Coherence*.

Using SSL to Secure TCMP Communications

A Coherence cluster can be configured to use SSL with TCMP. Coherence allows you to use both one-way and two-way authentication. Two-Way authentication is typically used more often than one-way authentication, which has fewer use cases in a cluster environment. In addition, it is important to realize that TCMP is a peer-to-peer protocol that generally runs in trusted environments where many cluster nodes are expected to remain connected with each other. The implications of SSL on administration and performance should be carefully considered.

In this configuration, you can use the pre-defined, out-of-the-box SSL socket provider that allows for two-way communication SSL connections based on peer trust, or you can define your own SSL socket provider.

Using SSL to Secure Extend Client Communication

Communication between extend clients and extend proxies can be secured using SSL. SSL requires configuration on both the client side as well as the cluster side. On the cluster side, you configure SSL in the cluster-side cache configuration file by defining a SSL socket provider for a proxy service. You can define the SSL socket provider either for all proxy services or for individual proxy services.

On the client side, you configure SSL in the client-side cache configuration file by defining a SSL socket provider for a remote cache scheme and, if required, for a remote invocation scheme. Like the cluster side, you can define the SSL socket provider either for all remote services or for individual remote services.

Customizing the Name of the Session Cache Configuration File

By default, Coherence*Web uses the information in the `default-session-cache-config.xml` file to configure the session caches in Coherence*Web. You can direct Coherence*Web to use a different file by specifying the `coherence-cache-configuration-path` context parameter in the `web.xml` file, for example:

```
...
<context-param>
   <param-name>coherence-cache-configuration-path</param-name>
   <param-value>my-default-session-cache-config-name.xml</param-value>
</context-param>
...
```

Configuring Logging for Coherence*Web

Coherence*Web uses the logging framework provided by Coherence. Coherence has its own logging framework and also supports the use of log4j, slf4j, and Java logging to provide a common logging environment for an application. Logging in Coherence occurs on a dedicated and low-priority thread to reduce the impact of logging on the critical portions of the system. Logging is pre-configured and the default settings should be changed as required. For more information, see "Configuring Logging" in *Oracle Fusion Middleware Developing Applications with Oracle Coherence*.

The Coherence*Web logging level can also be set using the context parameter/system property `coherence-session-logger-level`. This is an alternative way to set the logging level for Coherence*Web (as opposed to using JDK logging). See Appendix A, "Coherence*Web Context Parameters" for more information on this parameter.

WARNING: Applications that use the JDK logging framework can configure Coherence to use JDK logging as well. Note, however, that setting the log level to FINEST can expose session IDs in the log file.

Getting Concurrent Access to the Same Session Instance

A cache delegator class is a class that is responsible for manipulating (getting, putting, or deleting) any data in the distributed cache. Use the `<coherence-cache-delegator-class>` context parameter in the `web.xml` file to specify the name of the class responsible for the data manipulation.

One of the possible values for the context parameter is the `com.tangosol.coherence.servlet.LocalSessionCacheDelegator` class. This class indicates that the local cache should be used for storing and retrieving the session instance before attempting to use the distributed cache. This delegator is useful for applications that require concurrent access to the same session instance.

Note: This feature must be enabled when working with PeopleSoft applications.

To enable the `LocalSessionCacheDelegator` cache delegator, the following items must be configured in `web.xml`:

- The `coherence-cache-delegator-class` context parameter with the value set to `com.tangosol.coherence.servlet.LocalSessionCacheDelegator`.
- The `coherence-preserve-attributes` context parameter set to `true` to allow nonserializable objects to be stored in the session object.
- The `coherence-distributioncontroller-class` context parameter with the value set to `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$HybridCont`

roller. This value forces all sessions and serializable attributes to be managed in a distributed manner. All session attributes that do not implement the Serializable interface will be kept local. Note that the use of this context parameter also requires coherence-sticky-sessions optimization to be enabled.

Example 5–5 illustrates a sample configuration for the cache delegator in the web.xml file.

Example 5–5 Configuring Cache Delegator in the web.xml File

```
...
 <context-param>
    <param-name>coherence-cache-delegator-class</param-name>
    <param-value>com.tangosol.coherence.servlet.LocalSessionCacheDelegator
</param-value>
 </context-param>
 <context-param>
    <param-name>coherence-preserve-attributes</param-name>
    <param-value>true</param-value>
 </context-param>
 <context-param>
    <param-name>coherence-distributioncontroller-class</param-name>

<param-value>com.tangosol.coherence.servlet.AbstractHttpSessionCollection$HybridCo
ntroller</param-value>
 </context-param>
...
```

Also, when using LocalSessionCacheDelegator as the cache delegator, you should not configure a near cache in the session-cache-config.xml file. This is because local session instances are used. Appendix D, "Session Cache Confguration File Without a Near Cache" illustrates a sample session-cache-config.xml file that omits a near cache configuration.

6

Monitoring Applications

This chapter describes how to use the provided JMX MBeans to monitor the health and performance of Coherence*Web on your system.It also describes how to run the *Reporter*—a JMX-based reporting utility that provides several preconfigured reports that help administrators and developers manage capacity and troubleshoot problems.

This chapter contains the following sections:

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 "Using JMX to Manage Coherence" in *Oracle Fusion Middleware Managing Oracle Coherence*.

Managing and Monitoring Applications with JMX

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

During startup, each Coherence*Web Web application registers a single instance of the `HttpSessionManager` class. You can use a monitoring tool, such as JConsole, to view the values of the MBean attributes. The MBean is unregistered when the Web application shuts down.

Table 6–1 describes the object name that the MBean uses for registration.

Table 6–1 Object Name for HttpSessionManagerMBean

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

Table 6–2 describes the information that `HttpSessionManager` provides. All of the names represent attributes, except `resetStatistics`, which is an operation.

Several of the MBean attributes use the following prefixes:

- `LocalSession`, which 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`, which indicates a session attribute that is not distributed to all members of the cluster.
- `Overflow`, a cache which stores the large session attributes when the Split Session model is used.

Table 6–2 Information Returned by the HttpSessionManager

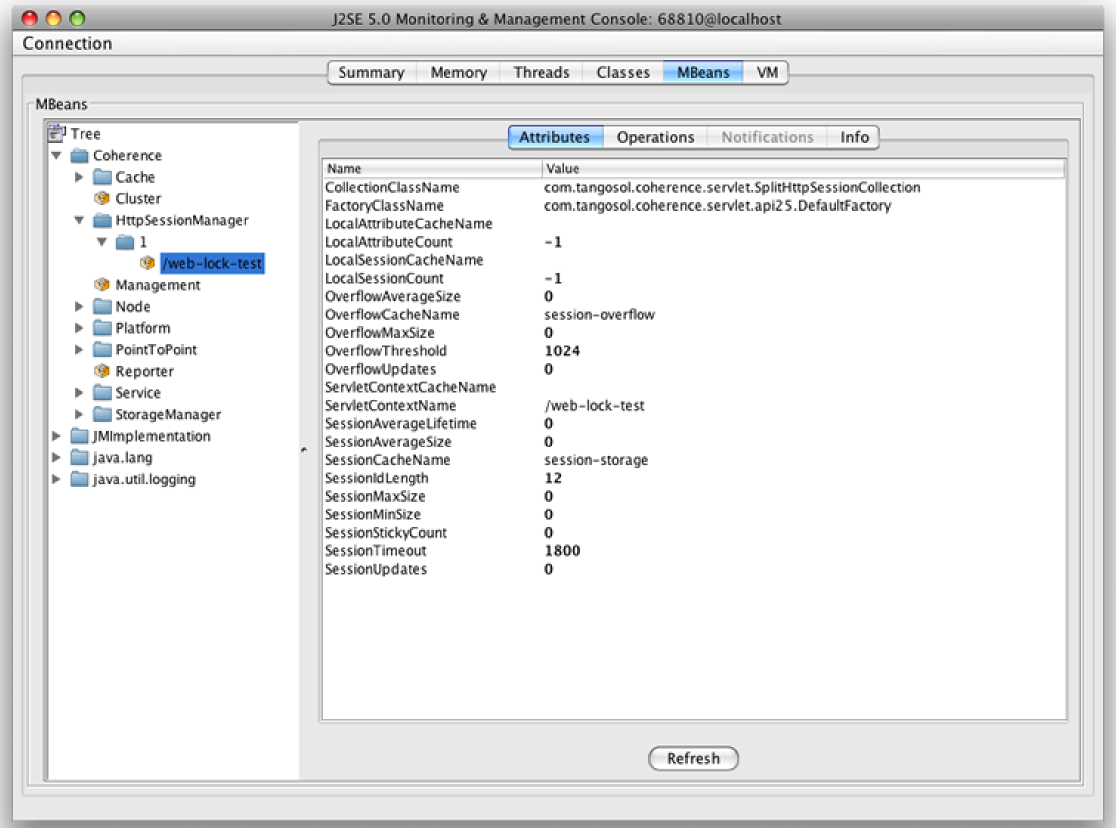
Attribute Name	Data Type	Description
`AverageReapDuration`	long	The average reap duration (the time it takes to complete a reap cycle) in milliseconds, since the statistic was reset. See "Getting Session Reaper Performance Statistics" on page 7-4.
`CollectionClassName`	String	The fully qualified class 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 affected by how the sessions communicate between the clients and the servers.
`FactoryClassName`	String	The fully-qualified class name of the `Factory` implementation being used. The `SessionHelper.Factory` class is used by the `SessionHelper` class to obtain objects that implement various important parts of the servlet specification. The `Factory` implementation can be placed in front of the application instead of the application server's own objects. This changes the apparent implementation of the application server itself (for example, adding clustering.)
`LastReapDuration`	long	The amount of time, in milliseconds, it took for the last reap cycle to finish. See "Getting Session Reaper Performance Statistics" on page 7-4.
`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 nondistributed sessions. If the attribute displays a null value, then local session storage is disabled.
`LocalSessionCount`	Integer	The number of nondistributed sessions stored in the local session cache. If the attribute displays a -1 value, then local session storage is disabled.
`MaxReapedSessions`	long	The maximum number of sessions reaped in a reap cycle since the statistic was reset. See "Getting Session Reaper Performance Statistics" on page 7-4.
`NextReapCycle`	java.lang. Date	The time, expressed as a `java.lang.Date` data type, for the next reap cycle. See "Getting Session Reaper Performance Statistics" on page 7-4.
`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` model is not in use.

Table 6–2 (Cont.) Information Returned by the HttpSessionManager

Attribute Name	Data Type	Description
`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. A null value is displayed if a `SplitHttpSessionCollection` model 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 a -1 value if a `SplitHttpSessionCollection` model is not in use.
`OverflowThreshold`	Integer	The minimum length (in bytes) that the serialized form of an attribute value must be stored in the separate overflow cache that is reserved for large attributes. The attribute displays a -1 value if a `SplitHttpSessionCollection` model 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 a -1 value if a `SplitHttpSessionCollection` model is not in use.
`ReapedSessions`	long	The number of sessions reaped during the last cycle. See "Getting Session Reaper Performance Statistics" on page 7-4.
`ReapedSessionsTotal`	long	The number of expired sessions that have been reaped since the statistic was reset. See "Getting Session Reaper Performance Statistics" on page 7-4.
`ServletContextCacheName`	String	The name of the clustered cache that stores `javax.servlet.ServletContext` attributes. The attribute displays `null` if `ServletContext` is not clustered.
`ServletContextName`	String	The name of the Web application `ServletContext`.
`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.
`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 belong 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.
`resetStatistics` (operation)	void	Reset the session management statistics.

Figure 6–1 illustrates the attributes of the `HttpSessionManager` MBean displayed in the JConsole monitoring tool.

Figure 6–1 HttpSessionManager Displayed in the JConsole Monitoring Tool



Name	Value
CollectionClassName	com.tangosol.coherence.servlet.SplitHttpSessionCollection
FactoryClassName	com.tangosol.coherence.servlet.api25.DefaultFactory
LocalAttributeCacheName	
LocalAttributeCount	-1
LocalSessionCacheName	
LocalSessionCount	-1
OverflowAverageSize	0
OverflowCacheName	session-overflow
OverflowMaxSize	0
OverflowThreshold	1024
OverflowUpdates	0
ServletContextCacheName	
ServletContextName	/web-lock-test
SessionAverageLifetime	0
SessionAverageSize	0
SessionCacheName	session-storage
SessionIdLength	12
SessionMaxSize	0
SessionMinSize	0
SessionStickyCount	0
SessionTimeout	1800
SessionUpdates	0

Managing and Monitoring Applications on WebLogic Server

For WebLogic Server, management attributes and operations for Web applications that use Coherence*Web for HTTP session management are visible through the `WebLogicHttpSessionManagerMBean` MBean (`com.tangosol.coherence.servlet.management.WebLogicHttpSessionManagerMBean`) .

Table 6–3 describes the object name that the MBean uses for registration.

Table 6–3 Object Name for WebLogicHttpSessionManagerMBean

Managed Bean	Object Name
WebLogicHttpSessionManager	`type=WebLogicHttpSessionManager, nodeId=cluster node id, appId=web application id`

The `WebLogicHttpSessionManager` class extends the `HttpSessionManager` class. In addition to the information described in Table 6–2, the `WebLogicHttpSessionManager` class also returns the information listed in Table 6–4. Enterprise Manager uses this information to correlate the Coherence*Web instances to the server.

Table 6–4 Information Returned by the WebLogicHttpSessionManager MBean

Attribute Name	Data Type	Description
`ApplicationId`	String	The WebLogic Web application ID.
`ApplicationName`	String	The name of this Web application.
`ApplicationVersion`	String	The version of this Web application.
`DomainName`	String	The WebLogic domain name on which the application is deployed.
`IsEar`	Boolean	Displays `true` if this Web application is a module of an EAR file.
`IsListenAddressEnabled`	Boolean	Displays `true` if a HTTP port is available on this server.
`IsSSLListenPortEnabled`	Boolean	Displays `true` if a HTTPS port is available on this server.
`ListenAddress`	String	The address on which the server is listening.
`ListenPort`	Integer	The port on which this server listens for HTTP requests.
`ServerName`	String	The WebLogic Server name on which the application is deployed.
`SSLListenPort`	Integer	The port on which this server is listening for HTTPS requests.

Running Performance Reports

Note: You can find a detailed discussion of the Reporter, including configuring the Reporter, running preconfigured reports, and creating custom reports, in the chapters under "Using JMX Reporting" in *Oracle Fusion Middleware Managing Oracle Coherence*.

Coherence includes a JMX-based reporting utility known as the *Reporter*. The Reporter provides several preconfigured reports that help administrators and developers manage capacity and troubleshoot problems. These reports are specially tuned for Coherence*Web:

- Web Session Storage Report, which records statistics about the activity between the cluster and the cache where the cluster's session objects and data are stored.
- Web Session Overflow Report, which records statistics about the activity between the cluster and the cache where session objects and data are allowed to overflow from the Web session storage cache.
- Web Report, which records information about Coherence*Web activity for the cluster.
- WebLogic Web Report, which is intended for WebLogic Server environments. This report provides the same information as Web Report, but includes the name of the WebLogic Server and the WebLogic Server domain.
- Web Service Report, which records information about the service running the Coherence*Web application.

The Coherence*Web reports should be run as part of a batch report. They are defined in both the `report-web-group.xml` and the comprehensive `report-all.xml` batch reports. You can also include them in a custom batch report. The Coherence*Web reports are not defined in the default report group batch file, `report-group.xml`.

The Reporter runs the `report-group.xml` batch report by default. Use the `tangosol.coherence.management.report.configuration` system property to run `report-web-group.xml`, `report-all.xml`, or a custom batch report instead. Example 6–1 illustrates a command line where the property is used to change the report group batch file that is run to `report-web-group.xml`.

Example 6–1 Specifying a Report Group on the Command Line

```
java -Dcom.sun.management.jmxremote
-Dtangosol.coherence.management=all
-Dtangosol.coherence.management.remote=true
-Dtangosol.coherence.management.report.autostart=false
-Dtangosol.coherence.management.report.distributed=false
-Dtangosol.coherence.management.report.configuration=reports/report-web-group.xml
-jar coherence.jar
```

The `report-web-group.xml`, `report-all.xml`, and `report-group.xml` report group batch files, can be found in the `reports` folder in the `coherence.jar` file.

Web Session Storage Report

The Web Session Storage report records statistics on the activity between the cluster and the cache where session objects and data are stored. The statistics include information about the number of put, get, and prune operations performed on the session storage cache, and the amount of time spent on these operations.

The report is a tab-delimited file that is prefixed with the date in `YYYYMMDDHH` format and appended with `-session-storage.txt`. For example `2010013113-session-storage.txt` would be created on January 31, 2010 1:00 pm. Table 6–5 describes the contents of the Web Session Storage report.

Table 6–5 Contents of the Web Session Storage Report

Column Title	Data Type	Description
`Batch Counter`	long	A sequential counter to help integrate information between related files. This value resets when the reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
`Cache Name`	String	Always `session-storage`. It is used to maintain consistency with the Cache Utilization report.
`Evictions`	long	The total number of sessions that have been evicted for the cache across the cluster since the last time the report was created.
`Report Time`	Date	The system time when the report was created.
`Tier`	String	The value can be either `front` or `back`. Describes whether the cache resides in the front tier (local cache) or back tier (remote cache).
`TotalFailures`	long	The total number of session storage write failures for the cache across the cluster since the last time the report was created.
`TotalGets`	long	The total number of session get operations across the cluster since the last time the report was created.
`TotalGetsMillis`	long	The total number of milliseconds spent for each `get()` invocation (`GetsMillis`) to get the sessions across the cluster since the last time the report was created.
`TotalHits`	long	The total number of session hits across the cluster since the last time the report was created.

Table 6–5 (Cont.) Contents of the Web Session Storage Report

Column Title	Data Type	Description
`TotalHitsMillis`	long	The total number of milliseconds spent for each `get()` invocation that is a hit (`HitsMillis`) for the session storage across the cluster since the last time the report was created.
`TotalMisses`	long	The total number of sessions get operations that returned misses for the cache across the cluster since the last time the report was created.
`TotalMissesMillis`	long	The total number of milliseconds spent for each `get()` invocation that is a miss (`MissesMillis`) for the session storage across the cluster since the last time the report was created.
`TotalPrunes`	long	The total number of times the session storage cache has been pruned across the cluster since the last time the report was created.
`TotalPrunesMillis`	long	The total number of milliseconds spent for the prune operation (`PrunesMillis`) to prune the session storage cache across the cluster since the last time the report was created.
`TotalPuts`	long	The total number of session updates (put operations) across the cluster since the last time the report was created.
`TotalPutsMillis`	long	The total number of milliseconds spent for each `put()` invocation (`PutsMillis`) to update sessions across the cluster since the last time the report was created.
`TotalQueue`	long	The sum of the queue links for the session storage cache across the cluster.
`TotalWrites`	long	The total number of sessions written to an external cache storage for the cache across the cluster since the last time the report was created.
`TotalWritesMillis`	long	The total number of milliseconds spent for each write operation (`WritesMillis`) to update an external cache storage across the cluster since the last time the report was created.

Web Session Overflow Report

The Web Session Overflow report records statistics on the activity between the cluster and the cache where the overflow of session objects and data is stored. The statistics include information about the number of put, get, and prune operations performed on the session overflow cache, and the amount of time spent on these operations.

The report is a tab-delimited file that is prefixed with the date in `YYYYMMDDHH` format and appended with `-cache-session-overflow.txt`. For example `2010013113-cache-session-storage.txt` would be created on January 31, 2010 1:00 pm. Table 6–6 describes the contents of the Web Session Overflow report.

Table 6–6 Contents of the Web Session Overflow Report

Column Title	Data Type	Description
`Batch Counter`	long	A sequential counter to help integrate information between related files. This value does reset when the Reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
`Cache Name`	String	Always `session-overflow`. It is used to maintain consistency with the Cache Utilization report.
`Evictions`	long	The total number of session overflows that have been evicted for the cache across the cluster since the last time the report was created.
`Report Time`	Date	The system time when the report executed.
`Tier`	String	The value can be either `front` or `back`. Describes whether the cache resides in the front-tier (local cache) or back tier (remote cache).
`TotalFailures`	long	The total number of session overflows storage write failures for the cache across the cluster since the last time the report was created.
`TotalGets`	long	The total number of session overflows get operations across the cluster since the last time the report was created.
`TotalGetsMillis`	long	The total number of milliseconds spent for each `get()` invocation (`GetsMillis`) to get the session overflows across the cluster since the last time the report was created.
`TotalHits`	long	The total number of session overflow hits across the cluster since the last time the report was created.
`TotalHitsMillis`	long	The total number of milliseconds spent for each `get()` invocation that is a hit (`HitsMillis`) for the session overflow across the cluster since the last time the report was created.
`TotalMisses`	long	The total number of session overflow get operations that returned misses for the cache across the cluster since the last time the report was created.
`TotalMissesMillis`	long	The total number of milliseconds spent for each `get()` invocation that is a miss (`MissesMillis`) for the session overflow across the cluster since the last time the report was created.
`TotalPrunes`	long	The total number of times the session overflow cache has been pruned across the cluster since the last time the report was created.
`TotalPrunesMillis`	long	The total number of milliseconds spent for the prune operations (`PrunesMillis`) to prune the session overflow cache across the cluster since the last time the report was created.
`TotalPuts`	long	The total number of session overflows (put operations) across the cluster since the last time the report was created.
`TotalPutsMillis`	long	The total number of milliseconds spent per `put()` invocation (`PutsMillis`) to update session overflows across the cluster since the last time the report was created.
`TotalQueue`	long	The sum of the queue link size for the session overflow cache across the cluster.

Table 6–6 (Cont.) Contents of the Web Session Overflow Report

Column Title	Data Type	Description
`TotalWrites`	long	The total number of session overflows written to an external cache storage for the cache across the cluster since the last time the report was created.
`TotalWritesMillis`	long	The total number of milliseconds spent for each write operation (`WritesMillis`) to update an external session overflow storage across the cluster since the last time the report was created.

Web Report

The Web Report (`report-web.xml`) provides information about Coherence*Web activity for the cluster. The report is a tab-delimited file that is prefixed with the date and hour in `YYYYMMDDHH` format and appended with `-web.txt`. For example `2009013102-web.txt` would be created on January 1, 2009 at 2:00 am. Table 6–7 describes the contents of the Web Report.

Table 6–7 Contents of the Web Report

Column	Data Type	Description
`Application`	String	The application name.
`Batch Counter`	long	A sequential counter to help integrate information between related files. This value does reset when the Reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
`Current Overflow Updates`	long	The number of overflow updates since the last time the report was created.
`Current Session Updates`	long	The number of session updates since the last time the report was created.
`LocalAttributeCount`	long	The attribute count on the node.
`LocalSessionCount`	long	The session count on the node.
`Node Id`	integer	The node identifier.
`OverflowAvgSize`	float	The average size for attribute overflows.
`OverflowMaxSize`	long	The maximum size for an attribute overflow.
`OverflowUpdates`	long	The total number of attribute overflow updates since the last time statistics were reset.
`Report Time`	Date	The system time when the report was created.
`SessionAverageLifetime`	float	The average number of seconds a session is active.
`SessionAverageSize`	float	The average size for a session.
`SessionMaxSize`	long	The maximum size for a session.
`SessionMinSize`	long	The minimum size for a session.
`SessionStickyCount`	long	The number of sticky sessions on the node.
`SessionUpdateCount`	long	The number of session updates since the last time statistics were reset.

WebLogic Web Report

The Weblogic Web Report (`report-web-weblogic.xml`) provides information on Coherence*Web activity when it is being used in WebLogic Server environments. This report provides the same information as provided by the Web Report (see "Web Report", above), with additional columns for the WebLogic Server name and domain name. The report is a tab-delimited file that is prefixed with the date and hour in `YYYYMMDDHH` format and appended with `-web-weblogic.txt`. For example `2009013102-web-weblogic.txt` would be created on January 1, 2009 at 2:00 am.

Table 6–8 Contents of the WebLogic Web Report

Column	Data Type	Description
`Application`	String	The application name.
`Batch Counter`	long	A sequential counter to help integrate information between related files. This value does reset when the Reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
`Current Overflow Updates`	long	The number of overflow updates since the last time the report was created.
`Current Session Updates`	long	The number of session updates since the last time the report was created.
`DomainName`	String	The name of the WebLogic Server domain in which Coherence*Web is running.
`LocalAttributeCount`	long	The attribute count on the node.
`LocalSessionCount`	long	The session count on the node.
`Node Id`	integer	The node identifier.
`OverflowAvgSize`	float	The average size for attribute overflows.
`OverflowMaxSize`	long	The maximum size for an attribute overflow.
`OverflowUpdates`	long	The total number of attribute overflow updates since the last time statistics were reset.
`Report Time`	Date	The system time when the report was created.
`ServerName`	String	The name of the WebLogic Server on which Coherence*Web is running.
`SessionAverageLifetime`	float	The average number of seconds a session is active.
`SessionAverageSize`	float	The average size for a session.
`SessionMaxSize`	long	The maximum size for a session.
`SessionMinSize`	long	The minimum size for a session.
`SessionStickyCount`	long	The number of sticky sessions on the node.
`SessionUpdateCount`	long	The number of session updates since the last time statistics were reset.

Web Service Report

The Web Service report provides information about the service running the Coherence*Web application. The report records the requests processed, request failures, and request backlog, tasks processed, task failures, and task backlog. `Request Count` and `Task Count` are useful to determine performance and throughput of the service. `RequestPendingCount` and `Task Backlog` are useful in determining capacity issues or blocked processes. `Task Hung Count`, `Task Timeout Count`, `Thread Abandoned Count`, `Request Timeout Count` are the number of unsuccessful executions that have occurred in the system.

The report is a tab-delimited file that is prefixed with the date and hour in `YYYYMMDDHH` format and appended with `-web-session-service.txt`. For example `2009013102-web-session-service.txt` would be created on January 1, 2009 at 2:00 am. Table 6–9 describes the contents of the Web Service Report.

Table 6–9 Contents of the Web Service Report

Column Title	Data Type	Description
`Batch Counter`	Long	A sequential counter to help integrate information between related files. This value does reset when the Reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
`Node Id`	String	The numeric node identifier.
`Refresh Time`	Date	The system time when the service information was updated from a remote node.
`Request Count`	Long	The number of requests by the Coherence*Web application since the last report was created.
`RequestPendingCount`	Long	The number of pending requests by the Coherence*Web application at the time of the report.
`RequestPendingDuration`	Long	The duration for the pending requests of the Coherence*Web application at the time of the report.
`Request Timeout Count`	Long	The number of request timeouts by the Coherence*Web application since the last report was created.
`Report Time`	Date	The system time when the report executed.
`Service`	String	A static value (`DistributedSessions`) used as the service name if merging the information with the service file.
`Task Backlog`	Long	The task backlog of the Coherence*Web application at the time of the report was created.
`Task Count`	Long	The number of tasks executed by the Coherence*Web application since the last report was created.
`Task Hung Count`	Long	The number of tasks that were hung by the Coherence*Web application since the last report was created.
`Task Timeout Count`	Long	The number of task timeouts by the Coherence*Web application since the last report was created.
`Thread Abandoned Count`	Long	The number of threads abandoned by the Coherence*Web application since the last report was created.

7

Cleaning Up Expired HTTP Sessions

This chapter describes how to configure and use the Session Reaper. The Session Reaper is responsible for destroying any session that is no longer used, which is determined by when that session has timed out.

This chapter contains the following sections:

As part of Coherence*Web Session Management Module, HTTP sessions that have expired are eventually cleaned up by the Session Reaper. The Session Reaper provides a service similar to the JVM Garbage Collection (GC) capability: the Session Reaper is responsible for destroying any session that is no longer used, which is determined by when that session has timed out.

Each HTTP session contains two pieces of information that determine when it has timed out. The first is the `LastAccessedTime` property of the session, which is the time stamp of the most recent activity involving the session. The second is the `MaxInactiveInterval` property of the session, which specifies how long the session is kept active without any activity; a typical value for this property is 30 minutes. The `MaxInactiveInterval` property defaults to the value configured for Coherence*Web, but it can be modified on a session-by-session basis.

Each time that an HTTP request is received by the server, if there is an HTTP session associated with that request, then the `LastAccessedTime` property of the session is automatically updated to the current time. As long as requests continue to arrive related to that session, it is kept active, but when a period of inactivity occurs longer than that specified by the `MaxInactiveInterval` property, then the session expires. Session expiration is passive—occurring only due to the passing of time. The Coherence*Web Session Reaper scans for sessions that have expired, and when it finds expired sessions it destroys them.

Understanding the Session Reaper

The Session Reaper configuration addresses three basic questions:

- On which servers will the Reaper run?
- How frequently will the Reaper run?
- When the Reaper runs, on which servers will it look for expired sessions?

Every application server running Coherence*Web runs the Session Reaper. That means that if Coherence is configured to provide a separate cache tier (made up of cache servers), then the Session Reaper does not run on those cache servers.

By default, the Session Reaper runs concurrently on all of the application servers, so that all of the servers share the workload of identifying and cleaning up expired sessions. The `coherence-reaperdaemon-cluster-coordinated` context parameter causes the cluster to coordinate reaping so that only one server at a time performs the actual reaping; the use of this option is not suggested, and it cannot be used with the Coherence*Web over Coherence*Extend topology.

The `coherence-reaperdaemon-cluster-coordinated` context parameter should not be used if sticky optimization (`coherence-sticky-sessions`) is also enabled. Because only one server at a time performs the reaping, sessions owned by other nodes cannot be reaped. This means that it will take longer for sessions to be reaped as more nodes are added to the cluster. Also, the reaping ownership does not circulate over the nodes in the cluster in a controlled way; one node can be the reaping node for a long time before it is taken over by another node. During this time, only its own sessions are reaped.

The Session Reaper is configured to scan the entire set of sessions over a certain period, called a reaping cycle, which defaults to five minutes. This length of the reaping cycle is specified by the `coherence-reaperdaemon-cycle-seconds` context parameter. This setting indicates to the Session Reaper how aggressively it must work. If the cycle length is configured too short, the Session Reaper uses additional resources without providing additional benefit. If the cycle length is configured too long, then expired sessions will use heap space in the Coherence caches unnecessarily. In most situations, it is far preferable to reduce resource usage than to ensure that sessions are cleaned up quickly after they expire. Consequently, the default cycle of five minutes is a good balance between promptness of cleanup and minimal resource usage.

During the reaping cycle, the Session Reaper scans for expired sessions. In most cases, the Session Reaper takes responsibility for scanning all of the HTTP sessions across the entire cluster, but there is an optimization available for the single tier topology. In the single tier topology, when all of the sessions are being managed by storage-enabled Coherence cluster members that are also running the application server, the session storage is colocated with the application server. Consequently, it is possible for the Session Reaper on each application server to scan only the sessions that are stored locally. This behavior can be enabled by setting the `coherence-reaperdaemon-assume-locality` configuration option to `true`.

Regardless of whether the Session Reaper scans only colocated sessions or all sessions, it does so in a very efficient manner by using these advanced capabilities of the Coherence data grid:

- The Session Reaper delegates the search for expired sessions to the data grid using a custom `ValueExtractor` implementation. This `ValueExtractor` takes advantage of the `BinaryEntry` interface so that it can determine if the session has expired without even deserializing the session. As a result, the selection of expired sessions can be delegated to the data grid just like any other parallel query, and can be executed by storage-enabled Coherence members in a very efficient manner.
- The Session Reaper uses the `com.tangosol.net.partition.PartitionedIterator` class to automatically query on a member-by-member basis, and in a random order that avoids harmonics in large-scale clusters.

Each storage-enabled member can very efficiently scan for any expired sessions, and it has to scan only one time per application server per reaper cycle. The result is a default

Session Reaper configuration that works well for application server clusters with one or multiple servers.

The Session Reaper can invalidate sessions either in parallel or serially. By default, it invalidates sessions in parallel. This ensures that sessions are invalidated in a timely manner. However, if the application server JVM has a high system load due to a large number of concurrent threads then you have the option of invalidating serially. To configure the reaper to invalidate sessions serially, set the `coherence-reaperdaemon-parallel` context parameter to `false`.

Coherence*Web uses a work manager to retrieve threads to execute the parallel reaping. WebLogic Server defines a default work manager, `wm/CoherenceWorkManager`, which it will attempt to use. If no work manager is defined with that name, it will use the default work manager implemented in Coherence.

To ensure that the Session Reaper does not impact the smooth operation of the application server, it breaks up its work into chunks and schedules that work in a manner that spreads the work across the entire reaping cycle. Because the Session Reaper has to know how much work it must schedule, it maintains statistics on the amount of work that it performed in previous cycles, and uses statistical weighting to ensure that statistics from recent reaping cycles count more heavily. There are several reasons why the Session Reaper breaks up the work in this manner:

- If the Session Reaper consumed a large number of CPU cycles simultaneously, it could cause the application to be less responsive to users. By doing a small portion of the work at a time, the application remains responsive.
- One of the key performance enablers for Coherence*Web is the near-caching feature of Coherence; because the sessions that are expired are accessed through that same near cache to clean them, expiring too many sessions too quickly could cause the cache to evict sessions that are being used on that application server, leading to performance loss.

The Session Reaper performs its job efficiently, even with the default configuration by:

- Delegating as much work as possible to the data grid
- Delegating work to only one member at a time
- Enabling the data grid to find expired sessions without deserializing them
- Restricting the usage of CPU cycles
- Avoiding cache-thrashing of the near caches that Coherence*Web relies on for performance

Configuring the Session Reaper

The following are suggestions for tuning the default Session Reaper configuration:

- If the application is deployed with the in-process topology, then set the `coherence-reaperdaemon-assume-locality` configuration option to `true`.
- Because all of the application servers are responsible for scanning for expired sessions, it is reasonable to increase the `coherence-reaperdaemon-cycle-seconds` configuration option if the cluster is larger than 10 application servers. The larger the number of application servers, the longer the cycle can be; for example, with 200 servers, it would be reasonable to set the length of the reaper cycle as high as 30 minutes (that is, setting the `coherence-reaperdaemon-cycle-seconds` configuration option to 1800).

Getting Session Reaper Performance Statistics

The `HttpSessionManagerMBeanWeb` provides several attributes that serve as performance statistics for the Session Reaper. These statistics, described in the following list, include the average time duration for a reap cycle, the number of sessions reaped, and the time until the next reap cycle:

- `AverageReapDuration`, which is the average reap duration (the time it takes to complete a reap cycle), in milliseconds, since the statistic was reset
- `LastReapDuration`, which is the time in milliseconds it took for the last reap cycle to finish
- `MaxReapedSessions`, which is the maximum number of sessions reaped in a reap cycle since the statistic was reset
- `NextReapCycle`, which is the time (as a `java.lang.Date` data type) for the next reap cycle
- `ReapedSessions`, which is the number of sessions reaped during the last cycle
- `ReapedSessionsTotal`, which is the number of expired sessions that have been reaped since the statistic was reset

These attributes are also described in Table 6–2 in the section "Managing and Monitoring Applications with JMX" on page 6-1.

You can access these attributes in a monitoring tool such as JConsole. However, you must set up the Coherence Clustered JMX Framework before you can access them. The configuration and installation instructions for the framework is provided in "Using JMX to Manage Coherence" in *Oracle Fusion Middleware Managing Oracle Coherence*.

Understanding Session Invalidation Exceptions for the Session Reaper

Each Coherence*Web instance has a Session Reaper that will periodically iterate through all of the sessions in the session cache and check for expired sessions. If multiple Web applications are using Coherence*Web, then a reaper from one Web application can invalidate sessions used in a different application. Session listeners registered with the Web application that is reaping expired sessions will be executed.

Session attribute listeners will attempt to retrieve the session attribute values during invalidation. If the session attributes are dependent on classes that exist only in the original Web application, then a *class not found* exception will be thrown and logged in the Session Reaper. These exceptions will not cause any disruption in the Web application or the application server.

Coherence*Web provides a context parameter, `coherence-session-log-invalidation-exceptions`, to control whether these exceptions are logged. The default value, `true`, allows the exceptions to be logged. If you want to suppress the logging of these exceptions, set this context parameter to `false`.

8 Working with JSF and MyFaces Applications

This chapter describes how to configure Coherence*Web for JavaServer Faces (JSF) and MyFaces applications. JSF is a framework that enables you to build user interfaces for Web applications. MyFaces, from the Apache Software Foundation, provides JSF components that extend the JSF specification. MyFaces components are completely compatible with the JSF 1.1 Reference Implementation or any other compatible implementation.

This chapter contains the following sections:

- Configuring for all JSF and MyFaces Web Applications:
- Configuring for Instrumented Applications that use MyFaces
- Configuring for Instrumented Applications that use Mojarra

Configuring for all JSF and MyFaces Web Applications:

JSF and MyFaces attempts to cache the state of the view in the session object. This state data should be serializable by default, but there could be situations where this would not be the case. For example:

- If Coherence*Web reports `IllegalStateException` due to a non-serializable class, and all the attributes placed in the session by your Web-application are serializable, then you must configure JSF/MyFaces to store the state of the view in a hidden field on the rendered page.
- If the Web application puts non-serializable objects in the session object, you must set the `coherence-preserve-attributes` context parameter to `true`.

The JSF parameter `javax.faces.STATE_SAVING_METHOD` identifies where the state of the view is stored between requests. By default, the state is saved in the servlet session. Set the `STATE_SAVING_METHOD` parameter to `client` in the `context-param` stanza of the `web.xml` file, so that JSF stores the state of the entire view in a hidden field on the rendered page. If you do not, then JSF may attempt to cache that state, which is not serializable, in the session object.

Example 8–1 illustrates setting the `STATE_SAVING_METHOD` parameter in the `web.xml` file.

Example 8–1 Setting STATE_SAVING_METHOD in the web.xml File

```
...
<context-param>
    <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
    <param-value>client</param-value>
</context-param>
```

```
...
```

Configuring for Instrumented Applications that use MyFaces

If you are deploying the MyFaces application with the Coherence*Web WebInstaller (that is, an *instrumented* application), then you might have to complete an additional step based on the version of MyFaces.

- If you are using Coherence*Web WebInstaller to deploy a Web-application built with a pre-1.1.*n* version of MyFaces, then nothing more needs to be done.
- If you are using Coherence*Web WebInstaller to deploy a Web-application built with a 1.2.x version of MyFaces, then add the context parameter `org.apache.myfaces.DELEGATE_FACES_SERVLET` to the `web.xml` file. This parameter allows you to specify a custom servlet instead of the default `javax.faces.webapp.FacesServlet`.

  Example 8–2 illustrates setting the `DELEGATE_FACES_SERVLET` context parameter in the `web.xml` file.

Example 8–2 Setting DELEGATE_FACES_SERVLET in the web.xml File

```
...
<context-param>
    <param-name>org.apache.myfaces.DELEGATE_FACES_SERVLET</param-name>
    <param-value>com.tangosol.coherence.servlet.api23.ServletWrapper</param-value>
</context-param>
...
```

Configuring for Instrumented Applications that use Mojarra

If you are using Coherence*Web WebInstaller to deploy a Web application based on the JSF Reference Implementation (Mojarra), then you must declare the `FacesServlet` class in the `servlet` stanza of the `web.xml` file.

Example 8–3 Declaring the Faces Servlet in the web.xml File

```
...
<servlet>
     <servlet-name>Faces Servlet (for loading config)</servlet-name>
     <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
 </servlet>
...
```

A Coherence*Web Context Parameters

This appendix describes the Coherence*Web context parameters. The parameters can be configured in the `web.xml` file or they can also be entered on the command line as system properties. The system properties have the same name as the context parameters, but the dash (`-`) is replaced with a period (`.`).

For example, the context parameter `coherence-enable-sessioncontext` can be declared on the command line by:

```
-Dcoherence.enable.sessioncontext=true
```

If both a system property and the equivalent context parameter are configured, the value from the system property is honored.

Table A–1 describes the context parameters for Coherence*Web.

Table A–1 Context Parameters for Coherence*Web

Parameter Name	Description
`coherence-application-name`	Coherence*Web uses the value of this parameter to determine the name of the application that uses the `ApplicationScopeController` interface to scope attributes. The value for this parameter should be provided in the following format: *application name* + ! + *Web module name* The *application name* is the name of the application that uses the `ApplicationScopeController` interface and *Web module name* is the name of the Web module in which it appears. For example, if you have an EAR file named `test.ear` and a Web-module named `app1` defined in the EAR file, then the default value for the `coherence-application-name` parameter would be `test!app1`. If this parameter is not configured, then Coherence*Web uses the name of the class loader instead. Also, if the parameter is not configured and the `ApplicationScopeController` interface *is* configured, then a warning is logged saying that the application name was not configured. See "Session Attribute Scoping" on page 5-12 for more information.
`coherence-attribute-overflow-threshold`	For the Split Model, described in "Session Models" on page 5-2, this value specifies the minimum length (in bytes) that the serialized form of an attribute value must be for it to be stored in the separate overflow cache that is reserved for large attributes. If unspecified, this parameter defaults to `1024`.
`coherence-cache-configuration-path`	Specifies the name of the file that Coherence*Web should use to obtain session cache information, instead of using the default `default-session-cache-config.xml` file. See "Customizing the Name of the Session Cache Configuration File" on page 5-27.

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

Parameter Name	Description
`coherence-cache-delegator-class`	Specifies a cache delegator class that is responsible for manipulating (getting, putting, or deleting) data in the distributed cache. Valid value is: ■ `com.tangosol.coherence.servlet.LocalSessionCacheDelegator`—This class indicates that the local cache should be used for storing and retrieving the session instance before attempting to use the distributed cache. See "Getting Concurrent Access to the Same Session Instance" on page 5-28 for more information.
`coherence-cluster-owned`	If `true`, Coherence*Web automatically shuts 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 5-16 for more information. If `false`, the Web application is responsible for shutting down the Coherence node (see `com.tangosol.net.CacheFactory.shutdown()` in the Javadoc). 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 5-13 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 Java EE application. If unspecified, this parameter defaults to `false`.
`coherence-configuration-consistency`	If `true`, Coherence*Web runs a configuration check at startup to determine whether all nodes in the Web tier have the same Coherence*Web configuration. If the configuration of a particular node is not consistent, then it will fail to start (which, in turn, prevents the application from starting). If `false`, (there is no checking) and the configurations are not consistent, then the cluster members might exhibit inconsistent behavior in managing the session data. If unspecified, this parameter defaults to `false`.
`coherence-contextless-session-retain-millis`	Specifies the number of milliseconds that a server holds a lock on a session while accessing it 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. Because session access requires session ownership, out of context access to the session object automatically obtains 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 valid range is 10 to 10000 (from `1/100`th of a second up to `10` seconds). If unspecified, this parameter defaults to `200`.

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

Parameter Name	Description
`coherence-distributioncontroller-class`	This value specifies a class name of the `com.tangosol.coherence.servlet.HttpSessionCollection$SessionDistributionController` interface implementation. Valid values include: ■ `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$DistributedController` an implementation of the `SessionDistributionController` interface that forces all sessions (and thus their attributes) to be managed in a distributed manner. This is the default behavior, but by having an implementation that forces this, the raw overhead of using a `HttpSessionController` can be measured. ■ `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$HybridController` an implementation of the `SessionDistributionController` interface that forces all sessions and serializable attributes to be managed in a distributed manner. All session attributes that do not implement the `Serializable` interface will be kept local. ■ `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$LocalController` an implementation of the `SessionDistributionController` interface that forces all sessions (and thus their attributes) to be managed locally. This might not be useful for production purposes, but it can be useful for testing the difference in scalable performance between local-only and fully-distributed implementations.
`coherence-enable-sessioncontext`	When set to `true`, this parameter allows the application to iterate sessions from the session context, thus disobeying the deprecation in the servlet specification. If unspecified, this parameter defaults to `false`.
`coherence-eventlisteners`	This is 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 the `web.xml` file.
`coherence-enable-suspect-attributes`	If set to `true`, Coherence*Web attempts to detect whether the value of any session-related attributes have changed. Attributes that can be changed (determined with a simple check) and that can be accessed by a `get` method are deemed to be suspect. Changeable objects might have been changed by application code and must be re-serialized back into the cache. See "Detecting Changed Attribute Values" on page 5-26 for more information. If unspecified, this parameter defaults to `true`.
`coherence-factory-class`	This is the fully qualified name of the class that implements the `SessionHelper.Factory` factory class. This parameter defaults to `com.tangosol.coherence.servlet.api`*`nn`*`.DefaultFactory` where *nn* is `22`, `23`, `24`, or `25` for Servlet 2.2, 2.3, 2.4, or 2.5 containers respectively.
`coherence-local-session-cachename`	This name overrides the name of the local cache that stores nondistributed sessions when the `coherence-distributioncontroller-class` parameter is specified. If unspecified, this parameter defaults to `local-session-storage`. Appendix C, "Session Cache Configuration File" describes this parameter.
`coherence-local-attribute-cachename`	This name overrides the name of the local cache that stores non-distributed sessions when either the `coherence-sessiondistributioncontroller-class` parameter is specified or the `coherence-preserve-attributes` parameter is `true`. If unspecified, this parameter defaults to `local-attribute-storage`. Appendix C, "Session Cache Configuration File" describes this parameter.

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

Parameter Name	Description
`coherence-preserve-attributes`	This value, if set to `true`, specifies that non-serializable attributes should be preserved as local ones. This parameter requires a load balancer to be present to retrieve non-serializable attributes for a session. These attributes will be lost if the client (application server) fails. The application would need to be able to recover from this. If you are using Coherence*Web on GlassFish, this value will be set to `true` because the GlassFish Server requires local sessions to be available. If unspecified, this parameter defaults to `false`.
`coherence-reaperdaemon-assume-locality`	This setting allows the Session 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 expiration. 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. If cache servers are being used, select the Split Model and run the session overflow storage in a separate distributed cache service that is managed entirely by the cache servers. Leave the session storage cache itself in a distributed cache service that is managed entirely by the application server JVMs so they can take advantage of this assume locality feature. See Chapter 7, "Cleaning Up Expired HTTP Sessions" for more information about the Session Reaper. If unspecified, this parameter defaults to `true`.
`coherence-reaperdaemon-cluster-coordinated`	If `true`, the Session Reaper 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. See Chapter 7, "Cleaning Up Expired HTTP Sessions" for more information about the Session Reaper. This option should not be used if sticky optimization (`coherence-sticky-sessions`) is also enabled. See "Understanding the Session Reaper" on page 7-1 for more information. If unspecified, this parameter defaults to `false`.
`coherence-reaperdaemon-cycle-seconds`	This is the number of seconds that the daemon rests between reaping. For production clusters with long session timeout intervals, this can safely be set higher. For testing, particularly with short session timeout intervals, it can be set much lower. Setting it too low can cause more network traffic and use more processing cycles, and has benefit only if the application requires the sessions to be invalidated quickly when they have expired. See Chapter 7, "Cleaning Up Expired HTTP Sessions" for more information about the Session Reaper. If unspecified, this parameter defaults to `300`.
`coherence-reaperdaemon-parallel`	If set to `true`, the Session Reaper will invalidate expired sessions in parallel. When set to `false`, expired sessions will be invalidated serially. See "Understanding the Session Reaper" on page 7-1. The default is `true`.
`coherence-reaperdaemon-priority`	This is the priority for the Session Reaper daemon. For more information, see Chapter 7, "Cleaning Up Expired HTTP Sessions" and the source for the `java.lang.Thread` class. If unspecified, this parameter defaults to `5`.
`coherence-reaperdaemon-sweep-modulo`	This parameter is deprecated as of Coherence Release 3.5.

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

Parameter Name	Description
`coherence-scopecontroller-class`	This value specifies a class name of the optional `com.tangosol.coherence.servlet.HttpSessionCollection$AttributeScopeController` interface implementation. See "Session Attribute Scoping" on page 5-12 for more information. Valid values include: ■ `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$ApplicationScopeController` ■ `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$GlobalScopeController` The default value for Coherence*Web is `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$ApplicationScopeController` For Coherence*Web WebInstaller, there is no declared default value.
`coherence-servletcontext-clustered`	This value is 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, this parameter defaults to `false`, primarily because the servlet specification indicates that the `ServletContext` attributes are local to a JVM and should not be clustered.
`coherence-servletcontext-cachename`	This specifies the name of the Coherence cache to be used to hold the servlet context data if the servlet context is clustered. If unspecified, this parameter defaults to `servletcontext-storage`. Appendix C, "Session Cache Configuration File" describes this parameter.

`coherence-session-affinity-token`	Configures the session affinity suffix token with a given value. For example, to set the session affinity suffix to `abcd`, add the following code to the Web application's `web.xml` file:
	
	`<context-param>`
	  `<param-name>coherence-session-affinity-token</param-name>`
	  `<param-value>abcd</param-value>`
	`</context-param>`
	
	To strip the session affinity suffix from the token, enter an exclamation point (`!`) as the parameter value. See "Sharing Coherence*Web Sessions with Other Application Servers" on page 2-16 for more information.

`coherence-session-app-locking`	This value, if set to `true`, will prevent two threads in different applications 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, because application locking implies member locking. A value of `false` is incompatible with thread locking. If unspecified, this parameter defaults to `false`. See also `coherence-session-member-locking`, `coherence-session-locking`, and `coherence-session-thread-locking` parameter descriptions in this table and "Session Locking Modes" on page 5-17.
`coherence-session-cachename`	This name overrides the name of the clustered cache that stores the sessions. If unspecified, this parameter defaults to `session-storage`. Appendix C, "Session Cache Configuration File" describes this parameter.
`coherence-session-cookie-domain`	This specifies the domain of the session cookie as defined by *Request for Comments 2109: HTTP State Management Mechanism* (RFC 2109). By default, no domain is set explicitly by the session management implementation. See "Session and Session Attribute Scoping" on page 5-10 for more information.

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

Parameter Name	Description
`coherence-session-cookie-httponly`	Appends the `HttpOnly` attribute to the session cookie. Note that not all browsers support this functionality. This context parameter can be used only with instrumented applications. See "Preventing Cross-Site Scripting Attacks" on page 4-11 for more information.
`coherence-session-cookie-name`	This specifies the name of the session cookie. If unspecified, this parameter defaults to `JSESSIONID`.
`coherence-session-cookie-path`	This specifies the path of the session cookie as defined by RFC 2109. By default, no path is set explicitly by the session management implementation. See "Session and Session Attribute Scoping" on page 5-10 for more information.
`coherence-session-cookie-max-age`	This specifies the maximum age in seconds of the session cookie as defined by RFC 2109. A value of -1 indicates that the cookie will not persist on the client; a positive value gives the maximum age that the cookie will be persistent for the client. Zero is not permitted. If unspecified, this parameter defaults to -1.
`coherence-session-cookie-secure`	If `true`, this value ensures that the session cookie will be sent only from a Web client over a Secure Socket Layer (SSL) connection. If unspecified, the default is `false`.
`coherence-session-cookies-enabled`	If unspecified, this parameter defaults to `true` to enable session cookies.
`coherence-session-deathcert-cachename`	This name overrides the name of the clustered cache that stores the IDs of recently departed sessions. If unspecified, this parameter defaults to `session-death-certificates`. Appendix C, "Session Cache Configuration File" describes this parameter.
`coherence-session-expire-seconds`	This value overrides the session expiration time, and is expressed in seconds. Setting it to -1 causes sessions to never expire. See Chapter 7, "Cleaning Up Expired HTTP Sessions" for more information. If unspecified, this parameter defaults to `1800`.
`coherence-session-get-lock-timeout`	This value configures a timeout for lock acquisition for Coherence*Web. See "Troubleshooting Locking in HTTP Sessions" on page 5-19 for more information.
`coherence-session-id-length`	This is the length, in characters, of generated session IDs. The suggested absolute minimum length is 8. If unspecified, this parameter defaults to `12`.
`coherence-session-lazy-access`	This value enables lazy acquisition of sessions. A session will be acquired only when the servlet or filter attempts to access it. This is relevant only for instrumented Web applications. See "Accessing Sessions with Lazy Acquisition" on page 5-24. If unspecified, this parameter defaults to `false`.
`coherence-session-locking`	If `false`, concurrent modification to sessions, with the last update being saved, will be allowed. If `coherence-session-app-locking`, `coherence-session-member-locking`, or `coherence-session-thread-locking` are set to `true`, then this value is ignored (being logically `true`). See "Optimistic Locking" on page 5-17 and "Last-Write-Wins Locking" on page 5-18. If unspecified, this parameter defaults to `false`. See also `coherence-session-app-locking`, `coherence-session-member-locking`, and `coherence-session-thread-locking` in this table.

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

Parameter Name	Description

`coherence-session-locking-mode`	The value of this context parameter determines the locking mode that will govern concurrent access to HTTP sessions.
	■ `none`—This value allows concurrent access to a session by multiple threads in a single member or multiple members. In this case, the last write is saved. This is the default locking mode. See "Last-Write-Wins Locking" on page 5-18.
	■ `optimistic`—This value allows multiple web container threads in one or more members to access the same session concurrently. See "Optimistic Locking" on page 5-17.
	■ `app`—This value prevent two threads in different applications from processing a request for the same session at the same time. If this parameter is set to `app`, then the value of the `coherence-session-member-locking` parameter will be ignored, because application locking implies member locking. A value of `false` is incompatible with thread locking. See "Application Locking" on page 5-18.
	■ `member`—This value allows multiple web container threads in the same cluster node to access and modify the same session concurrently, but prohibits concurrent access by threads in different members. See "Member Locking" on page 5-18.
	■ `thread`—This value prevents 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 is ignored, because thread locking implies member locking. See "Thread Locking" on page 5-18.
	For example, to set the `coherence-session-locking-mode` context parameter to application locking in `web.xml`:
	`<context-param>`
	   `<param-name>coherence-session-locking-mode</param-name>`
	   `<param-value>app</param-value>`
	`</context-param>`

`coherence-session-log-invalidation-exceptions`	During session invalidation, many *class not found* exceptions might be thrown and logged in the session reaper. If this context parameter is set to `false`, then the exceptions will be suppressed. If set to `true`, then the exceptions will be logged. If unspecified, this parameter defaults to `true`. For more information on session invalidation and the cause of the *class not found* exceptions that might occur during operation of the session reaper, see "Understanding Session Invalidation Exceptions for the Session Reaper" on page 7-4.
`coherence-session-log-threads-holding-lock`	If `true`, this value specifies if a diagnostic invocation service is executed when a member cannot acquire the cluster lock for a session. The invocation service will cause the member that has ownership of the session to log the stack trace of the threads that are currently holding the lock. The `coherence-session-log-threads-holding-lock` context parameter is available only when the `coherence-sticky-sessions` context parameter is set to `true`. If unspecified, this parameter defaults to `true`. See "Troubleshooting Locking in HTTP Sessions" on page 5-19 for more information.

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

Parameter Name	Description
`coherence-session-logger-level`	An alternative way to set the logging level for Coherence*Web (as opposed to JDK logging). The valid values for this parameter are the same as for JDK logging: `SEVERE`, `WARNING`, `INFO`, `CONFIG`, `FINE`, `FINER` (default), and `FINEST`. See "Configuring Logging for Coherence*Web" on page 5-28 for more information. See also the Javadoc for `java.util.logging`: `http://docs.oracle.com/javase/1.4.2/docs/api/java/util/logging/Level.html`
`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, this parameter defaults to `session-management`. Appendix C, "Session Cache Configuration File" describes this parameter.
`coherence-session-member-locking`	If `true`, this value prevents two threads in different members from processing a request for the same session at the same time. If unspecified, this parameter defaults to `false`. See also `coherence-session-thread-locking`, `coherence-session-locking`, and `coherence-session-app-locking` in this table.
`coherence.session.optimizeModifiedSessions`	This JVM system property, if set to `true`, enables near cache optimizations which can improve performance with applications that use Last-Write-Wins locking. If unspecified, this value defaults to `false`. This parameter can be set only on the command line as a system property.
`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, this parameter defaults to `session-overflow`. Appendix C, "Session Cache Configuration File" describes this parameter.
`coherence-session-strict-spec`	If `false`, then the implementation will not be required to adhere to the servlet specification. The implementation will ignore certain types of exceptions and the application will not terminate. Setting, getting, and removing attributes, or invalidating sessions will not generate any callbacks to session listeners. Any `ClassNotFound` exceptions will not be propagated back to the caller if an attribute cannot be deserialized because the class does not exist in the invoking application. If `true`, then the implementation strictly adheres to the servlet specification. `ClassNotFound` exceptions must be handled by the application, and session listener events will be sent, even if retrieving the attribute value fails. If unspecified, this parameter defaults to `true`.
`coherence-session-thread-locking`	If `true`, this value prevents 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 is ignored, because thread locking implies member locking. If unspecified, this parameter defaults to `false`. See also `coherence-session-app-locking`, `coherence-session-locking`, and `coherence-session-member-locking` parameter descriptions in this table and "Session Locking Modes" on page 5-17.
`coherence-session-urldecode-bycontainer`	If `true`, this value 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, this parameter defaults to `true`.

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

Parameter Name	Description
`coherence-session-urlencode-bycontainer`	If `true`, this value uses the container's encoding of the URL session ID. Setting this to `true` could conflict with the setting for `coherence-session-urlencode-name` if it has been specified. If unspecified, this parameter defaults to `false`.
`coherence-session-urlencode-enabled`	If `true`, this value enables URL encoding of session IDs. If unspecified, this parameter defaults to `true`.
`coherence-session-urlencode-name`	This is the parameter name to encode the session ID into the URL. On some containers, this value cannot be overridden. If unspecified, this parameter defaults to `jsessionid`.
`coherence-session-weblogic-compatibility-mode`	If `true`, a single session ID (with the cookie path set to "`/`") will map to a unique Coherence*Web session instance in each Web application. If `false`, then the standard behavior will apply: that is, a single session ID will map to a single session instance. All other session persistence mechanisms in WebLogic Server use a single session ID in each Web application to refer to different session instances. If unspecified, this parameter defaults to `true`. An exception is when the application is configured to use the global scope controller. In this case, the default is `false`. See "Scoping the Session Cookie Path" on page 2-15.
`coherence-sessioncollection-class`	This is the fully-qualified class name of the `HttpSessionCollection` implementation to use. Possible values include: ■ `com.tangosol.coherence.servlet.MonolithicHttpSessionCollection` ■ `com.tangosol.coherence.servlet.SplitHttpSessionCollection` (default) ■ `com.tangosol.coherence.servlet.TraditionalHttpSessionCollection` A value must be specified for this parameter. See "Configuring a Session Model" on page 5-7.
`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 shutdown; any positive value indicates asynchronous shutdown. If unspecified, this parameter defaults to 0, because some servers are not capable of asynchronous shutdown.
`coherence-sticky-sessions`	If `true`, this value specifies whether sticky session optimizations will be used. This should be enabled only if a sticky load balancer is being used. This feature requires member, application or thread locking to be enabled. See "Enabling Sticky Session Optimizations" on page 5-19. See also `coherence-session-thread-locking`, `coherence-session-member-locking`, and `coherence-session-app-locking` in this table. If unspecified, this parameter defaults to `false`.
`coherence-web-sessions-enabled`	Enables Coherence*Web sessions in WebLogic Portal applications.

B Capacity Planning

This appendix helps you estimate the number of cache servers that an application will need. These equations will help you only to arrive at a *reasonable* estimate; they do not account for the effects of cache indexes, nonapplication objects that might reside on the cache server heap, failover headroom, and so on.

To find the number of cache servers that you will need, you must first calculate the application's heap requirements and the cache server's available tenured generation.

1. Calculate your application's total heap requirements.

   When trying to determine the number of cache servers that you will need for your application, a good starting point is to determine your application's total heap requirements. The total heap requirement can be calculated as the number of sessions that you will run, multiplied by the average number of cached objects per session, multiplied by average number of bytes per cached object. Because you typically make one backup copy per cache entry, multiply the total by 2. Written as an equation, this becomes:

   ```
   Total_Heap_Requirement = 2 * (Number_of_Sessions) * (Average_Number_of_Cached_
   Objects per Session) * (Average_Number_of_Bytes per Cached_Object)
   ```

   The units of measure for `Total_Heap_Requirement` are bytes. The `Average_ Number_of_Bytes per Cached_Object`, means the number of bytes in the serialized byte stream of primary copies only. Note that this equation does not address unserialized object size. Space requirements for backup copies are accounted for separately.

2. Calculate the available tenured generation in a cache server JVM.

   The available tenured generation is a function of the maximum heap size allocation and other user-specified JVM heap-sizing parameters. Another factor in the available tenured generation is the percentage of the heap that is available for storage. Typically, 66% is used as the maximum percentage of the heap available for storage, but this figure might be too low for your system. Make it a variable:

   ```
   Percent_of_Heap_Available_for_Storage = 0.66

   Available_Tenured_Generation = (Maximum_Heap_Size) * (Percent_of_Heap_
   Available_for_Storage)
   ```

3. Calculate the number of cache servers that will be needed.

   To calculate the number of cache servers that will be needed, divide the total heap requirement by the available tenured generation.

   ```
   Number_of_Cache_Servers = (Total_Heap_Requirement / Available_Tenured_
   Generation)
   ```

C Session Cache Configuration File

This appendix describes the contents of the session cache configuration file `default-session-cache-config.xml`. Coherence*Web uses the caches and services defined in the file to implement HTTP session management. This file is deployed in the `WEB-INF/classes` directory. Table C–1 describes the default cache-related values used in the `default-session-cache-config.xml` file.

Table C–1 Cache-Related Values Used in default-session-cache-config.xml

Value	Description
`local-attribute-storage`	This local cache is used to store attributes that are not distributed. This can happen under these 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-preserve-attributes` is set to `true`, then non-serializable attributes will be placed in the cache. Table A–1 describes this parameter.
`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. Table A–1 describes this parameter.
`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, because it is expected that there will a few read-mostly attributes.
`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.
`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 1 K will be stored in this cache. This is configured as a distributed cache.
`session-storage`	This cache is used to store session models. By default it is mapped to a near cache backed by a distributed cache because it is expected that a container will access and modify a subset of sessions multiple times (if sticky session load balancing is configured.) See "Session Models" on page 5-2 for more information.

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

Table C–2 Services-Related Values Used in default-session-cache-config.xml

Value	Description
`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 if a JVM stores and manages data for these caches. Under most circumstances, this should be set to `false` for Web container JVMs. See "Deployment Topologies" on page 5-20 for more details.
`ReplicatedSessionsMisc`	This replicated service is used by the `session-management` and `servletcontext-storage` caches.
`SessionOwnership`	This invocation service is used by the sticky session optimization feature (if `coherence-sticky-sessions` is set to `true`).

Example C–1 illustrates the contents of the `default-session-cache-config.xml` file. The cache- and services-related values described in Table C–1 and Table C–2 appear in **`bold`**.

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

```
<?xml version="1.0"?>
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!--                                                                     -->
<!--      Cache configuration descriptor for Coherence*Web               -->
<!--                                                                     -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"

xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
coherence-cache-config.xsd">

  <scope-name>oracle.coherence.web</scope-name>

  <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 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>
    <request-timeout>30s</request-timeout>
    <backing-map-scheme>
      <local-scheme>
        <scheme-ref>unlimited-local</scheme-ref>
      </local-scheme>
    </backing-map-scheme>
    <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>

<!--
"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>
  <request-timeout>30s</request-timeout>
  <backing-map-scheme>
    <local-scheme>
      <scheme-ref>unlimited-local</scheme-ref>
    </local-scheme>
  </backing-map-scheme>
  <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>
```

D

Session Cache Confguration File Without a Near Cache

This appendix illustrates a sample session cache configuration file (`session-cache-config.xml`) that does not contain a near cache configuration. Typically, you would omit the near cache configuration when you are using local session instances. "Getting Concurrent Access to the Same Session Instance" on page 5-28 describes a use case where you would use local session instances instead of a near cache.

In Example D–1, the `scheme-name` under `cache-mapping` has been changed from `session-storage` to `session-distributed`. The `near-scheme` section has been removed, but the `local-scheme` and `distributed-scheme` remain. This is noted in **`bold`** font in the example.

Example D–1 session-cache-config.xml File Without a Near Cache Configuration

```
<?xml version="1.0"?>
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!--                                                                    -->
<!--      Cache configuration descriptor for Coherence*Web              -->
<!--                                                                    -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"

xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
coherence-cache-config.xsd">

  <caching-scheme-mapping>
    <!--
    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.
    -->
<!--
session-near has been changed to session-distributed
-->
    <cache-mapping>
      <cache-name>session-storage</cache-name>
      <scheme-name>session-distributed</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 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>
      <request-timeout>30s</request-timeout>
      <backing-map-scheme>
        <local-scheme>
          <scheme-ref>unlimited-local</scheme-ref>
        </local-scheme>
      </backing-map-scheme>
      <autostart>true</autostart>
    </replicated-scheme>

<!--
The near-scheme section has been removed
```

```
-->
    <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>
      <request-timeout>30s</request-timeout>
      <backing-map-scheme>
        <local-scheme>
          <scheme-ref>unlimited-local</scheme-ref>
        </local-scheme>
      </backing-map-scheme>
```

```
        <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>
```

E

Oracle Coherence*Web Extension for OVAB

This appendix describes Coherence*Web extensions for *Oracle Virtual Assembly Builder*, a tool for virtualizing installed Oracle components, modifying those components, and then deploying them into your own environment. Using Oracle Virtual Assembly Builder, you capture the configuration of existing software components in artifacts called software *appliances*. Appliances can then be grouped, and their relationships defined into artifacts called software *assemblies* which provide a blueprint describing a complete multi-tier application topology.

The Oracle Coherence*Web introspection extension for Coherence*Web extends the functionality of the WebLogic Server Introspector. It examines the configuration of Coherence cache clusters and servers configured as part of a WebLogic domain.

This appendix has the following sections:

Versions Supported

The plug-in extension works with Oracle WebLogic Server 11gR1 version 11.1.1.4.0, which includes Coherence 3.6.

Oracle Coherence*Web Introspection Parameters

There are no additional parameters required beyond those needed by Oracle WebLogic Server. For the parameters required by WebLogic Server, see “Using the Introspection Plug-in for Oracle Virtual Assembly Builder,” in *Administering Server Environments for Oracle WebLogic Server*.

Reference System Prerequisites

There are no additional prerequisites beyond those defined by Oracle WebLogic Server. For the prerequisites required by WebLogic Server, see “Using the Introspection Plug-in for Oracle Virtual Assembly Builder,” in *Administering Server Environments for Oracle WebLogic Server*.

Requirements

Oracle Coherence*Web has the following requirements:

Note: In addition to the Coherence*Web requirements described in this section, all of WebLogic Server requirements must also be satisfied. For the WebLogic Server requirements, see “Using the Introspection Plug-in for Oracle Virtual Assembly Builder,” in *Administering Server Environments for Oracle WebLogic Server*.

Deployment Model Requirement

The plug-in extension requires you to use an out-of-process deployment model for Oracle Coherence*Web, in which storage-enabled cache servers are executed as separate processes rather than running within Oracle WebLogic Server. For more information on the out-of-process deployment model, see "Out-of-Process Topology" on page 5-20.

Requirement to Manually Update Custom Cluster Configuration Files

The plug-in extension examines Oracle Coherence*Web configuration defined through the WebLogic Server Administration Console and WebLogic Server MBeans (including WLST). It does not examine or modify custom cluster configuration files such as `tangosol-coherence-override.xml`. Custom cluster configuration files are passed through to the deployed environment, but no configuration changes are made to those files to reflect the deployed environment.

After deployment, ensure that you make appropriate manual configuration changes to any custom cluster configuration files.

Resulting Artifact Type

For each Coherence cluster that is defined in an introspected WebLogic domain, the plug-in extension creates a new appliance within the atomic Oracle WebLogic Server assembly.

Wiring

No wiring can be performed for Coherence cluster appliances. Each cluster appliance has a fixed, pre-defined connection to the domain's Administration Server, which is used at rehydration time to modify the cluster's configuration.

Wiring Properties

None.

Oracle Coherence*Web Appliance Properties

Each Oracle Coherence*Web cluster appliances has the following system and user properties:

Table E–1 describes Oracle Coherence*Web cluster appliance system properties:

Table E–1 Oracle Coherence*Web Appliance System Properties

Name	Type	Req'd	Default	Description
cache-servers	String	false	none	A list of the cache servers that are part of the cluster.
targets	String	false	none	A list of WebLogic Server Managed Servers that are part of the cluster.
<cacheserver>.node-manager-type	String	false	none	For each cache server in the above list, there is a property indicating the Node Manager type.
well-known-addresses	String	false	none	A list of well-known-addresses defined for the cluster. If no well-known-address are defined for this cluster (meaning it uses multicast), then this property will not be present.
<wellknownaddress>.server	String	false	none	For each of the well-known-addresses in the above list, there is a property indicating which cache server the well known address maps to (based on matching listen address and port information).

Table E–2 describes Oracle Coherence*Web cluster appliance user properties:

Table E–2 Oracle Coherence*Web Appliance User Properties

Name	Type	Req'd	Default	Description
<cacheserver>.node-manager-port	String	false	none	For each of the cache servers in the cluster, the Node Manager port is listed and may be modified by the user.
<cacheserver>.unicast-listen-port	String	false	none	For each of the cache servers in the cluster, the unicast listen port of that server is listed and may be modified by the user.
multicast-listen-address	String	false	none	The cluster-wide multicast listen address. If one or more well-known-addresses are listed (meaning the cluster uses unicast for cluster discovery), then this multicast property will not be present.
multicast-listen-port	String	false	none	The cluster-wide multicast listen port. If one or more well-known-addresses are listed (meaning the cluster uses unicast for cluster discovery), then this multicast property will not be present.
unicast-listen-port	String	false	none	The default unicast listen port for the cluster. This value is used by any cache servers that do not have a unicast listen port defined, as well as by any WebLogic Server Managed Servers that join the cluster.

Table E–2 (Cont.) Oracle Coherence*Web Appliance User Properties

Name	Type	Req'd	Default	Description
<wellknownaddress>.server	String	true	none	If any of the defined well known addresses could not be correlated with a cache server (based on matching listen address and port information), they will be listed here, and the user is responsible for specifying a cache server name to be used as the well known address. This property is mandatory, meaning it must be specified either as an appliance property or via a deployment plan.

Supported Template Types

The supported template type is Oracle Enterprise Linux (OEL).