Oracle® Coherence

User's Guide for Oracle Coherence*Web

Release 3.6.1

E15829-02

December 2010

ORACLE®

Oracle Coherence User's Guide for Oracle Coherence*Web, Release 3.6.1

E15829-02

Primary Author: Tom Pfaeffle

Contributing Author: Noah Arliss, Ballev Bihani, Torkel Dominique, Mark Falco, Alex Gleyzer, Gene Gleyzer, Jason Howes, James Kirsch, Adam Leftik, Rosemary Marano, Rob Misek, Patrick Peralta, Everett Williams

Contents

3 Installing Coherence*Web on Other Application Servers

4 Coherence*Web Session Management Features

List of Examples

List of Figures

List of Tables

Preface

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

Audience

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

Documentation Accessibility

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

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/support/contact.html or visit http://www.oracle.com/accessibility/support.html if you are hearing impaired.

Related Documents

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

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

Conventions

The following text conventions are used in this document:

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

1 Introduction

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

- brings Coherence data grid's data scalability, availability, reliability, and performance to in-memory session management and storage.
- supports fine-grained session and session attribute scoping by way of pluggable policies (see "Session and Session Attribute Scoping" on page 4-7).
- supports mainstream application servers such as Oracle WebLogic Server, IBM WebSphere, Tomcat, and so on (see "Supported Web Containers" on page 1-1).
- supports numerous portal containers, including Oracle WebLogic Portal (see Chapter 5, "Using Coherence*Web on WebLogic Portal").
- allows storage of session data outside of the Java EE application server, freeing application server heap space and enabling server restarts without session data loss (see "Deployment Topologies" on page 4-16).
- enables session sharing and management across different Web applications, domains and heterogeneous application servers (see "Session and Session Attribute Scoping" on page 4-7).
- supports multiple advanced session models (that is, Monolithic, Traditional, and Split Session) which define how the session state is serialized/deserialized in the cluster (see "Session Models" on page 4-2).

Coherence*Web and other Application Servers

For third-party application servers and a few legacy versions of WebLogic Server (9.2.1, 9.2.3, 10.3, 10.3.1), Coherence*Web provides a generic installer, the *WebInstaller*, that transparently instruments your Web applications. Chapter 3, "Installing Coherence*Web on Other Application Servers," describes how to use the WebInstaller to install Coherence*Web on these servers.

Supported Web Containers

Table 1–1 summarizes the Web containers supported by the Coherence*Web Session Management Module. It also provides links to the information required to install Coherence*Web. Notice that all of the Web containers (except Oracle WebLogic Server 10.3.3) share the same general installation instructions. A few, such as Caucho Resin, and WebLogic 10.*x*, require extra, container-specific steps that you must complete before starting the general installation instructions.

To install the Coherence*Web Session Management Module on WebLogic Server 10.3.1 and later, you can use SPI-based installation. For instructions on installing the Management Module on WebLogic Server 10.3.3, see Chapter 2, "Using Coherence*Web with WebLogic Server."

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

Table 1–1 Web Containers Supported by Coherence*Web

Application Server	Server Type Alias	See this Installation Section
Apache Tomcat 5.5.*x*	Tomcat/5.5.x	"General Instructions for Installing Coherence*Web Session Management Module" and "Enabling Sticky Sessions for Apache Tomcat Servers"
Apache Tomcat 6.0.*x*	Tomcat/6.0.x	"General Instructions for Installing Coherence*Web Session Management Module" and "Enabling Sticky Sessions for Apache Tomcat Servers"
Caucho Resin 3.1.*x*	Resin/3.1.*x*	"Installing on Caucho Resin 3.1.x"
IBM WebSphere 5.*x*	WebSphere/5.*x*	"General Instructions for Installing Coherence*Web Session Management Module"
IBM WebSphere 6.*x*	WebSphere/6.*x*	"General Instructions for Installing Coherence*Web Session Management Module"
IBM WebSphere 7.*x*	WebSphere/7.*x*	"General Instructions for Installing Coherence*Web Session Management Module" and "Decoding URL Session IDs for IBM WebSphere 7.x Servers"
JBoss Application Server	Generic or Jetty/5.1.*x*	"General Instructions for Installing Coherence*Web Session Management Module"
Jetty 5.1.*x*	Jetty/5.1.*x*	"General Instructions for Installing Coherence*Web Session Management Module"
Jetty 6.1.*x*	Generic	"General Instructions for Installing Coherence*Web Session Management Module"
Oracle OC4J 10.1.2.x	Oracle/10.1.2.x	"General Instructions for Installing Coherence*Web Session Management Module"
Oracle OC4J 10.1.3.*x*	Oracle/10.1.3.*x*	"General Instructions for Installing Coherence*Web Session Management Module"
Oracle WebLogic 9.2 MP1 Oracle WebLogic 10.3	WebLogic/9.x WebLogic/10.x	For WebInstaller and SPI-based installations, see the User's Guide for Oracle Coherence*Web, Release 3.5
Oracle WebLogic 10.*x*	WebLogic/10.*x*	For WebInstaller installations, see "Installing on Oracle WebLogic 10.x" on page 3-2
Sun Application Server 8.*x*	Generic	"General Instructions for Installing Coherence*Web Session Management Module"
Sun GlassFish 2.*x*	Generic	"General Instructions for Installing Coherence*Web Session Management Module"

Installation and Deployment Road Map

This section provides a general outline of the deployment decisions you should make before you configure and 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 may be deploying multiple applications to the container that require the use of the same cluster (or one Coherence node); you may have multiple Web applications packaged in a single EAR file that want to use a single cluster; or you may have Web applications that must keep their session data separate and 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 4-10.

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 4-13.

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 4-7.

Choose When to Clean Up Expired HTTP Sessions

Coherence*Web provides a Session Reaper which invalidates sessions that have expired. "Cleaning Up Expired HTTP Sessions" on page 4-27 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 supported by Coherence*Web.

- For WebLogic Server 11gR1 and later, and use the native WebLogic Server SPI-based installation procedure. See Chapter 2, "Using Coherence*Web with WebLogic Server."

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

- For other application servers, use the generic Java EE Web application instrumentation. See Chapter 3, "Installing Coherence*Web on Other Application Servers."

2

Using Coherence*Web with WebLogic Server

Note: Except where noted, this chapter pertains to the 11g Release 2 (10.3.3) of WebLogic Server.

Coherence*Web provides session state persistence and management. It is a session management module that uses Coherence for storing and managing session data. This chapter describes how to set up and deploy Coherence*Web so that it can be used by applications running on WebLogic Server.

Coherence*Web is an alternative to WebLogic Server's 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 off-load HTTP session storage to an existing Coherence cluster
- you want to share session state across EAR files.

Overview of the Coherence*Web SPI

The Coherence*Web Service Provider Interface (SPI) consists of the `coherence-web-spi.war` file. To enable Coherence*Web functionality, the SPI also requires `coherence.jar`.

Coherence*Web SPI Configurations

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

- The Coherence*Web SPI 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.

The `session-cache-config.xml` file contains the Coherence cache configurations used by the Coherence*Web SPI. This file is located in the `coherence-web-spi.war` file under the `WEB-INF\classes` directory. Enter any cache configuration changes in the `session-cache-config.xml` file and then repackage it in the `coherence-web-spi.war` file.

Coherence*Web provides several session locking modes to control concurrent access of sessions. Both Coherence*Web and the Coherence*Web SPI employ Optimistic 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 4-13 for more information about locking modes.

By itself, the Coherence*Web SPI does not require a load balancer to run in front of the WebLogic Server tier. However, you will require a load balancer if you employ sticky session optimization with any of the non-Optimistic locking modes. 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 here:

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

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

Coherence*Web includes a deployable shared library that contains a native plug-in to WebLogic Server's HTTP Session Management interface. The following steps summarize how to prepare your deployments to use Coherence*Web with applications running on WebLogic Server:

1. Download Oracle Coherence to your file system. See "Download Oracle Coherence".
2. Modify the `web.xml` file in the WAR deployment if your application requires advanced configuration for Coherence*Web. "Configure Coherence*Web" on page 2-3 describes the parameters that can be configured for Web applications. See Appendix A, "Coherence*Web Context Parameters" for descriptions of the entire set of Coherence*Web parameters.
3. (Optional) Configure the WebLogic-generated HTTP session cookie parameters in `weblogic.xml` or `weblogic-application.xml`. See "Configure the Session Cookies" on page 2-5.
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 2-8.
5. Determine the appropriate packaging based on your deployment requirements and follow the packaging instructions. Depending on your version of Weblogic Server, see "Configure Cluster Nodes (WebLogic Server 10.3.3 and Later)" on page 2-9 or "Configure Cluster Nodes (WebLogic Server 10.3.2 and Earlier)" on page 2-13.

Download Oracle Coherence

All of the files supporting Coherence*Web, including `coherence-web-spi.war` are included in the Coherence distribution.

By default, Coherence 3.5 is installed with WebLogic Server 10.3.3. If you are using Weblogic Server 10.3.3, you must replace the contents of the `coherence` directory that was installed with the server with the contents of the Coherence 3.6 distribution. The default location of the Coherence directory is `C:\Oracle\Middleware\coherence_3.5`.

If you are using WebLogic Server 10.3.2 or earlier, download the Coherence distribution to your file system.

Configure Coherence*Web

The Coherence*Web SPI provides a default configuration that should satisfy most Web applications. Table 2–1 lists only those Coherence*Web context parameters where the default for the SPI version is different from the non-SPI version. For a full 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 honored.

Table 2–1 Coherence*Web SPI Context Parameters

Parameter Name	Description
`coherence-application-name`	Coherence*Web uses the value of this parameter to determine the name of the application that uses `ApplicationScopeController` to scope attributes. The value for this parameter should be provided in the following format: *application name* + ! + *Web module name* where *application name* is the name of the application that uses `ApplicationScopeController` and *Web module name* is the name of the Web module in which it appears. For example, if you have an EAR named `test.ear` and a Web-module named `app1` defined in the EAR, then the default value for `coherence-application-name` 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 `ApplicationScopeController` *is* configured, then a warning is logged saying that the application name was not configured. See "Session Attribute Scoping" on page 4-9 for more information.
`coherence-factory-class`	The fully qualified class name of the `SessionHelper.Factory` to use. Other possible values include `com.tangosol.coherence.servlet.api`*XX*`.DefaultFactory` where *XX* is `22`, `23`, `24`, or `25` for Servlet 2.2, 2.3, 2.4, or 2.5 containers respectively. Default is `weblogic.servlet.internal.session.WebLogicSPIFactory`

*Table 2–1 (Cont.) Coherence*Web SPI Context Parameters*

Parameter Name	Description
coherence-reaperdaemon-assume-locality	This setting allows the reaper to assume that the sessions that are stored on this node (for example, by a distributed cache service) are the only sessions that this node must check for expiry. This value must be set to false if the session storage cache is being managed by nodes that are not running a reaper, for example if cache servers are being used to manage the session storage cache. 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. Default is false.
coherence-scopecontroller-class	This value specifies the class name of the optional com.tangosol.coherence.servlet.HttpSessionCollection$AttributeScopeController interface implementation to use. Valid values include: ■ com.tangosol.coherence.servlet.AbstractHttpSessionCollection$SimpleScopeController ■ com.tangosol.coherence.servlet.AbstractHttpSessionCollection$ApplicationScopeController (default) ■ com.tangosol.coherence.servlet.AbstractHttpSessionCollection$GlobalScopeController Default is com.tangosol.coherence.servlet.AbstractHttpSessionCollection$ApplicationScopeController
coherence-session-weblogic-compatibility-mode	This parameter is available only for the SPI version of 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-app. If it is false, then the standard behavior will apply: a single session ID will map to a single session instance using the Coherence*Web SPI in WebLogic. All other session persistence mechanisms in WebLogic use a single session ID in each web-app to refer to different session instances. Defaults to true unless the global scope controller is specified. If the controller is specified, then it defaults to false.

Configure the Session Cookies

If you run the Coherence*Web SPI, then WebLogic Server generates and parses the session cookie. Any native Coherence*Web session cookie configuration parameters will be ignored. Session cookies must be configured by using the session cookie parameters in weblogic.xml or weblogic-application.xml.

Table 2–2 describes the WebLogic-generated HTTP session cookie parameters that you can configure in weblogic.xml or weblogic-application.xml.

In this table, **Updatable?** indicates whether the value of the parameter can be changed while the server is running. n/a indicates that there is no corresponding Coherence cookie parameter.

Table 2–2 HTTP Session Cookie Parameters

This WebLogic-Generated Session Cookie Parameter...	Replaces this Coherence*Web Cookie Parameter	Description
`cookie-comment`	`n/a`	Specifies the comment that identifies the session tracking cookie in the cookie file. 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 from Sun Microsystems. 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*. 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 may set this to a more specific name for your application. Default is `JSESSIONID`. **Updatable?** Yes
`cookie-path`	`coherence-session-cookie-path`	Defines the session tracking cookie path. If not set, this attribute defaults to `/` (slash) where the browser sends cookies to all URLs served by WebLogic Server. You may set the path to a narrower mapping, to limit the request URLs to which the browser sends cookies. Default is `null`. **Updatable?** Yes
`cookie-secure`	`coherence-session-cookie-secure`	Tells the browser to only send the cookie back over an HTTPS connection. This ensures that the cookie ID is secure and should only be used on Web sites that use HTTPS. Session Cookies over HTTP no longer work if this feature is enabled. You should disable the `url-rewriting-enabled` element if you intend to use this feature. WebLogic Server generates the session cookie Default is `false`. **Updatable?** Yes

Table 2–2 (Cont.) HTTP Session Cookie Parameters

This WebLogic-Generated Session Cookie Parameter...	Replaces this Coherence*Web Cookie Parameter	Description
cookies-enabled	`coherence-session-cookies-enabled`	Use of session cookies is enabled by default and is recommended, but you can disable them by setting this property to `false`. You might turn this option off for testing purposes. Default is `true`. **Updatable?** Yes
`debug-enabled`	n/a	Enables the debugging feature for HTTP sessions. Support it by enabling `HttpSessionDebug` logging and the WebLogic Server trace logger. Default value is `false`. **Updatable?** Yes
`encode-session-id-in-query-params`	n/a	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. Default value is `false`. **Updatable?** Yes
`http-proxy-caching-of-cookies`	n/a	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. Default value is `true`. **Updatable?** Yes
`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 WAP application, you must use URL rewriting because the WAP protocol does not support cookies. Also, some WAP devices have a 128-character limit on URL length (including attributes), which limits the amount of data that can be transmitted using URL re-writing. To allow more space for attributes, use this attribute to limit the size of the session ID that is randomly generated by WebLogic Server. You can also limit the length to a fixed 52 characters, and disallow special characters, by setting the `WAPEnabled` attribute. For more information, see "URL Rewriting and Wireless Access Protocol" in *Developing Web Applications for WebLogic Server*. Default is 52. **Updatable?** No

Table 2–2 (Cont.) HTTP Session Cookie Parameters

This WebLogic-Generated Session Cookie Parameter...	Replaces this Coherence*Web Cookie Parameter	Description
`invalidation-interval-secs`	`coherence-reaperdaemon-cycle-seconds`	Sets the time, in seconds, that Coherence*Web waits between doing house-cleaning 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. Default is `60`. **Updatable?** No
`timeout-secs`	`coherence-session-expire-seconds`	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`. Default is `3600` seconds. **Updatable?** No
`tracking-enabled`	`n/a`	Enables session tracking between HTTP requests. WebLogic Server generates the HTTP response. 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. Default is `true`. **Updatable?** Yes

Start a Cache Server

A Coherence data node (also known as a cache server) is responsible for storing and managing all cached data. It can be either a dedicated JVM or run within a WebLogic Server instance. 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.

Whether you start the cache servers first or the WebLogic Server instances first, depends on the server topology you are employing.

- If you are using an In-Process topology (all storage-enabled WebLogic Server instances), then it does not matter if you start the cache servers first or WebLogic Server instances first.
- If you are using an Out-of-Process topology (storage-disabled WebLogic server instances and stand alone Coherence cache servers), then start the cache servers first, followed by the WebLogic 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 WebLogic Server startup.

To Start a Stand-Alone Coherence Data Node

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

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

   In this example, *cache_configuration_file* refers to the cache configuration in the `coherence-cache-config.xml` file. (**Note:** if you are only running Coherence*Web, then *cache_configuration_file* is `session-cache-config.xml`.) The cache configuration defined for the cache server must be the same as the configuration defined for the application servers which run on the same Coherence cluster.

   If you run Coherence*Web for session management, then the information about the cache configuration should be merged with the session configuration contained in the `session-cache-config.xml` file. Similarly, the cache and session configuration must be consistent across WebLogic and Cache servers.

2. Start one or more Coherence data nodes 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, include the command line property `-Dtangosol.coherence.session.localstorage=true` in the server startup command.

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

Configure Cluster Nodes (WebLogic Server 10.3.3 and Later)

Coherence*Web can have *application server-scope*, *EAR-scope*, or *WAR-scope*. Like Coherence clusters, scoping of Coherence*Web depends on the placement of the `coherence.jar` file in the classloader's hierarchy. You can find detailed information about each of the scopes in "Cluster Node Isolation" on page 4-10.

WebLogic Server 10.3.3 provides several features, collectively known as *ActiveCache*, that allow your applications to more easily interact with the Coherence cache. For a complete discussion of these features see the *Using ActiveCache* guide. To employ ActiveCache functionality in your applications, you must also deploy the `active-cache.jar` file, which you can find in the *WL_HOME*`/common/deployable-libraries` directory.

The following sections describe how you can configure the different cluster node configurations to run Coherence*Web:

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

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.

Configuring Application Server-Scoped Cluster Nodes

To add Coherence*Web for session management to a Coherence cluster, follow these steps:

1. Edit your WebLogic Server system classpath to include the `coherence.jar` and *`WL_HOME`*`/common/deployable-libraries/active-cache.jar` files in the system classpath. The `active-cache.jar` file should be referenced only from the `deployable-libraries` folder in the system classpath and should not be copied to any other location.
2. Use the WebLogic Server Administration Console or the command line to deploy `coherence-web-spi.war` file as a shared library.
3. Enable Coherence*Web in your Web application.

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

Example 2–1 Library Reference for a WAR File

```
<weblogic-web-app>
  ...
    <library-ref>
      <library-name>coherence-web-spi</library-name>
    </library-ref>
  ...
</weblogic-web-app>
```

4. (Optional) If you must configure Coherence cluster properties, create a `CoherenceClusterSystemResourceMBean` MBean and reference it in the `ServerMBean` MBean.

You can use WebLogic Scripting Tool (WLST) to reference the MBean. See `createServerScopedCoherenceSystemResource` in the *Using ActiveCache* book.

Configuring EAR-Scoped Cluster Nodes

To use Coherence*Web for session management in EAR-scoped cluster nodes, follow these steps:

1. Use the WebLogic Server Administration Console or the command line to deploy the `coherence.jar`, `active-cache.jar`, and `coherence-web-spi.war` files as shared libraries to all of the target servers where the application will be deployed.

   See "Install a Java EE Library" in the *Oracle Fusion Middleware Oracle WebLogic Server Administration Console Help*.

2. Reference the `coherence.jar`, `active-cache.jar`, and `coherence-web-spi.war` files in the `weblogic-application.xml` file. Store the file in the EAR's `META-INF/` directory.

   Example 2–2 illustrates a sample `weblogic-application.xml` file.

Example 2–2 Coherence, Coherence Web SPI, and ActiveCache Referenced in weblogic-application.xml

```
<weblogic-application ... >
...
      <library-ref>
           <library-name>coherence</library-name>
      </library-ref>
      <library-ref>
           <library-name>coherence-web-spi</library-name>
     </library-ref>
     <library-ref>
           <library-name>active-cache</library-name>
     </library-ref>
...
</weblogic-application>
```

3. (Optional) If you must configure Coherence cluster properties, create a `CoherenceClusterSystemResourceMBean` MBean and reference it as a `coherence-cluster-ref` element in `weblogic.xml` file. This element allows the applications to enroll in the Coherence cluster as specified by the `CoherenceClusterSystemResourceMBean` attributes. For more information, see the *Using ActiveCache* book.

   Example 2–3 illustrates a sample configuration. The `myCoherenceCluster` MBean in the example is of type `CoherenceClusterSystemResourceMBean`.

Example 2–3 Identifying a Coherence Cluster for EAR-Scoped Cluster Nodes

```
<weblogic-web-app>
...
  <coherence-cluster-ref>
    <coherence-cluster-name>
     myCoherenceCluster
    </coherence-cluster-name>
  </coherence-cluster-ref>
...
</weblogic-web-app>
```

Configuring WAR-Scoped Cluster Nodes

To use Coherence*Web for session management in WAR-scoped cluster nodes, follow these steps:

1. Use the WebLogic Server Administration Console or the command line to deploy the `active-cache.jar` and `coherence-web-spi.war` files as shared libraries to all of the target servers where the application will be deployed. See "Install a Java EE Library" in the *Oracle Fusion Middleware Oracle WebLogic Server Administration Console Help*.
2. Copy the `coherence.jar` file to the WAR file's `WEB-INF/lib` directory.
3. If you deploy the `coherence-web-spi.war` file as a shared library, you must also create a shared library reference by adding the stanza illustrated in Example 2–4 to the `weblogic.xml` file in the WAR file's `WEB-INF` directory.

Example 2–4 Library Reference for a Web Application

```
<weblogic-web-app ... >
     ...
      <library-ref>
           <library-name>coherence-web-spi</library-name>
     </library-ref>
     ...
<weblogic-web-app>
```

4. Create a `manifest.mf` file to reference the `active-cache.jar` file. Copy the `manifest.mf` file to each WAR file's `META-INF` directory. Example 2–5 illustrates a sample `manifest.mf` file.

Example 2–5 Sample manifest.mf File to Reference active-cache.jar

```
Extension-List: active-cache
active-cache-Extension-Name: active-cache
active-cache-Specification-Version: 1.0
active-cache-Implementation-Version: 1.0
```

5. (Optional) If you must configure Coherence cluster properties, create a `CoherenceClusterSystemResourceMBean` MBean and reference it as a `coherence-cluster-ref` element in the `weblogic.xml` or `weblogic-ejb-jar.xml` file. For more information, see the *Using ActiveCache* book.

   Example 2–6 illustrates a sample configuration for WAR-scoped cluster nodes in the `weblogic.xml` file. The `myCoherenceCluster` MBean is of type `CoherenceClusterSystemResourceMBean`.

Example 2–6 Identifying a Coherence Cluster for EAR-Scoped Cluster Nodes

```
<weblogic-web-app>
...
  <coherence-cluster-ref>
    <coherence-cluster-name>
     myCoherenceCluster
    </coherence-cluster-name>
  </coherence-cluster-ref>
...
</weblogic-web-app>
```

Configure Cluster Nodes (WebLogic Server 10.3.2 and Earlier)

WebLogic Server 10.3.2 and earlier does not support ActiveCache.

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

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

You can find detailed information about each of the options under "Cluster Node Isolation" on page 4-10.

> **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.

Configuring Application Server-Scoped Cluster Nodes

To add Coherence*Web for session management to a Coherence cluster, follow these steps:

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

   Add the library reference stanza illustrated in Example 2–7 to the `weblogic.xml` file in each WAR file deployed in the WebLogic Server that intends to use Coherence*Web.

Example 2–7 Library Reference for a WAR File

```
<weblogic-web-app>
     ...
      <library-ref>
           <library-name>coherence-web-spi</library-name>
     </library-ref>
     ...
</weblogic-web-app>
```

Configuring EAR-Scoped Cluster Nodes

To use Coherence*Web for session management in EAR-scoped cluster nodes, follow these steps:

1. Use the WebLogic Server Administration Console to deploy the `coherence.jar` and `coherence-web-spi.war` files as shared libraries to all of the target servers where the application will be deployed. See "Install a Java EE Library" in *Oracle Fusion Middleware Oracle WebLogic Server Administration Console Help* for more information.
2. Reference the `coherence.jar` and `coherence-web-spi.war` files in the `weblogic-application.xml` file. Store the file in the EAR's `META-INF` directory.

   Example 2–8 illustrates a `weblogic-application.xml` file.

Example 2–8 Coherence and Coherence Web SPI Referenced in weblogic-application.xml

```
<weblogic-application ...>
...
      <library-ref>
           <library-name>coherence</library-name>
      </library-ref>
      <library-ref>
           <library-name>coherence-web-spi</library-name>
     </library-ref>
...
</weblogic-application>
```

Configuring WAR-Scoped Cluster Nodes

To use Coherence*Web for session management in WAR-scoped cluster nodes, follow these steps:

1. Use the WebLogic Server Administration Console or the command line to deploy the `coherence-web-spi.war` file as a shared library to all of the target servers where the application will be deployed. See "Install a Java EE Library" in *Oracle Fusion Middleware Oracle WebLogic Server Administration Console Help*.
2. Copy the `coherence.jar` file to the WAR file's `WEB-INF/lib` directory.
3. If you deploy the `coherence-web-spi.war` file as a shared library, create a shared library reference by adding the stanza illustrated in Example 2–9 to the `weblogic.xml` file in the WAR file's `WEB-INF` directory.

Example 2–9 Library Reference for a Web Application

```
<weblogic-web-app>
     ...
      <library-ref>
           <library-name>coherence-web-spi</library-name>
     </library-ref>
     ...
<weblogic-web-app>
```

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 "/". 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. Since 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 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 lifecycle 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 lifecycle 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.

Securing Coherence*Web Deployments

To prevent unauthorized Coherence TCMP cluster members from accessing HTTP session cache servers, you can configure symmetric or asymmetric encryption filters. Coherence ships with two JCA-based encryption filters which can be used to protect the clustered communications for privacy and authenticity.

- symmetric filter, which uses symmetric encryption to protect cluster communications. The encryption key is generated from a shared password known to all cluster members. This filter is suitable for small deployments or where the maintenance and protection of a shared password is feasible.
- PKCS Encryption filter, which uses public key cryptography (asymmetric encryption) to protect the cluster join protocol, and then switches over to much faster symmetric encryption for service level data transfers. Unlike the symmetric encryption filter, there is no persisted shared secret. The symmetric encryption key is randomly generated by the cluster's senior member, and is securely transfer to authenticated cluster members as part of the cluster join protocol. This encryption filter is suitable for deployments where maintenance of a shared secret is not feasible.

See "Using Network Filters" in the *Developer's Guide for Oracle Coherence* for information on implementing these filters.

Enabling Coherence*Web SPI for ColdFusion Applications

If your ColdFusion applications run on WebLogic server, follow these steps to enable Coherence*Web for session management.

1. In the ColdFusion installer create a WAR version of ColdFusion.
2. Follow the provided instructions for configuring WebLogic Server for ColdFusion.
3. Extract the WAR into a directory for exploded directory deployment to WebLogic Server.
4. Create a `weblogic.xml` file in the `/WEB-INF` directory of the exploded ColdFusion WAR. Specify the Coherence SPI as a `<library-ref>`.

```
<?xml version="1.0"?>
<weblogic-web-app xmlns="http://www.bea.com/ns/weblogic/weblogic-web-app"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/ns/weblogic/weblogic-web-app http://www.
bea.com/ns/weblogic/weblogic-web-app/1.0/weblogic-web-app.xsd">
   <library-ref>
      <library-name>coherence-web-spi</library-name>
   </library-ref>
</weblogic-web-app>
```

5. Copy the `coherence.jar` to the `WEB-INF/lib` directory of the exploded application.
6. Copy `cfusion.jar` from `WEB-INF/cfusion/lib` to `WEB-INF/`.
7. Start the WebLogic Server instance. Deploy the Coherence*Web SPI to WebLogic. The SPI is located at `coherence\lib\coherence-web-spi.war` in the Coherence distribution.
8. Deploy the ColdFusion application to the WebLogic Server instance.
9. Configure ColdFusion MX to use J2EE session management.
   a. Access the ColdFusion administration page at the following URL:

      `http://<host>:<port>/coldfusion-context-root/CFIDE/administrator.`

   b. Select **Memory Variables** under **Server Sessions** and enable the **Use J2EE session variables** checkbox.
10. Add your ColdFusion application under the exploded ColdFusion WAR.

The application must have a `Application.cfm` file that specifies `sessionmanagement="Yes"`, but should not configure the session using ColdFusion configuration (otherwise, exceptions are thrown). The `Application.cfm` should contain the following line:

```
<cfapplication sessionmanagement="Yes">
```

3

Installing Coherence*Web on Other Application Servers

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

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

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

- Use the WebInstaller approach described in this chapter
- Use the SPI-based installation for WebLogic Server 11gR1 or later. See Chapter 2, "Using Coherence*Web with WebLogic Server."

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 inspect step and an install step. For more information on what the installer does during these steps, see "How the Coherence*Web Installer Instruments a Java EE Application" on page 3-8.

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 3-5.

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 3-3.

A few of the Web containers, such as Caucho Resin, and WebLogic 10.*x*, 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 10.*x*

Complete the following steps to install the Coherence*Web Session Management Module into an Oracle WebLogic 10-10.2 server:

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

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

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

   On Windows:

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

   On UNIX:

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

2. For each WebLogic 10.*x* 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
   ```

   For example, on Windows:

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

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

Installing on Caucho Resin 3.1.*x*

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

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

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

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

   On Windows:

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

   On UNIX:

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

2. For each Resin installation that will be running in the server cluster, update the libraries using the following command:

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

   For example, on Windows:

   ```
   java -cp coherence.jar;coherence-web.jar
   com.tangosol.coherence.servlet.WebPluginInstaller C:\opt\resin31 -install
   ```

3. Follow the instructions described in "General Instructions for Installing Coherence*Web Session Management Module" on page 3-3 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 3-5 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.x Servers" on page 3-5 for additional instructions.

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

1. Make sure that the application directory and the `.ear` file or `.war` file are not being used or accessed by another process.
2. Change the current directory to the Coherence library directory (*%COHERENCE_HOME%*`\lib` on Windows and *$COHERENCE_HOME*`/lib` on UNIX).
3. Make sure that the paths are configured so that Java commands will run.
4. Complete the application inspection step by running the following command. Specify the full path to your application and the name of your server found in Table 1–1 (replacing the `<app-path>` and `<server-type>` with them in the command line below):

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

   The system will create (or update, if it already exists), the `coherence-web.xml` configuration descriptor file for your Java EE application in the directory where the application is located. This configuration descriptor contains the default Coherence*Web settings for your application recommended by the installer.

5. 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. Appendix A, "Coherence*Web Context Parameters," describes the Coherence*Web settings that can be modified. Use the `param-name` and `param-value` subelements of `context-param` to enable the features you want. The following list provides some examples of different settings:

- The setting inTable 3–1 will cluster 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:

Table 3–1 Settings to Cluster ServletContext Attributes

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

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

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

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

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

Table 3–3 Settings to Increase Length of HTTPSession ID

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

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

Table 3–4 Settings to Support URI Encoding

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

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 to reside in the same directory as the application.

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.

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.*x* 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.*x* 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.*x* 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 WebInstaller Ant task
- Configuring the WebInstaller Ant Task
- WebInstaller Ant Task Examples

Using the WebInstaller Ant task

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

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

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

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

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

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

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

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

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

Configuring the WebInstaller Ant Task

Table 3–5 describes the attributes that can be used with the Coherence*Web WebInstaller Ant Task.

Table 3–5 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. This attribute defaults to `coherence-web.xml` in the directory that contains the target Java EE application.	No
failonerror	Stop the Ant build if the Coherence*Web installer exits with a status other than 0. The default is `true`.	No
nowarn	Suppress warning messages. This attribute can be either `true` or `false`. The default is `false`.	No
operations	A comma- or space-separated list of operations to perform; each operation must be one of `inspect`, `install`, `uninstall`, or `version`.	Yes
server	The alias of the target Java EE application server.	No
touch	Touch JSPs and TLDs that are modified by the Coherence*Web installer. This attribute can be either `true`, `false`, or `M/d/y h:mm a`' The default is `false`.	No
verbose	Show verbose output. This attribute can be either `true` or `false`. The default is `false`.	No

WebInstaller Ant Task Examples

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

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

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

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

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

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

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

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

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

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

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

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

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

```
java -jar coherence-loadbalancer.jar server1:80 server1:7001 server2:7001
server3:7001
```

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

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

Table 3–6 describes the command line options for the load balancer:

Table 3–6 Load Balancer Command Line Options

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

How the Coherence*Web Installer Instruments a Java EE Application

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

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

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

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

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

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

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

2. Adds the Coherence*Web configuration context parameters generated in Step (1) of the inspect step to the `web.xml` descriptor of each Web application contained in the target Java EE application.
3. Unregisters any application-specific `ServletContextListener`, `ServletContextAttributeListener`, `ServletRequestListener`, `ServletRequestAttributeListener`, `HttpSessionListener`, and `HttpSessionAttributeListener` classes (including those registered by TLDs) from each Web application.
4. Registers a Coherence*Web `ServletContextListener` in each `web.xml` descriptor. At run time, the Coherence*Web `ServletContextListener` propagates each `ServletContextEvent` to each application-specific `ServletContextListener`.
5. Registers a Coherence*Web `ServletContextAttributeListener` in each `web.xml` descriptor. At run time, the Coherence*Web `ServletContextAttributeListener` propagates each `ServletContextAttributeEvent` to each application-specific `ServletContextAttributeListener`.
6. Wraps each application-specific `Servlet` declared in each `web.xml` descriptor with a Coherence*Web `SessionServlet`. At run time, each Coherence*Web `SessionServlet` delegates to the wrapped `Servlet`.
7. Adds the following directive to each JSP enumerated in Step (2) of the inspect step:

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

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

Installing Coherence*Web into Applications using Java EE Security

> **Note:** This section does not apply to the native WebLogic SPI implementation of Coherence*Web. It applies only if you are using the WebInstaller to install Coherence*Web into an application that uses Java EE security. For instructions on using the SPI implementation, see Chapter 2, "Using Coherence*Web with WebLogic Server."

If you want 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 which is different from the one used by the target Web container.

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

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

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

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

Enabling Coherence*Web for ColdFusion Applications

Note: This section applies only to application servers that do not use the Coherence*Web SPI.

Oracle Coherence*Web can be employed as the session management module for ColdFusion applications. Follow these steps to enable Coherence*Web with for your application server.

1. In the ColdFusion installer, create a WAR version of ColdFusion.
2. Follow the provided instructions for configuring your J2EE container for ColdFusion.

   If you run Caucho Resin as your application server, run the Coherence WebInstaller on the container. See "Installing on Caucho Resin 3.1.x" on page 3-2 for more information.
3. Instrument the generated `cfusion.war` using the Coherence WebInstaller.

   See "General Instructions for Installing Coherence*Web Session Management Module" on page 3-3 for the additional steps to install Coherence*Web for a Java EE application.
4. Deploy `cfusion.war` to the Web container.
5. Configure ColdFusion MX to use J2EE session management.
   a. Access the ColdFusion administration page at the following URL:

      ```
      http://<host>:<port>/coldfusion-context-root/CFIDE/administrator
      ```

   b. Select **Memory Variables** under **Server Sessions** and enable the **Use J2EE session variables** check box.
6. Add your ColdFusion application.

   The application must have a `Application.cfm` file that specifies `sessionmanagement="Yes"`, but should not configure the session using ColdFusion configuration (otherwise, exceptions are thrown). The `Application.cfm` should contain the following line:

   ```
   <cfapplication sessionmanagement="Yes">
   ```

4

Coherence*Web Session Management Features

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
- Managing and Monitoring Applications with JMX, which describes how you can display the HTTP session management attributes and operations for Web applications that use Coherence*Web
- Running Performance Reports, which describes the preconfigured reports that help you manage capacity and troubleshoot problems
- Cleaning Up Expired HTTP Sessions, which describes how you can clean up expired HTTP sessions and free associated memory
- 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 and/or its attributes remain local (stored on the originating server's heap and only accessible by that server) or distributed (stored within the Coherence grid, and thus, accessible to other server JVMs)
- Configuring Coherence*Web with Coherence*Extend, which describes how to configure Coherence*Web so that it can use Coherence*Extend to connect Web container JVMs to the cluster

- Configuring Coherence*Web for JSF and MyFaces, which describes how to configure Coherence*Web if you are using Java Server Faces (JSF) or MyFaces to build user interfaces for Web applications

Session Models

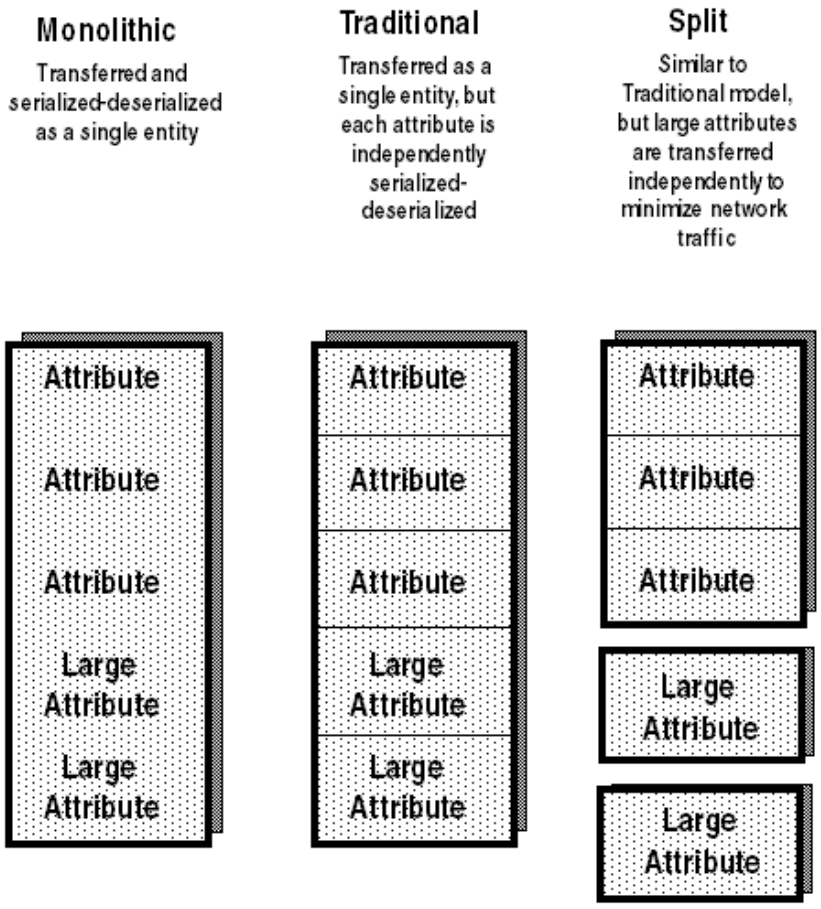
A session model describes how Coherence*Web stores session state in Coherence. Session data is managed by an `HttpSessionModel` object while the session collection in a Web application is managed by an `HttpSessionCollection` object. You must configure only the collection type in `web.xml`—the model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

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

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

Figure 4–1 illustrates the three session models.

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

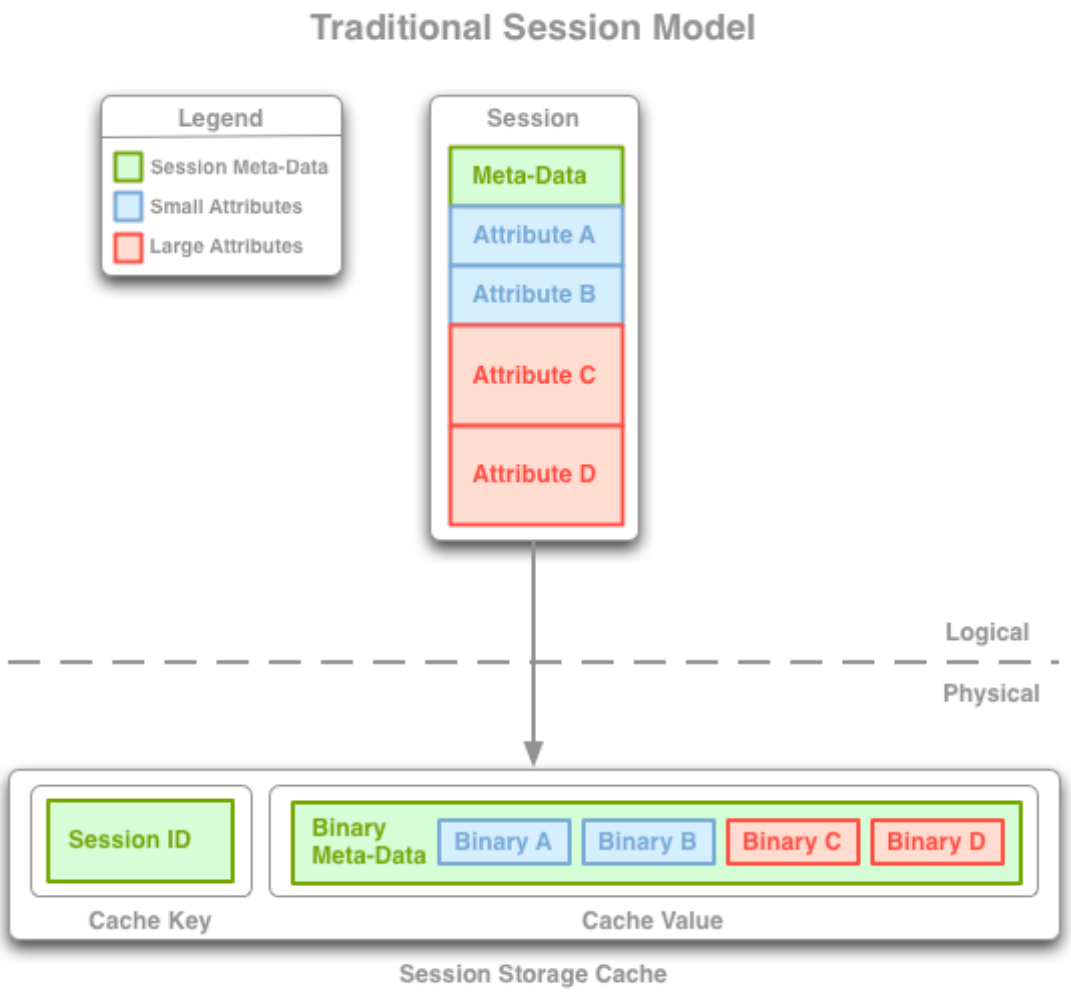


Traditional Model

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

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

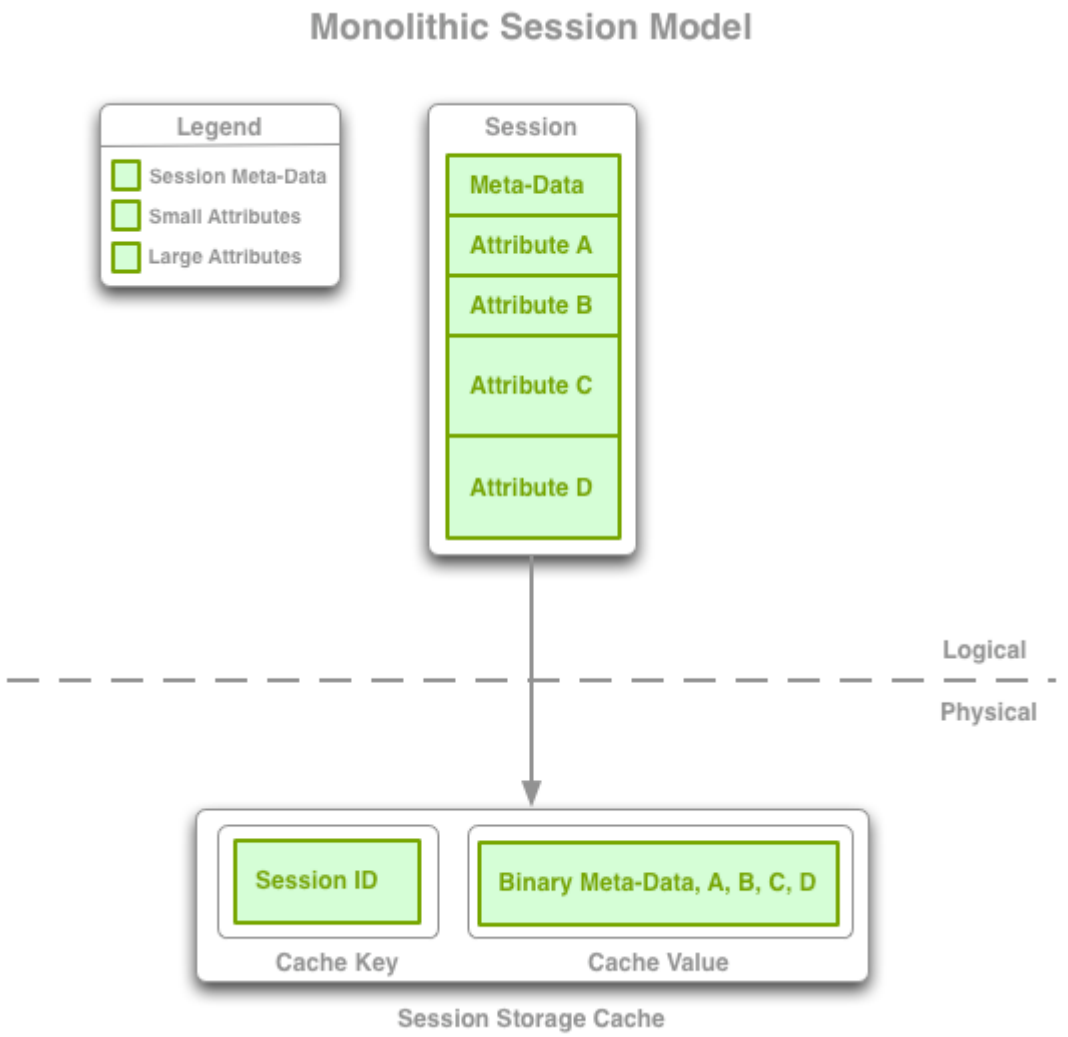
Figure 4–2 Traditional Session Model



Monolithic Model

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

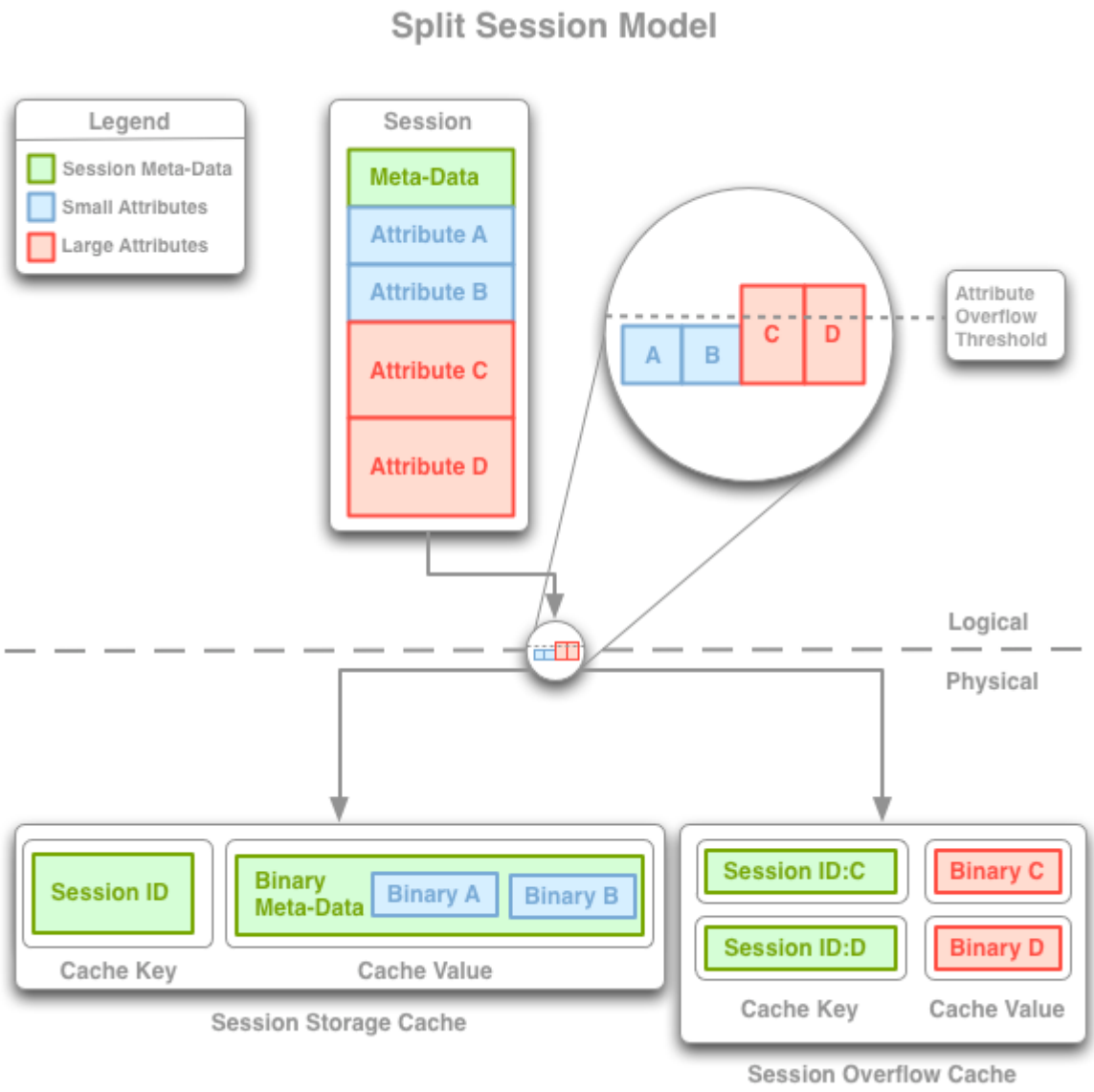
Figure 4–3 Monolithic Session Model



Split Model

`SplitHttpSessionModel` and `SplitHttpSessionCollection` 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 out into separate cache entries to be managed individually, thus supporting very large HTTP session objects without unduly increasing the amount of data that must be accessed and updated within the cluster on each request. In other words, only the large attributes that are modified within a particular request incur any network overhead for their updates, and (because it uses Near Caching) the Split Model generally does not incur any network overhead for accessing either the core HTTP session data or any of the session attributes.

Figure 4–4 Split Session Model



Session Model Recommendations

The following list offers some recommendations on which session model to choose for your applications:

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

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

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

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 contexts paths `/web/HRPortal` and `/web/InWeb`, you should set the `coherence-session-cookie-path` parameter to `/web`. On the other hand, if the two Web applications are registered under the context paths `/HRPortal` and `/InWeb`, you should set the `coherence-session-cookie-path` parameter to `/`.

If the Web applications that you would like to share session data are deployed on different Web containers running on different machines (that are not behind a common load balancer), you must also 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 may want to explicitly prevent HTTP session data from being shared by different Java EE applications that participate in the same Coherence cluster. For example, assume you have two applications `HRPortal` and `InWeb` that share cached data in their EJB tiers but 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 `ApplicationScopeController` to define the scope of an applications's attributes. "Session Attribute Scoping" on page 4-9 describes this technique. Another way is to specify a unique session cache service name for each application.

To specify a unique session cache service name for each application:

1. Locate the `<service-name/>` elements in each `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 `session-cache-config.xml` file with the application.

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

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

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

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

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

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 may want to scope individual session attributes so that they are either globally visible (that is, all Web applications can see and modify these attributes) or scoped to an individual Web application (that is, not visible to any instance of another application).

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

Two implementations of the `AttributeScopeContoller` interface are available out of the box: `ApplicationScopeController` and the `GlobalScopeController`. The `GlobalScopeController` implementation does not scope attributes, while `ApplicationScopeController` scopes all attributes to the application by pre-pending the name of the application to all attribute names.

You can 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` 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` 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 4–2 illustrates the `GlobalScopeController` interface specified in the `web.xml` file.

Example 4–2 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:

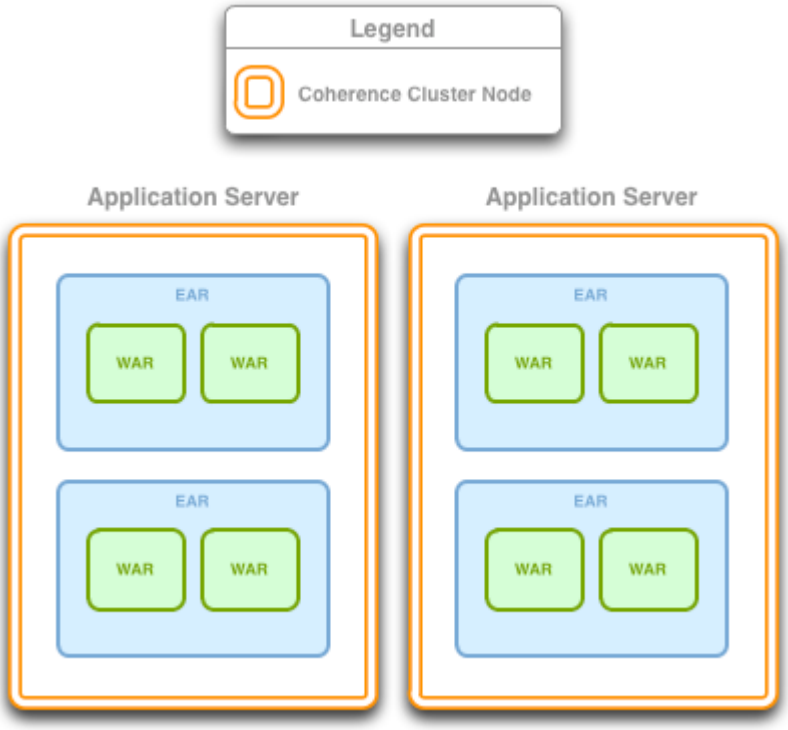
- the number of Coherence nodes that are created within an application server JVM
- where the Coherence library is deployed

Applications can be application server-scoped, EAR-scoped, or WAR-scoped. This section describes these considerations. For detailed information on the XML configuration for each of these options, see "Configure Cluster Nodes (WebLogic Server 10.3.3 and Later)" on page 2-9.

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, since the Coherence library (`coherence.jar`) is deployed in the container's classpath, only one copy of the Coherence classes is loaded into the JVM. This minimizes the use of resources. On the other hand, since all applications are using the same cluster node, all applications are affected if one application misbehaves.

Figure 4–5 Application Server-Scoped Cluster



Requirements for using this configuration are:

- each deployed application must use the same version of Coherence and participate in the same cluster
- the classes of objects placed in the HTTP session must be available

"Configuring Application Server-Scoped Cluster Nodes" on page 2-10 describes the XML configuration for application server-scoped cluster nodes.

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

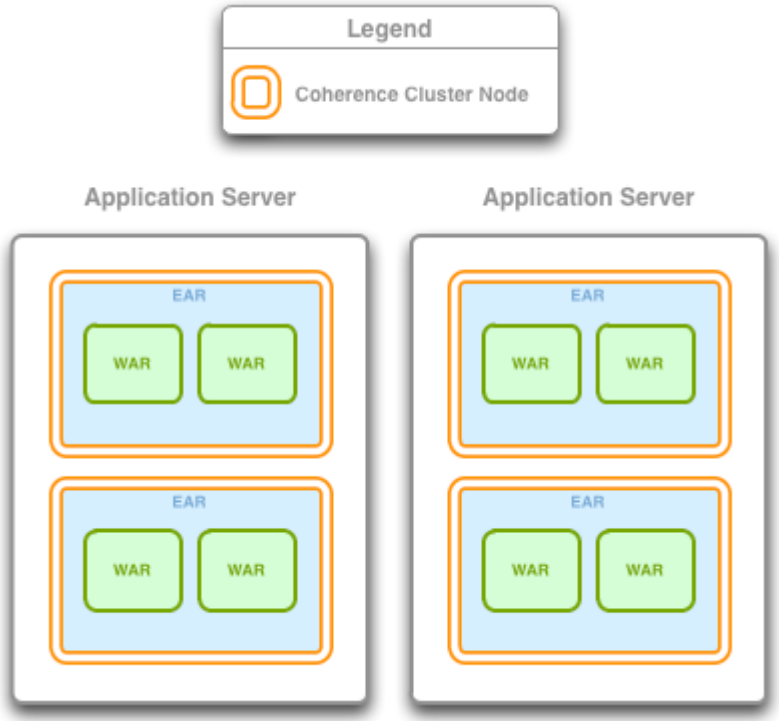
An example of such an environment may be a deployment where multiple application groups are deploying applications written independently, without carefully coordinating and enforcing their conventions and naming standards. With this configuration, all applications are part of the same cluster and the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

For these reasons, Oracle Coherence strongly recommends that you use EAR-scoped and WAR-scoped cluster node configurations. If you are in doubt 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 become part of one Coherence node. This configuration produces one Coherence node for each deployed EAR that uses Coherence*Web. Since the Coherence library (`coherence.jar`) is deployed in the application's classpath, only one copy of the Coherence classes is loaded for each EAR. Since all Web applications in the EAR use the same cluster node, all Web applications in the EAR are affected if one of the Web applications misbehaves.

Figure 4–6 EAR-Scoped Cluster



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

Requirements for using this configuration are:

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

"Configuring EAR-Scoped Cluster Nodes" on page 2-11 describes the XML configuration for EAR-scoped cluster nodes.

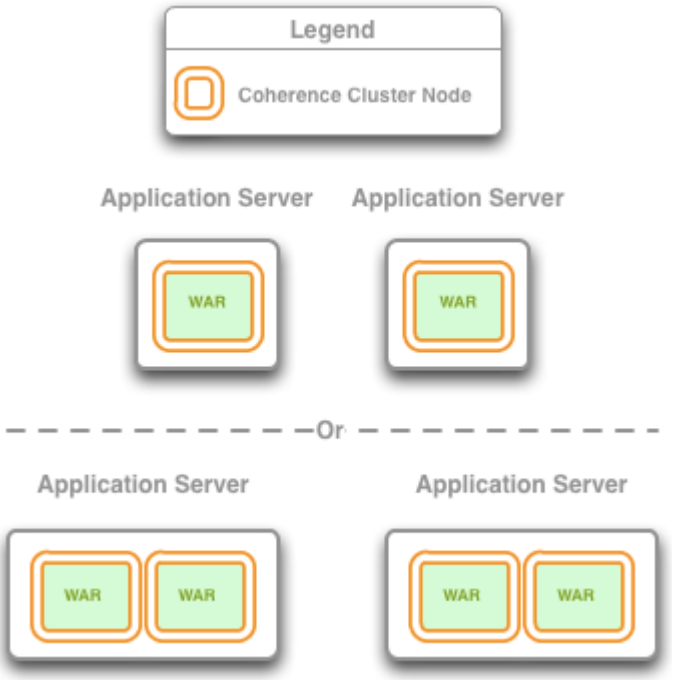
WAR-Scoped Cluster Nodes

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 that uses Coherence*Web) and since the Coherence library (`coherence.jar`) is deployed in the Web application's classpath, there will be as many copies of the Coherence classes loaded as there are deployed WARs. This results in the largest resource utilization out of the three options.

However, since each deployed Web application is its own cluster node, Web applications are completely isolated from other potentially misbehaving Web applications.

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

Figure 4–7 WAR-Scoped Clusters



Requirements for using this configuration are:

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

"Configuring WAR-Scoped Cluster Nodes" on page 2-14 describes the XML configuration for WAR-scoped cluster nodes.

Session Locking Modes

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

- Optimistic Locking, which allows concurrent access to a session by multiple threads in a single JVM or multiple JVMs while prohibiting concurrent modification. This is the default locking mode.
- Last Write Wins Locking, which is a variation on Optimistic Locking. This allows concurrent access to a session by multiple threads in a single JVM or multiple JVMs. In this case, the last write is allowed to win.

- Member Locking, which allows concurrent access and modification of a session by multiple threads in the same JVM while prohibiting concurrent access by threads in different JVMs.
- 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 JVM.

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

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

Optimistic Locking

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

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

Last Write Wins Locking

Last Write Wins Locking mode is a variation on the Optimistic Locking mode. It allows multiple Web container threads in one or more JVMs 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 is allowed to modify the session.

Last Write Wins Locking mode can be configured by setting the `coherence-session-locking` parameter to `false`. This value will allow concurrent modification to sessions with the last update winning. If `coherence-session-app-locking`, `coherence-session-member-locking`, or `coherence-session-thread-locking` context parameter is set to `true`, this value is ignored (being logically `true`). Default is `false`.

Member Locking

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 JVMs. This is accomplished by acquiring a member-level lock for an HTTP session when the session is acquired. For more information on

member-level locks, see <lease-granularity> in the "distributed-scheme" section of the *Developer's Guide for Oracle Coherence*.

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

Application Locking

Application Locking mode restricts access (and modification) to a session 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 request. For more information on member-level locks, see <lease-granularity> in the "distributed-scheme" section of the *Developer's Guide for Oracle Coherence*.

Application Locking mode can be configured by setting the coherence-session-app-locking context parameter to true. Note that setting this to true will imply a setting of true for coherence-session-member-locking.

Thread Locking

Thread Locking mode restricts access (and modification) to a session to a single thread in a single JVM at a time. This is accomplished by acquiring both a member level, 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 on member-level locks, see <lease-granularity> in the "distributed-scheme" section of the *Developer's Guide for Oracle Coherence*.

Thread Locking mode can be configured by setting the coherence-session-thread-locking context parameter to true. Note that setting this to true implies a setting of true for both coherence-session-member-locking and coherence-session-app-locking.

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; the exception to this is when sticky load balancing is available and the Coherence*Web sticky session optimization is enabled. By default, threads that attempt to access a locked session (locked by a thread in a different JVM) block until the lock can be acquired. If you want to enable a timeout for lock acquisition, configure it with the tangosol.coherence.servlet.lock.timeout system property in the container's startup script (for example: -Dtangosol.coherence.servlet.lock.timeout=30s).

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

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

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 `logging-config`, in the "Operation Configuration Elements" appendix of the *Developer's Guide for 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

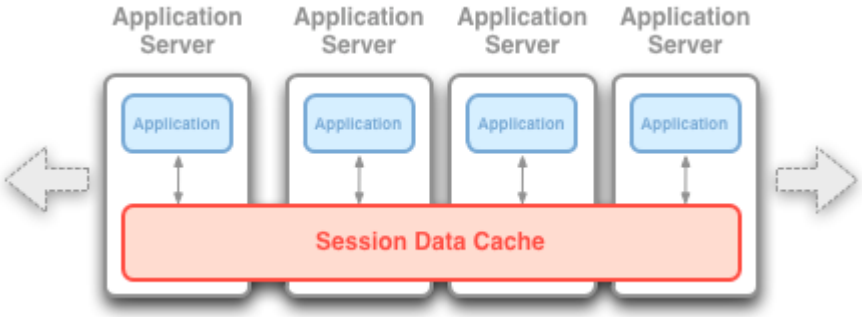
Coherence*Web supports most of the same deployment topologies that Coherence does including in-process, out-of-process (that is, client/server deployment), and bridging clients and servers over Coherence*Extend. The major supported deployment topologies are described in the following sections.

- In-Process, also known as "local storage enabled", is where session data is stored "in-process" with the application server
- Out-of-Process, 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, where communication between the application server tier and the cache server tier are over Coherence*Extend (TCP/IP)

In-Process

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, development and testing. In this topology, local storage is enabled (that is, `tangosol.coherence.distributed.localstorage=true`).

Figure 4–8 In-Process Deployment Topology



Out-of-Process

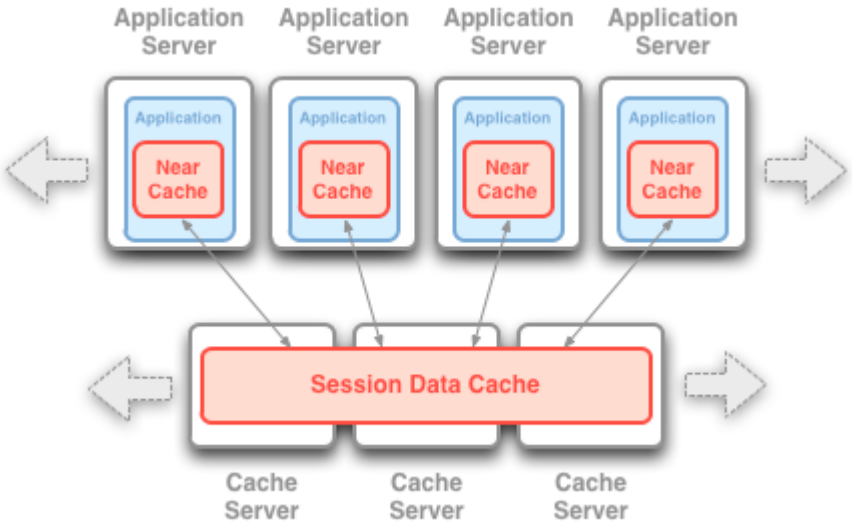
For the Out of Process deployment topology, the application servers (that is, application server tier) are configured as cache clients (that is, `tangosol.coherence.distributed.localstorage=false`) and there are dedicated JVMs running as cache servers, physically storing and managing the clustered data.

This approach has these benefits:

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

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

Figure 4–9 Out of Process Deployment Topology



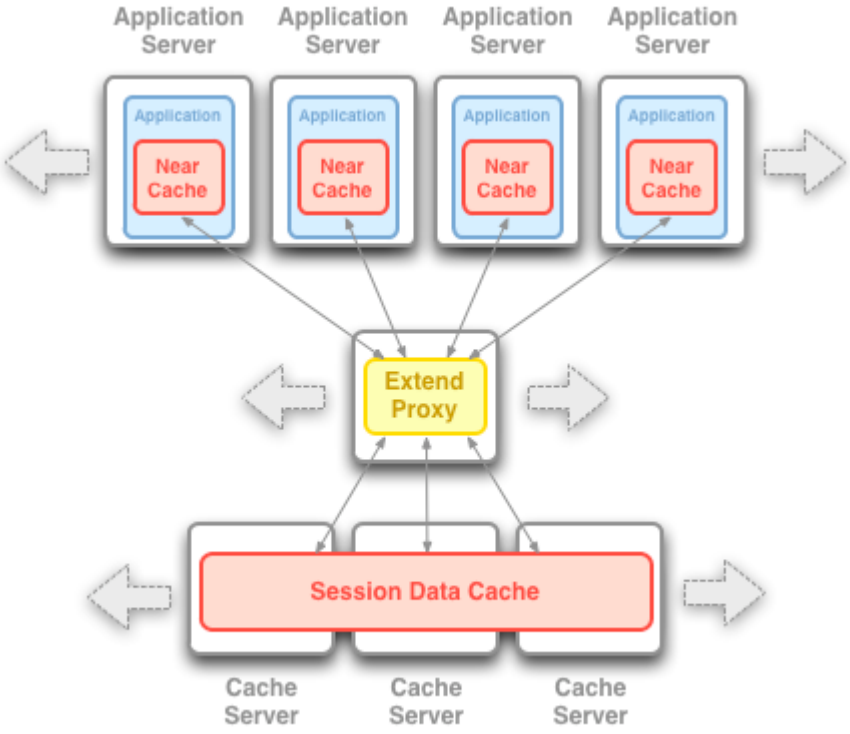
Out-of-Process with Coherence*Extend

The Out-of-Process with Coherence*Extend topology is similar to the Out-of-Process topology except that the communication between the application server tier and the

cache server tier are over Coherence*Extend (TCP/IP). For information on configuring this scenario, see "Configuring Coherence*Web with Coherence*Extend" on page 4-32.

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

Figure 4–10 Out-of-Process with Coherence*Extend Deployment Topology



Managing and Monitoring Applications with JMX

Note: To enable Coherence*Web JMX Management and Monitoring, you must set up the Coherence Clustered JMX Framework. See the configuration and installation instructions in *How to Manage Coherence with JMX* in the *Developer's Guide for Oracle Coherence*.

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

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

Table 4–1 Object Name for the HttpSessionManagerMBean

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

Table 4–2 describes the information that the `HttpSessionManagerMBean` 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`, which is typically, a larger and slower back-end cache that catches entries evicted from a faster front-end cache.

Table 4–2 Information Returned by the HttpSessionManagerMBean

Name	Data Type	Description
`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 4-30.
`CollectionClassName`	String	The fully qualified class name of the `HttpSessionCollection` implementation in use. The `HttpSessionCollection` interface is an abstract model for a collection of `HttpSessionModel` objects. The interface is not at all concerned with how the sessions are communicated between the clients and the servers.
`FactoryClassName`	String	The fully qualified class name of the `Factory` implementation in use. The `SessionHelper.Factory` is used by the `SessionHelper` to obtain objects that implement various important parts of the Servlet specification. It can be placed in front of the application in place of the application server's own objects, thus changing the "apparent implementation" of the application server itself (for example, adding clustering.)
`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 4-30.
`LocalAttributeCacheName`	String	The name of the local cache that stores non-distributed session attributes. If the attribute displays null then local session attribute storage is disabled.
`LocalAttributeCount`	Integer	The number of non-distributed session attributes stored in the local session attribute cache. If the attribute displays `-1`, then local session attribute storage is disabled.
`LocalSessionCacheName`	String	The name of the local cache that stores non-distributed sessions. If the attribute displays `null`, then local session storage is disabled.
`LocalSessionCount`	Integer	The number of non-distributed sessions stored in the local session cache. If the attribute displays `-1`, then local session storage is disabled.

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

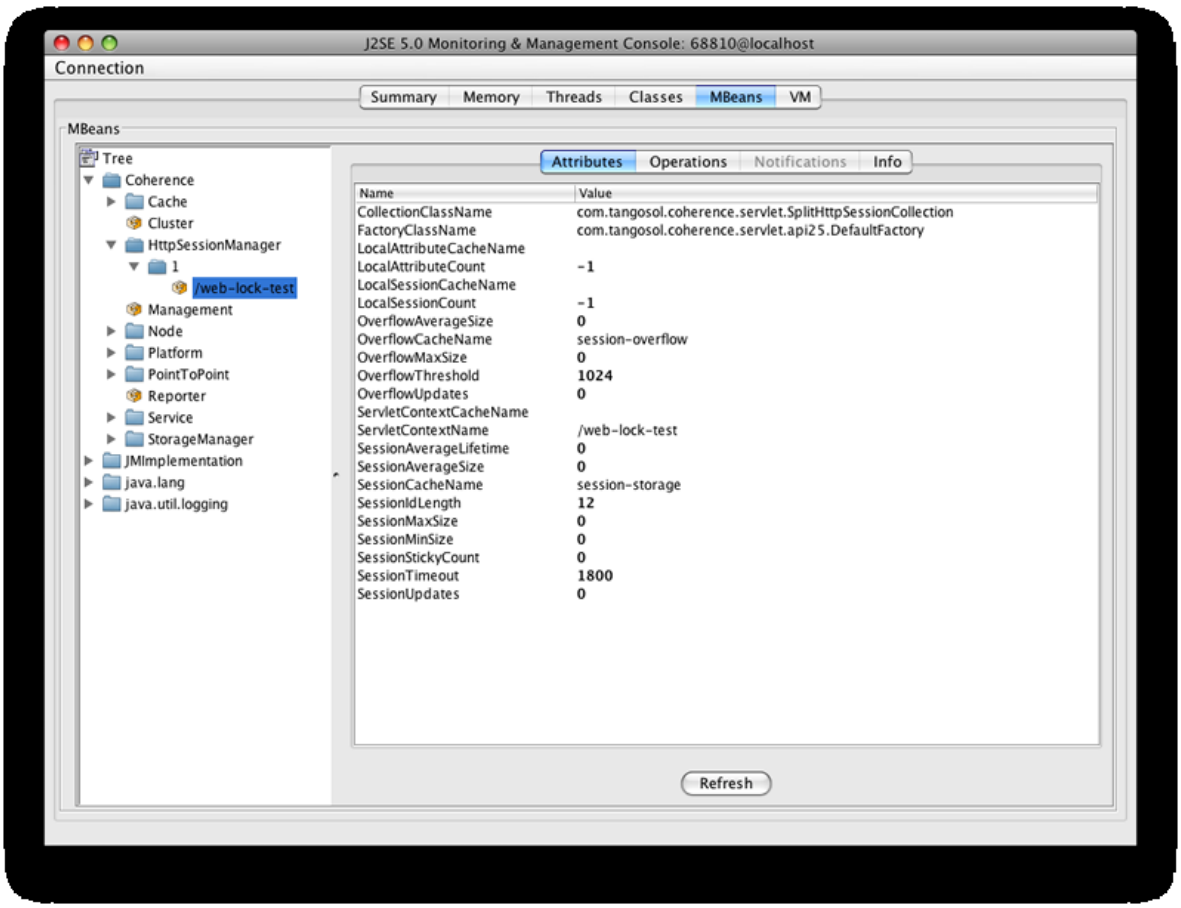
Name	Data Type	Description
`MaxReapedSessions`	long	The maximum number of sessions reaped in a reap cycle since the statistic was reset. See "Getting Session Reaper Performance Statistics" on page 4-30.
`NextReapCycle`	java.lang. Date	The time, expressed as a `java.lang.Date`, for the next reap cycle. See "Getting Session Reaper Performance Statistics" on page 4-30.
`OverflowAverageSize`	Integer	The average size (in bytes) of the session attributes stored in the "overflow" clustered cache since the last time statistics were reset. If the attribute displays `-1`, then a `SplitHttpSessionCollection` is not in use.
`OverflowCacheName`	String	The name of the clustered cache that stores the "large attributes" that exceed a certain size and thus are determined to be more efficiently managed as separate cache entries and not as part of the serialized session object itself. `Null` is displayed if a `SplitHttpSessionCollection` is not in use.
`OverflowMaxSize`	Integer	The maximum size (in bytes) of a session attribute stored in the "overflow" clustered cache since the last time statistics were reset. The attribute displays `-1` if a `SplitHttpSessionCollection` is not in use.
`OverflowThreshold`	Integer	The minimum length (in bytes) that the serialized form of an attribute value must be stored in the separate "overflow" cache that is reserved for large attributes. The attribute displays `-1` if a `SplitHttpSessionCollection` is not in use.
`OverflowUpdates`	Integer	The number of updates to session attributes stored in the "overflow" clustered cache since the last time statistics were reset. The attribute displays `-1` if a `SplitHttpSessionCollection` is not in use.
`ReapedSessions`	long	The number of sessions reaped during the last cycle. See "Getting Session Reaper Performance Statistics" on page 4-30.
`ReapedSessionsTotal`	long	The number of expired sessions that have been reaped since the statistic was reset. See "Getting Session Reaper Performance Statistics" on page 4-30.
`ServletContextCacheName`	String	The name of the clustered cache that stores `javax.servlet.ServletContext` attributes. The attribute displays `null` if the `ServletContext` is not clustered.
`ServletContextName`	String	The name of the Web application `ServletContext`.
`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 are pinned to this instance of the Web application. The attribute displays `-1` if sticky session optimizations are disabled.

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

Name	Data Type	Description
`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 4–11 illustrates the `HttpSessionManagerMBean` as it is displayed in the JConsole browser.

Figure 4–11 HttpSessionManagerMBean Displayed in the JConsole Browser



Running Performance Reports

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 on 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 on 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.
- Web Service Report, which records information on 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 4–3 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 4–3 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 `coherence.jar`.

> **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 *Managing Coherence* in the *Developer's Guide for Oracle Coherence*.

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 on the number of puts, gets, and prunes 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 post fixed with `-session-storage.txt`. For example `2010013113-session-storage.txt` would be created on January 31, 2010 1:00 pm. Table 4–3 describes the contents of the Web Session Storage report.

Table 4–3 Contents of the Web Session Storage Report

Column	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.

Table 4–3 (Cont.) Contents of the Web Session Storage Report

Column	Data Type	Description
`Cache Name`	String	This value is 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 executed.
`Report Time`	Date	The system time when the report executed.
`Tier`	String	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 executed.
`TotalGets`	long	The total number of session gets across the cluster since the last time the report was executed.
`TotalGetsMillis`	long	The total number of milliseconds spent per `get()` invocation (GetsMillis) to get the sessions across the cluster since the last time the report was executed.
`TotalHits`	long	The total number of session hits across the cluster since the last time the report was executed.
`TotalHitsMillis`	long	The total number of milliseconds spent per `get()` invocation that is a hit (HitsMillis) for the session storage across the cluster since the last time the report was executed.
`TotalMisses`	long	The total number of sessions gets that returned misses for the cache across the cluster since the last time the report was executed.
`TotalMissesMillis`	long	The total number of milliseconds spent per `get()` invocation that is a miss (MissesMillis) for the session storage across the cluster since the last time the report was executed.
`TotalPrunes`	long	The total number of times the session storage cache has been pruned across the cluster since the last time the report was executed.
`TotalPrunesMillis`	long	The total number of milliseconds spent for the prune operations (PrunesMillis) to prune the session storage cache across the cluster since the last time the report was executed.
`TotalPuts`	long	The total number of session updates (puts) across the cluster since the last time the report was executed.
`TotalPutsMillis`	long	The total number of milliseconds spent per `put()` invocation (PutsMillis) to update sessions across the cluster since the last time the report was executed.
`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 executed.

Table 4–3 (Cont.) Contents of the Web Session Storage Report

Column	Data Type	Description
TotalWritesMillis	long	The total number of milliseconds spent per write operation (WritesMillis) to update an external cache storage across the cluster since the last time the report was executed.

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 are stored. The statistics include information on the number of puts, gets, and prunes 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 post fixed with -cache-session-overflow.txt. For example 2010013113-cache-session-storage.txt would be created on January 31, 2010 1:00 pm. Table 4–4 describes the contents of the Web Session Overflow report.

Table 4–4 Contents of the Web Session Overflow Report

Column	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	The value is 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 executed.
Report Time	Date	The system time when the report executed.
Tier	String	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 executed.
TotalGets	long	The total number of session overflows gets across the cluster since the last time the report was executed.
TotalGetsMillis	long	The total number of milliseconds spent per get() invocation (GetsMillis) to get the session overflows across the cluster since the last time the report was executed.
TotalHits	long	The total number of session overflow hits across the cluster since the last time the report was executed.
TotalHitsMillis	long	The total number of milliseconds spent per get() invocation that is a hit (HitsMillis) for the session overflow across the cluster since the last time the report was executed.
TotalMisses	long	The total number of session overflow gets that returned misses for the cache across the cluster since the last time the report was executed.

Table 4–4 (Cont.) Contents of the Web Session Overflow Report

Column	Data Type	Description
TotalMissesMillis	long	The total number of milliseconds spent per get() invocation that is a miss (MissesMillis) for the session overflow across the cluster since the last time the report was executed.
TotalPrunes	long	The total number of times the session overflow cache has been pruned across the cluster since the last time the report was executed.
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 executed.
TotalPuts	long	The total number of session overflows (puts) across the cluster since the last time the report was executed.
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 executed.
TotalQueue	long	The sum of the queue link size for the session overflow cache across the cluster.
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 executed.
TotalWritesMillis	long	The total number of milliseconds spent per write operation (WritesMillis) to update an external session overflow storage across the cluster since the last time the report was executed.

Web Report

The Web Report 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 post-fixed with -web.txt. For example 2009013102-web.txt would be created on January 1, 2009 at 2:00 am. Table 4–5 describes the contents of the Web Report.

Table 4–5 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 executed.
Current Session Updates	long	The number of session updates since the last time the report was executed.
LocalAttributeCount	long	The Attribute count on the node.
LocalSessionCount	long	The session count on the node.

Table 4–5 (Cont.) Contents of the Web Report

Column	Data Type	Description
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 executed.
SessionAverageLifetime	float	The average number of seconds a session lives.
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 on 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 post-fixed with -web-session-service.txt. For example 2009013102-web-session-service.txt would be created on January 1, 2009 at 2:00 am. Table 4–6 describes the contents of the Web Service Report.

Table 4–6 Contents of the Web Service Report

Column	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 execution.
RequestPendingCount	Long	The number of pending requests by the Coherence*Web application at the time of the report.

Table 4–6 (Cont.) Contents of the Web Service Report

Column	Data Type	Description
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 execution.
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 execution.
Task Count	Long	The number of tasks executed by the Coherence*Web application since the last report execution.
Task Hung Count	Long	The number of tasks that hung by the Coherence*Web application since the last report execution.
Task Timeout Count	Long	The number of task timeouts by the Coherence*Web application since the last report execution.
Thread Abandoned Count	Long	The number of threads abandoned by the Coherence*Web application since the last report execution.

Cleaning Up Expired HTTP Sessions

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's own Garbage Collection (GC) capability: the Session Reaper is responsible for destroying any session that is no longer used, which is determined 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 timestamp 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. Since 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 co-located with the application server. Consequently, it is possible for the Session Reaper on each application server to only scan 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 co-located 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 only has to scan one time per application server per reaper cycle. The result is an out-of-the-box 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 is under high 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 `coherence-reaperdaemon-parallel` context parameter to `false`.

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. Since 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; since 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 out-of-the-box 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 list contains suggestions for tuning the out-of-the-box configuration of the Session Reaper:

- If the application is deployed with the in-process topology, then set the `coherence-reaperdaemon-assume-locality` configuration option to `true`.
- Since 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 ten 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 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`) 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 4–2 under "Managing and Monitoring Applications with JMX" on page 4-18.

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 *How to Manage Coherence with JMX* in the *Developer's Guide for Oracle Coherence*.

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. An implementation of the `SessionDistributionController` interface can mark sessions and/or attributes in either of the following ways:

- local, where a *local* session and/or attribute is stored on the originating server's heap, and thus, only accessible by that server

- distributed, where a *distributed* session and/or attribute is stored within the Coherence grid, and thus, accessible to other server JVMs

At any point during the life of a session, the session and/or attributes for that session can transition from local or distributed. However, when a session and/or attribute is distributed it cannot transition 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 on-line 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 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 non-distributed to distributed systems, especially when the cost of distributing all sessions and all attributes is a consideration.

Implementing a Session Distribution Controller

Example 4–4 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 to be present, then it is not distributed.

Example 4–4 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

Once you have written your `SessionDistributionController` implementation, you can register it with your application by using the `coherence-distributioncontroller-class` configuration parameter. Appendix A, "Coherence*Web Context Parameters" provides more information on these parameters.

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 and/or frequent garbage collection (GC) pauses.
- Web tier JVMs are restarted frequently.

In this type of deployment, there are three types of participants:

- Web tier JVMs, which are 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 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 returned through the TCP/IP connections.
- Storage JVMs, which are used to store the actual session data in memory.

These are the general steps to configure Coherence*Web to use Coherence*Extend:

1. Configure Coherence*Web to use the Optimistic Locking mode. See "Optimistic Locking" on page 4-14.
2. Create a cache configuration file for the proxy and storage JVMs. See "Configure the Cache for Proxy and Storage JVMs" on page 4-33.
3. Modify the Web tier cache configuration file to point to one or more of the proxy JVMs. See "Configuring the Cache for Web Tier JVMs" on page 4-36.

The following sections describe these steps in more detail.

Configure the Cache for Proxy and Storage JVMs

The session cache configuration file (`WEB-INF/classes/session-cache-config.xml`) is a Coherence*Web cache configuration file that can be used to configure the proxy and server JVMs for Coherence*Extend. The file contains system property overrides that allow it to be used for both proxy and storage JVMs. When used by a proxy JVM, specify the system properties described in Table 4–7:

Table 4–7 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 the proxy will bind to
`tangosol.coherence.session.proxy.localport`	a unique port number the proxy will bind to

When used by a cache server, specify the system properties described in Table 4–8:

Table 4–8 System Property Values for Storage JVMs

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

Example 4–5 illustrates the complete server-side session cache configuration file.

Example 4–5 session-cache-config-server.xml File

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!--                                                                                       -->
<!-- Server-side cache configuration descriptor for Coherence*Web over                     -->
<!-- Coherence*Extend (see session-cache-config-client.xml).                               -->
<!--                                                                                       -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<cache-config>
  <caching-scheme-mapping>
    <!--
    The clustered cache used to store Session management data.
    -->
    <cache-mapping>
      <cache-name>session-management</cache-name>

      <scheme-name>session-distributed</scheme-name>
    </cache-mapping>

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

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

    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <!--
    Distributed caching scheme used by the various Session caches.
    -->
    <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>
      </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>
      <serializer>
        <class-name>com.tangosol.io.DefaultSerializer</class-name>
      </serializer>
      <thread-count>0</thread-count>
      <lease-granularity>member</lease-granularity>
      <local-storage system-property="tangosol.coherence.session.
localstorage">true</local-storage>

      <partition-count>257</partition-count>
      <backup-count>1</backup-count>
      <backup-storage>
        <type>on-heap</type>
      </backup-storage>
      <backing-map-scheme>
        <local-scheme>

          <scheme-ref>unlimited-local</scheme-ref>
        </local-scheme>
      </backing-map-scheme>
      <autostart>true</autostart>
    </distributed-scheme>

    <!--
    Proxy scheme that Coherence*Web clients used to connect to the cluster.
    -->
    <proxy-scheme>

      <service-name>SessionProxy</service-name>
      <thread-count>10</thread-count>
      <acceptor-config>
        <serializer>
          <class-name>com.tangosol.io.DefaultSerializer</class-name>
        </serializer>
        <tcp-acceptor>

          <local-address>
            <address system-property="tangosol.coherence.session.proxy.
localhost">localhost</address>
            <port system-property="tangosol.coherence.session.proxy.
localport">9099</port>
            <reusable>true</reusable>
          </local-address>
        </tcp-acceptor>
      </acceptor-config>

      <autostart system-property="tangosol.coherence.session.
proxy">false</autostart>
    </proxy-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>
  </caching-schemes>

</cache-config>
```

Configuring the Cache for Web Tier JVMs

The `session-cache-config-client.xml` file illustrated in Example 4–6 is a client-side Coherence*Web cache configuration file that uses Coherence*Extend. This file should be used by the Web tier JVMs. Follow these steps to install and use this file:

1. Add proxy JVM hostnames/IP addresses and ports to the `<remote-addresses/>` section of the file. In most cases, you should include the hostname/IP address and port of all proxy JVMs for load balancing and failover.

   > **Note:** The `<remote-addresses>` element contains the proxy server(s) that the Web container will connect to. 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.

2. Rename the file to `session-cache-config-client.xml`.
3. 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.

Example 4–6 illustrates the complete client-side session cache configuration file.

Example 4–6 session-cache-config-client.xml File

```
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!--                                                                       -->
<!-- Client-side cache configuration descriptor for Coherence*Web over     -->
<!-- Coherence*Extend (see session-cache-config-server.xml).               -->
<!--                                                                       -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<cache-config>
  <caching-scheme-mapping>
    <!--
    The clustered cache used to store Session management data.
    -->
    <cache-mapping>
      <cache-name>session-management</cache-name>

      <scheme-name>session-near</scheme-name>
    </cache-mapping>

    <!--
    The clustered cache used to store ServletContext attributes.
    -->
```

```
  <cache-mapping>
    <cache-name>servletcontext-storage</cache-name>
    <scheme-name>session-near</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-remote</scheme-name>
  </cache-mapping>

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

  </cache-mapping>
</caching-scheme-mapping>

<caching-schemes>
  <!--
  Near caching scheme used by the Session attribute cache. The front cache
  uses a Local caching scheme and the back cache uses a Remote 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>
      <remote-cache-scheme>
        <scheme-ref>session-remote</scheme-ref>
      </remote-cache-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>

    <remote-cache-scheme>
      <scheme-name>session-remote</scheme-name>
      <initiator-config>
        <serializer>
          <class-name>com.tangosol.io.DefaultSerializer</class-name>

        </serializer>
        <tcp-initiator>
          <remote-addresses>
            <!--
            The following list of addresses should include the hostname and port
            of all running proxy JVMs. This is for both load balancing and
            failover of requests from the Web tier.
            -->
            <socket-address>
              <address>localhost</address>
              <port>9099</port>
            </socket-address>
          </remote-addresses>
        </tcp-initiator>
      </initiator-config>
    </remote-cache-scheme>
  </caching-schemes>
</cache-config>
```

Configuring Coherence*Web for JSF and MyFaces

Java Server Faces (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 Sun JSF 1.1 Reference Implementation or any other compatible implementation.

For all JSF and MyFaces Web-applications:

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

- If Coherence*Web reports an `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 enable the `coherence-preserve-attributes` context parameter.

The JSF parameter `javax.faces.STATE_SAVING_METHOD` identifies where the state of the view is stored between requests. By default, state is saved in the servlet session. Set the `STATE_SAVING_METHOD` parameter to `client` in the `context-param` stanza of `web.xml`, 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 4–7 illustrates setting the `STATE_SAVING_METHOD` parameter.

Example 4–7 Setting STATE_SAVING_METHOD in web.xml

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

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 may 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.x 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 `web.xml`. This parameter allows you to specify a custom servlet instead of the default `javax.faces.webapp.FacesServlet`.

  Example 4–8 illustrates setting the `DELEGATE_FACES_SERVLET` context parameter.

Example 4–8 Setting DELEGATE_FACES_SERVLET in web.xml

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

For Instrumented Applications that use the JSF Reference Implementation (Mojarra)

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

Example 4–9 Declaring the Faces Servlet in web.xml

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

For Non-instrumented Applications that use MyFaces and Coherence SPI

If you are using the Coherence SPI to deploy a Web-application built with MyFaces, then nothing more needs to be done. This is the recommended method of running MyFaces with Coherence*Web.

For Non-instrumented Applications that use the JSF Reference Implementation (Mojarra) and the Coherence SPI

If you are using the Coherence SPI to deploy a Web-application based on the JSF RI (Mojarra), then nothing needs to be done. This is the recommended method of running JSF with Coherence*Web.

5

Using Coherence*Web on WebLogic Portal

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

Using Coherence*Web with WebLogic Portal—Main Steps

Follow these steps to use Coherence*Web with WebLogic Portal.

1. Download Any Required Patch (optional)
2. Modify the Session Configuration (optional)
3. Start a Cache Server
4. Locate the Coherence JAR File
5. Enable a P13N (Personalization) Cache Provider (optional)
6. Reference the SPI in the Portal Application
7. Enable Coherence*Web Sessions
8. Create and Deploy the Application

Download Any Required Patch (optional)

> **Note:** WebLogic Server versions 10.3.1 or later do not require a patch. If you are using version 10.3.1 or later, you can skip this step.

If you are using WebLogic Server 10.3 or earlier, apply the WebLogic Server publicly available patch to all WebLogic Server instances that are hosting the Web applications that will use Coherence*Web. Table 5–1 lists the appropriate patches for your version of WebLogic Server and Coherence release level.

Table 5–1 Required WebLogic Server and Coherence Patch Release Levels

	WebLogic Server 9.2 MP1	WebLogic Server 10.3	WebLogic Server 10.3.1 and later
WebLogic Smart Update	Patch ID: AJQB	Patch ID: 6W2W	No patch required.

The patches can be downloaded by using either the MetaLink Web site or the WebLogic Server's Smart Update utility.

To Download from MetaLink:

1. Go to the Metalink Web site to manually locate the patch.

   `http://metalink.oracle.com/`

2. Select the **Patches** tab and click the **Simple Search** link. On the subsequent screen, submit a search for a **Patch Number/Name** with the appropriate value (for example, `11399293`).

3. Download the patch zip file from the displayed results.

4. See the `README.txt` included in the patch zip file for instructions for applying the Coherence patch.

To Download with Smart Update:

1. Review the instructions in the Smart Update Guide for using Smart Update to install WebLogic Server patches.

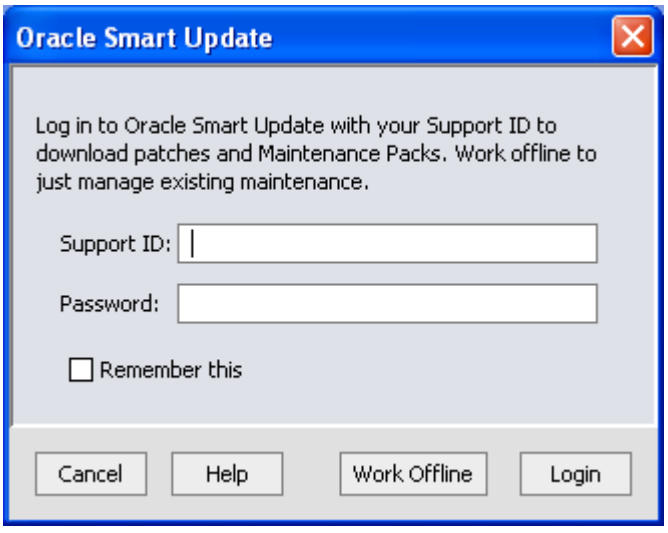
   For production environments, Oracle recommended that you review the Smart Update production installation.

   You can find the Smart Update Guide at this URL.

   `http://download.oracle.com/docs/cd/E14759_01/doc.32/e14143/toc.htm`

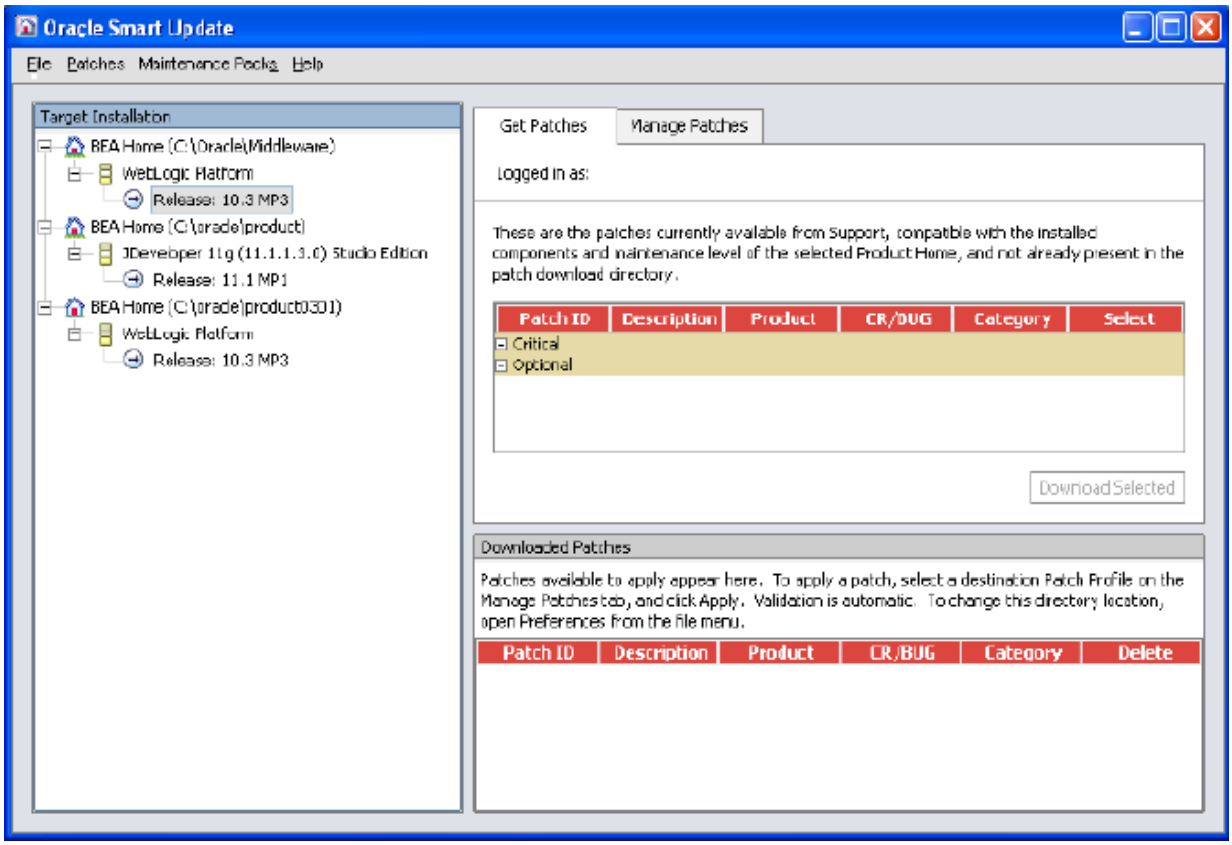
2. Select **Start** then **All Programs** then **Oracle WebLogic** then **Smart Update** to open the log-in dialog box. Use your Support ID and Password to log in.

Figure 5–1 Oracle Smart Update Login Dialog Box



3. Download and apply the appropriate patch for your version of the WebLogic Server. Figure 5–2 illustrates the Oracle Smart Update browser.

Figure 5–2 Oracle Smart Update Browser



Modify the Session Configuration (optional)

Modify the `session-cache-config.xml` file (if necessary) to customize the Cache topology for Coherence*Web.

This configuration file is located in the `/WEB-INF/classes` directory within the `coherence-web-spi.war` file. If you modify `session-cache-config.xml`, then it should be updated in the WAR file.

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

Start a Cache Server

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

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

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

```
localstorage=true com.tangosol.net.DefaultCacheServer
```

2. Start one or more Cache Server JVMs using the script described above.

Locate the Coherence JAR File

Copy the `coherence.jar` into the `APP-INF\lib` directory of the portal enterprise application.

Enable a P13N (Personalization) Cache Provider (optional)

If you want to use the Coherence P13N (Personalization) `CacheProvider`, then copy the `coherence-wlp.jar` into the `APP-INF\lib` of the portal enterprise application.

See the *Integration Guide for Oracle Coherence* for more information on the P13N CacheProvider SPI implementation and WSRP-federated portals.

If you want to use the Coherence P13N Cache as the **default** cache provider, add the following line before the first `<cache>` element to the `META-INF\p13n-cache-config.xml` file in the portal enterprise application:

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

Reference the SPI in the Portal Application

Reference `coherence-web-spi.war` by using a library-reference in the `WEB-INF\weblogic.xml` file in the portal Web application:

```
<wls:library-ref>
    <wls:library-name>coherence-web-spi</wls:library-name>
</wls:library-ref>
```

Enable Coherence*Web Sessions

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

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

Create and Deploy the Application

1. Create an EAR file for the test application using workshop's EAR deployment. This EAR will be deployed to the cluster for testing.
2. In the portal domain, first deploy `coherence-web-spi.war` as a library to the cluster.
3. In the portal domain, deploy the application EAR to the cluster.

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 Context Parameters for Coherence*Web

Parameter Name	Description
coherence-attribute-overflow-threshold	For the split model, this value specifies the minimum length (in bytes) that the serialized form of an attribute value must be for it to be stored in the separate "overflow" cache that is reserved for large attributes. If unspecified, defaults to `1024`.
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 4-12 for more information. If `false`, the Web application is responsible for shutting down the Coherence node (see `com.tangosol.net.CacheFactory.shutdown()`) according to its own considerations. You must carefully consider a cluster node scoping deployment model in this case and the circumstances under which the application shuts down the Coherence node and the side-effects of doing so. See "Cluster Node Isolation" on page 4-10 for more information on cluster node scoping. **Note:** When using the WebInstaller, a value of `true` instructs the WebInstaller to place the Coherence library in the `WEB-INF/lib` directory of *each* Web application found in your J2EE application. If unspecified, this parameter defaults to `false`.
coherence-configuration-consistency	If `true`, 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 members may exhibit inconsistent behavior in managing the session data. If unspecified, this parameter defaults to `false`.

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

Parameter Name	Description
coherence-contextless-session-retain-millis	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. Since 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 legal range is 10 to 10000 (from `1/100`th of a second up to `10` seconds). If unspecified, defaults to `200`.
coherence-distributioncontroller-class	This value specifies a class name of the `com.tangosol.coherence.servlet.HttpSessionCollection$SessionDistributionController` interface implementation to use. This feature requires `coherence-sticky-sessions` optimization to be enabled. Legal values include: ■ `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$DistributedController` ■ `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$HybridController` ■ `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$LocalController`
coherence-enable-sessioncontext (See Note 1)	When set to `true`, allows the application to iterate sessions from the session context, thus disobeying the deprecation in the servlet specification. If unspecified, defaults to `false`.
coherence-eventlisteners (See Note 1)	The comma-delimited list of names of application classes that want to receive events from the Web container. This list comes from the application listeners declared in the `listener` elements of `web.xml`.
coherence-enable-suspect-attributes	When set to `true`, an attempt is made to detect whether the value of any session-related attributes may have changed. Attributes that are mutable (determined with a simple check) and that can be accessed by a `get` are deemed to be suspect. Mutable objects may have been changed by application code and must be re-serialized back into the cache. If unspecified, defaults to `false`.
coherence-factory-class	The fully qualified class name of the `SessionHelper.Factory` to use. Defaults to `com.tangosol.coherence.servlet.api`*XX*`.DefaultFactory` where *XX* is `22`, `23`, `24`, or `25` for Servlet 2.2, 2.3, 2.4, or 2.5 containers respectively.
coherence-local-session-cachename	This name overrides the name of the local cache that stores non-distributed sessions when `coherence-distributioncontroller-class` parameter is specified. If unspecified, defaults to `local-session-storage`. 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 `coherence-sessiondistributioncontroller-class` parameter is specified or `coherence-preserve-attributes` parameter is `true`. If unspecified, 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 whether 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. If unspecified, 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 expiry. This value must be set to `false` if the session storage cache is being managed by nodes that are not running a reaper, for example if cache servers are being used to manage the session storage cache. 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 "Cleaning Up Expired HTTP Sessions" on page 4-27 for more information on the Session Reaper. If unspecified, defaults to `true`.
coherence-reaperdaemon-cluster-coordinated	When set to `true`, coordinates reaping in the cluster such that only one server will perform reaping within a given reaping cycle, and it will be responsible for checking all of the sessions that are being managed in the cluster. See "Cleaning Up Expired HTTP Sessions" on page 4-27 for more information on 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 4-28 for more information. If unspecified, defaults to `false`.
coherence-reaperdaemon-cycle-seconds	The number of seconds that the daemon rests between reaping. For production clusters with long session time-outs, this can safely be set higher. For testing, particularly with short session time-outs, it can be set much lower. Setting it too low can cause more network traffic and use more processing cycles, and only has benefit if the application requires the sessions to be invalidated quickly when they have expired. See "Cleaning Up Expired HTTP Sessions" on page 4-27 for more information on the Session Reaper. If unspecified, defaults to `300`.
coherence-reaperdaemon-parallel	When 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 4-28. The default is `true`.
coherence-reaperdaemon-priority	The priority for the Session Reaper daemon. For more information, see "Cleaning Up Expired HTTP Sessions" on page 4-27 and the source for the `java.lang.Thread` class. If unspecified, 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$AttributeScopeController` interface implementation to use. See "Session Attribute Scoping" on page 4-9 for more information. Legal values include: ▪ `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$SimpleScopeController` ▪ `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$ApplicationScopeController` ▪ `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$GlobalScopeController`
coherence-servletcontext-clustered (See Note 1)	Either `true` or `false` to indicate whether the attributes of the `ServletContext` will be clustered. If `true`, then all serializable `ServletContext` attribute values will be shared among all cluster nodes. If unspecified, defaults to `false`, primarily because the Servlet specification indicates that the `ServletContext` attributes are local to a JVM and should not be clustered.
coherence-servletcontext-cachename (See Note 1)	The name of the Coherence cache to be used to hold the servlet context data if the servlet context is clustered. If unspecified, defaults to `servletcontext-storage`. Appendix C, "Session Cache Configuration File" describes this parameter.
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, as application locking implies member locking. A value of `false` is incompatible with thread locking. If unspecified, defaults to `false`. See also: coherence-session-member-locking, coherence-session-locking, and coherence-session-thread-locking parameter descriptions and "Session Locking Modes" on page 4-13.
coherence-session-cachename	This name overrides the name of the clustered cache that stores the sessions. If unspecified, defaults to `session-storage`. Appendix C, "Session Cache Configuration File" describes this parameter.
coherence-session-cookie-domain (See Note 1)	The domain of the session cookie as defined by RFC 2109. By default, no domain is set explicitly by the session management implementation. See "Session and Session Attribute Scoping" on page 4-7 for more information.
coherence-session-cookie-name (See Note 1)	The name of the session cookie. If unspecified, defaults to `JSESSIONID`.
coherence-session-cookie-path (See Note 1)	The path of the session cookie as defined by RFC 2109. By default, no path is set explicitly by the session management implementation. See "Session and Session Attribute Scoping" on page 4-7 for more information.
coherence-session-cookie-max-age (See Note 1)	The maximum age in seconds of the session cookie as defined by RFC 2109. A value of -1 indicates that the cookie will not be persistent on the client; a positive value gives the maximum age that the cookie will be persisted by the client. Zero is not permitted. If unspecified, defaults to -1.
coherence-session-cookie-secure	Set to `true` to ensure that the session cookie will be sent only from a Web client over an SSL connection. If unspecified, the default is `false`.

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

Parameter Name	Description
coherence-session-cookies-enabled (See Note 1)	If unspecified, 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, defaults to `session-death-certificates`. Appendix C, "Session Cache Configuration File" describes this parameter.
coherence-session-expire-seconds	This value overrides the session expiry time, and is expressed in seconds. Setting it to -1 causes sessions to never expire. If unspecified, defaults to `1800`. See "Cleaning Up Expired HTTP Sessions" on page 4-27 for more information.
coherence-session-id-length (See Note 1)	This is the length, in characters, of generated session IDs. The suggested absolute minimum length is 8. If unspecified, defaults to `12`.
coherence-session-lazy-access	Enables lazy acquisition of sessions. A session will be acquired only when the servlet or filter attempts to access it. This is only relevant for instrumented Web applications—not when using the SPI. See "Accessing Sessions with Lazy Acquisition" on page 4-30. If unspecified, defaults to `false`.
coherence-session-locking	If `false`, concurrent modification to sessions, with the last update winning, 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 "Last Write Wins Locking" on page 4-14. If unspecified, defaults to `false`. See also: coherence-session-app-locking, coherence-session-member-locking, and coherence-session-thread-locking
coherence-session-log-threads-holding-lock	If `true`, specifies whether 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. If unspecified, defaults to `true`. See "Troubleshooting Locking in HTTP Sessions" on page 4-15 for more information.
coherence-session-management-cachename	This name overrides the name of the clustered cache that stores the management and configuration information for the session management implementation. Generally, it should be configured as a replicated cache. If unspecified, defaults to `session-management`. Appendix C, "Session Cache Configuration File" describes this parameter.
coherence-session-member-locking	This value, if set to `true`, prevents two threads in different JVMs from processing a request for the same session at the same time. See "Optimistic Locking" on page 4-14. If unspecified, defaults to `false`. See also: coherence-session-thread-locking, coherence-session-locking, and coherence-session-app-locking

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

Parameter Name	Description
coherence-session-overflow-cachename	For the split model, this value overrides the name of the clustered cache that stores the "large attributes" that exceed a certain size and thus are determined to be more efficiently managed as separate cache entries and not as part of the serialized session object itself. If unspecified, defaults to `session-overflow`. Appendix C, "Session Cache Configuration File" describes this parameter.
coherence-session-strict-spec	If the value is set to `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 the value is set to `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, defaults to `true`.
coherence-session-thread-locking	This value, if set to `true`, 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, as thread locking implies member locking. If unspecified, defaults to `true`. See also: coherence-session-app-locking, coherence-session-locking, and coherence-session-member-locking parameter descriptions and "Session Locking Modes" on page 4-13.
coherence-session-urldecode-bycontainer (See Note 1)	When set to `true`, uses the container's decoding of the URL session ID. If `coherence-session-urlencode-name` has been overridden, this must be set to `false`. Setting this to `false` will not work in some containers. If unspecified, defaults to `true`.
coherence-session-urlencode-bycontainer (See Note 1)	When set to `true`, uses the container's encoding of the URL session ID. Setting this to `true` may conflict with the setting for `coherence-session-urlencode-name` if it has been specified. If unspecified, defaults to `false`.
coherence-session-urlencode-enabled (See Note 1)	When set to true, enables URL encoding of session ids. If unspecified, defaults to `true`.
coherence-session-urlencode-name (See Note 1)	The parameter name to encode the session id into the URL with. On some containers, this value cannot be overridden. If unspecified, defaults to `jsessionid`.
coherence-session-weblogic-compatibility-mode	When set to `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 it is `false`, then the standard behavior will apply: that is, a single session ID will map to a single session instance using the Coherence*Web SPI in WebLogic. All other session persistence mechanisms in WebLogic use a single session ID in each web-app to refer to different session instances. If unspecified, 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.

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

Parameter Name	Description
coherence-sessioncollection-class	The fully qualified class name of the `HttpSessionCollection` implementation to use. Possible values include: ▪ `com.tangosol.coherence.servlet.MonolithicHttpSessionCollection` ▪ `com.tangosol.coherence.servlet.SplitHttpSessionCollection` (default) ▪ `com.tangosol.coherence.servlet.TraditionalHttpSessionCollection` A value must be specified for this parameter.
coherence-shutdown-delay-seconds	This value determines how long the session management implementation waits before shutting down after receiving the last indication that the application has been stopped, either from `ServletContextListener` events (Servlet 2.3 or later) or by the destruction of `Servlet` and `Filter` objects. This value is expressed in seconds. A value of zero indicates synchronous shut-down; any positive value indicates asynchronous shut-down. If unspecified, defaults to 0, because some servers are not capable of asynchronous shut-down.
coherence-sticky-sessions	This value, if set to `true`, specifies whether sticky sessions optimizations will be used. This should only be enabled if a sticky load balancer is being used. This feature requires member, application, or thread locking to be enabled. See "Enabling Sticky Session Optimizations" on page 4-16. See also coherence-session-thread-locking, coherence-session-member-locking, and coherence-session-app-locking. If unspecified, defaults to `false`.
coherence-web-sessions-enabled	Enables Coherence*Web sessions in WebLogic Portal applications. For more information, see "Enable Coherence*Web Sessions" on page 5-4.

Notes:

1. This parameter does not control Coherence*Web behavior when used with the WebLogic SPI implementation.

B

Capacity Planning

The objective of this appendix is to help you estimate the number of cache servers that your application will need. Keep in mind that the equations offered will only help you to arrive at a reasonable estimate. They do not account for the effects of cache indexes, non-application objects that may 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 severs 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. Since 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. As a rule-of-thumb, 66% is used as the maximum percentage of the heap available for storage, but you may find this figure too low for your system. Thus, 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

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

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

Value	Description
session-management	This cache is used to store internal configuration and management information for the session management implementation. This information is updated infrequently; therefore, it is a replicated cache by default.
servletcontext-storage	If `ServletContext` attribute clustering (see the `coherence-servletcontext-clustered` parameter in Table A–1) is enabled (it is disabled by default), this cache is used to store `ServletContext` attributes. This cache is replicated by default, as it is expected that there will a few read-mostly attributes.
session-storage	This cache is used to store session models. By default it is mapped to a near cache backed by a distributed cache since it is expected that a container will access and modify a subset of sessions multiple times (if sticky session load balancing is configured.) See "Session Models" on page 4-2 for more information.
session-overflow	If the `coherence-sessioncollection-class` parameter (described in Table A–1) is set to `com.tangosol.coherence.servlet.SplitHttpSessionCollection`, then this cache will hold "large" session attributes. By default, session attributes larger than 1K will be stored in this cache. This is configured as a distributed cache.
session-death-certificates	Recently expired session IDs are stored in this cache to prevent reuse of a recently used session ID. By default, each storage node will hold up to 4000 session IDs, and session IDs will be evicted after 24 hours. This is configured as a distributed cache.
local-session-storage	This local cache is used to store session models that are considered to be "local" by the configured (if any) `coherence-distributioncontroller-class` parameter. Table A–1 describes this parameter.
local-attribute-storage	This local cache is used to store attributes that are not distributed. This can happen under two conditions: ▪ A `coherence-distributioncontroller-class` is configured. Attributes for "local" sessions will be stored in this cache. ▪ A non-serializable attribute is set on a distributed session. If `coherence-preserve-attributes` is set to `true`, then non-serializable attributes will be placed in the cache. Table A–1 describes this parameter.

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

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

Value	Description
ReplicatedSessionsMisc	This replicated service is used by the `session-management` and `servletcontext-storage` caches.
DistributedSessions	This distributed service is used by the following caches: ■ `session-storage` ■ `session-overflow` ■ `session-death-certificates` The `tangosol.coherence.session.localstorage` system property controls whether a JVM 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 4-16 for more details.
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 `session-cache-config.xml` file. The cache- and services-related values described in the previous tables appear in **bold**.

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

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

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

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

    <!--
    The clustered cache used to store the "overflowing" (split-out due to size)
    Session attributes. Only used for the "Split" model.
```

```
  -->
  <cache-mapping>
    <cache-name>session-overflow</cache-name>
    <scheme-name>session-distributed</scheme-name>
  </cache-mapping>

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

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

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

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

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

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

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

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

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

  <!--
  "Base" Distributed caching scheme that defines common configuration.
  -->
  <distributed-scheme>
    <scheme-name>session-base</scheme-name>
    <service-name>DistributedSessions</service-name>
    <thread-count>0</thread-count>
    <lease-granularity>member</lease-granularity>
    <local-storage system-property="tangosol.coherence.session.
localstorage">false</local-storage>
    <partition-count>257</partition-count>
```

```
      <backup-count>1</backup-count>
      <backup-storage>
        <type>on-heap</type>
      </backup-storage>
      <backing-map-scheme>
        <local-scheme>
          <scheme-ref>unlimited-local</scheme-ref>
        </local-scheme>
      </backing-map-scheme>
      <request-timeout>30s</request-timeout>
      <autostart>true</autostart>
    </distributed-scheme>

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

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

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

Index

A

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

W