

Oracle® Application Server 10g

Best Practices

10g (9.0.4)

Part No. B12223-01

May 2004

Oracle Application Server 10g Best Practices, 10g (9.0.4)

Part No. B12223-01

Copyright © 2004 Oracle. All rights reserved.

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Contents

Send Us Your Comments	xi
Preface	xiii
Documentation Accessibility	xiii
Related Documents	xiii
Conventions	xiv
 1 Deployment	
1.1 Deployment Considerations.....	1-1
1.1.1 Security Requirements for Deployment	1-2
1.2 Infrastructure Deployment considerations	1-3
1.2.1 Infrastructure Data Tier Components	1-3
1.2.2 Infrastructure Midtier or Identity Management Server	1-4
1.3 Middle Tier Deployment Considerations.....	1-5
1.4 Use Remote Caching with Remote OracleAS Portal Instances	1-6
 2 Management and Monitoring	
2.1 Oracle Enterprise Manager Best Practices.....	2-1
2.1.1 Select the Framework Options That Best Suit Your Needs	2-2
2.1.2 Monitor and Diagnose Performance Bottlenecks and Availability Problems	2-2
2.1.3 Monitor Application Performance During Application Development or Test Cycles	2-2
2.1.4 Monitor Rate and Aggregated Performance Metrics	2-2
2.1.5 Diagnose Web Application Problems in OC4J	2-3
2.1.6 Monitor End-User Response Times of Web Pages	2-3
2.1.7 Monitor the Availability of a Web Application.....	2-3
2.1.8 Proactively Monitor Web Application Transactions	2-3
2.1.9 Tune Application SQL	2-4
2.1.10 Use the Host Home Page to Help Diagnose Performance Issues.....	2-4
2.1.11 Use Alerts and Notifications to Proactively Monitor System Availability	2-4
2.1.12 Change Configurations.....	2-5
2.1.13 Use Clusters for Application Deployment and Configuration Management.....	2-5
2.1.14 Use the Deployment Wizard to Deploy Applications.....	2-5
2.1.15 Use Job System to Schedule a Deployment	2-5
2.1.16 Use Job System to Periodically Back Up Your Configuration.....	2-5

2.1.17	Managing Both Oracle Application Server and the Oracle Database	2-5
2.2	Oracle Process Manager and Notification Server Best Practices	2-6
2.2.1	Starting and Stopping OPMN Server	2-6
2.2.2	Never Start or Stop OPMN Managed Components Manually	2-6
2.2.3	Review stdout and stderr Logs If A Component Does Not Start	2-6
2.2.4	Increase Timeout For Components That Take A Long Time To Start or Stop	2-6
2.2.5	Set Retry to High Values For Components Running on an Overloaded System	2-7
2.2.6	Leverage Additional Logging to Aid in Debugging	2-7
2.2.7	Start Order Dependencies	2-7
2.2.8	Event Scripts	2-7
2.2.9	Using OPMN to Manage External Components	2-8
2.3	Distributed Configuration Management Best Practices	2-8
2.3.1	Use Distributed Configuration Management Archiving	2-9
2.3.2	Specify a Single Instance in a Cluster as the Management Point	2-9
2.3.3	Do not Perform Concurrent Administration Operations	2-9
2.3.4	Do not Run updateConfig Concurrently with any Other Configuration Operation.	2-9
2.3.5	Restart Application Server Control after Joining or Leaving a Farm or Cluster in a File Based Repository	2-10
2.3.6	Use High Availability Features for Infrastructure Repository	2-10
2.3.7	dcmctl Usage	2-10
2.4	Dynamic Monitoring Services Best Practices	2-11
2.4.1	Monitor Your System Regularly	2-11
2.4.2	Take Regular Dumps of Metrics	2-11
2.4.3	Instrument Applications with DMS	2-12
2.4.4	Isolate Expensive Intervals Using PhaseEvent Metrics	2-12
2.4.5	Organize Performance Data	2-12
2.4.6	DMS Naming Conventions	2-12
2.4.7	DMS Coding Recommendations	2-12
2.4.8	Validate New Metrics	2-13

3 Platform Security and Identity Management

3.1	General Best Practices	3-1
3.1.1	Best Practices for HTTPS Use	3-1
3.1.2	Assign Lowest Level Privileges Adequate for the Task	3-2
3.1.3	Best Practices for Cookie Security	3-2
3.1.4	Best Practices in Systems Setup	3-3
3.1.5	Best Practices for Certificates Use	3-3
3.1.6	Review Code and Content Against Already Known Attack	3-4
3.1.7	Follow Common Sense Firewall Practices	3-4
3.1.8	Leverage Declarative Security	3-5
3.1.9	Use Switched Connections in DMZ	3-5
3.1.10	Place Application Server in the DMZ	3-5
3.1.11	Secure Sockets Layer	3-5
3.1.12	Tune the SSL SessionCacheTimeout Directive	3-6
3.1.13	Plan Out The Final Topology Before Installing Oracle Application Server Security Components	3-6
3.2	JAAS Best Practices	3-6

3.3	J2EE Security Best Practices.....	3-6
3.3.1	Avoid Writing Custom User Managers.....	3-6
3.3.2	Authentication Mechanism with the JAAS Provider	3-7
3.3.3	Use Fine-Grained Access Control.....	3-7
3.3.4	Use Oracle Internet Directory as the Central Repository	3-7
3.3.5	Develop Appropriate Logout Functionality for J2EE Applications	3-7
3.4	OracleAS Single Sign-On Best Practices	3-8
3.4.1	Configure for High Availability	3-8
3.4.2	Leverage Oracle Application Server Single Sign-On.....	3-8
3.4.3	Use an Enterprise-Wide Directory in Place	3-8
3.4.4	Use OracleAS Single Sign-On Instead of Writing Custom Authentication Logic.....	3-9
3.4.5	Always Use SSL with Oracle Application Server	3-9
3.4.6	Username and Password Only on Login Screen.....	3-9
3.4.7	Log Out So Cookies Do Not Remain Active	3-9
3.5	Oracle Internet Directory Deployment Best Practices	3-9
3.5.1	Use bulkload.sh Utility	3-10
3.5.2	Replicate for High Availability	3-10
3.5.3	Use SSL Binding.....	3-11
3.5.4	Use Backup and Restore Utilities	3-11
3.5.5	Monitoring and Auditing Oracle Internet Directory.....	3-11
3.5.6	Assign Oracle Internet Directory Privileges	3-12
3.5.7	Change Access Control Policies.....	3-12
3.5.8	Best Practice for Directory Integration Platform	3-12
3.5.8.1	Use Identity Management Realms	3-12
3.5.8.2	Configuring DIP Synchronization Service.....	3-13
3.5.8.3	Oracle HR Synchronization	3-13
3.5.9	Recommendations for Migrating Oracle9iAS Applications to an Existing Oracle Internet Directory	3-14
3.5.10	Configuration of the Self-Service Console	3-14
3.5.11	Use opmnctl instead of oidmon and oidctl.....	3-15
3.5.12	Configure Active Directory Synchronization	3-15
3.5.13	Use User Attributes and Password Hints for Resets	3-15

4 High Availability

4.1	Distribute Identity Management Components.....	4-1
4.2	Use OPMN for Crash Handling and Monitoring.....	4-2
4.3	Analyze High Availability using iHAT or Topology Viewer	4-3
4.4	Use Metric Based Load Balancing to Tune High Availability.....	4-4

5 Performance and Scalability

5.1	OracleAS Web Cache Best Practices.....	5-1
5.1.1	Improve Performance, Scalability, and Availability.....	5-1
5.1.2	Planning and Deployment	5-2
5.1.2.1	Use Two CPUs and Consider Deploying on Dedicated Hardware	5-2
5.1.2.2	Cluster Cache Instances for Better Availability, Scalability, and Performance...	5-2
5.1.2.3	Use a Network Load Balancer in Front of OracleAS Web Cache.....	5-3

5.1.2.4	Use OracleAS Web Cache Built-In Load Balancing for Availability and Scalability of Origin Servers	5-3
5.1.2.5	Deploy Caches in Remote Offices for Faster Response Times and Reduced WAN Traffic	5-4
5.1.2.6	Use the Latest Version	5-4
5.1.2.7	Test Application Upgrades and Patches to Ensure Existing Cache and Session Rules Still Function Correctly	5-4
5.1.3	OracleAS Web Cache Security	5-5
5.1.3.1	Route All HTTP and HTTPS Traffic Through OracleAS Web Cache	5-5
5.1.3.2	Secure Administration, Invalidation, and Statistics Monitoring Using HTTPS..	5-5
5.1.3.3	Use Web Caching to Help Defend Against Denial-of-Service Attacks	5-5
5.1.3.4	Change Passwords Frequently	5-6
5.1.4	Configuring OracleAS Web Cache.....	5-6
5.1.4.1	Use the OracleAS Web Cache Manager to Avoid Configuration Problems.....	5-6
5.1.4.2	Configure Enough Memory	5-7
5.1.4.3	Allocate Sufficient Network Bandwidth.....	5-7
5.1.4.4	Set a Reasonable Number of Network Connections	5-7
5.1.4.5	Create Custom Error Pages	5-7
5.1.5	Increasing Cache Hits.....	5-8
5.1.5.1	Use Cookies and URL Parameters to Increase Cache Hit Ratios	5-8
5.1.5.2	Use Redirection to Cache Entry Pages	5-9
5.1.5.3	Use Surrogate-Control Headers Instead of Caching Rules	5-9
5.1.5.4	Use Partial Page Caching Where Possible	5-10
5.1.5.5	Use ESI Variables for Improved Cache Hit Ratio for Personalized Pages.....	5-10
5.1.5.6	Use the <esi:environment> Tag for Authentication or Authorization Callbacks	5-11
5.1.5.7	Use esi:inline and esi:include Tags Appropriately	5-11
5.1.5.8	Leverage JESI Over Hand-Generating the ESI Tags.....	5-13
5.1.6	Invalidation and Expiration	5-13
5.1.6.1	Use Basic Invalidation for Single Objects.....	5-13
5.1.6.2	Use Substring Matching for Multiple Objects in Advanced Invalidations.....	5-14
5.1.6.3	Build Programmatic Invalidation Into Application Logic	5-15
5.1.6.4	Combine Invalidation and Expiration Policies	5-15
5.1.6.5	Use Invalidation Propagation in Clusters and Hierarchies.....	5-16
5.1.6.6	Tune Invalidation Performance Using Indexes	5-16
5.1.7	Optimizing Response Times	5-17
5.1.7.1	Optimize Response Time By Tuning Origin Server and OracleAS Web Cache Settings	5-17
5.1.7.2	Improve Response Times and Reduce Network Bandwidth With Compression.....	5-19
5.1.7.3	Use Only Warning or Notification Logging Levels to Conserve Resources.....	5-19

6 Oracle HTTP Server

6.1	Configure Appropriately For Modem Connections	6-1
6.2	Tune TCP/IP Parameters.....	6-1
6.3	Tune KeepAlive Directives.....	6-2
6.4	Tune MaxClients Directive	6-2
6.5	Avoid any DNS Lookup	6-2

6.6	Turn Off Access Logging	6-2
6.7	Use FollowSymLinks and Not SymLinkIfOwnerMatch	6-2
6.8	Set AllowOverride To None.....	6-3
6.9	Use mod_rewrite to Hide URL Changes for End Users.....	6-3
6.10	Sticky Routing at Load Balancer is not Required.....	6-3

7 J2EE Applications

7.1	Java Server Pages Best Practices	7-1
7.1.1	Pre-Translate JSPs Before Deployment.....	7-2
7.1.2	Separate Presentation Markup From Java	7-2
7.1.3	Use JSP Template Mechanism	7-2
7.1.4	Set Sessions=False If Not Using Sessions	7-2
7.1.5	Always Invalidate Sessions When No Longer Used	7-2
7.1.6	Set Main_Mode Attribute To "justrun"	7-3
7.1.7	Use Available JSP Tags In Tag Library.....	7-3
7.1.8	Minimize Context Switching Between Servlets and EJBs.....	7-3
7.1.9	Package JSP Files In EAR File For Deployment Rather Than Standalone.....	7-3
7.1.10	Use Compile-Time Object Introspection	7-3
7.1.11	Choose Static Versus Dynamic Includes Appropriately.....	7-3
7.1.12	Disable JSP Page Buffer If Not Used	7-4
7.1.13	Use Forwards Instead of Redirects	7-4
7.1.14	Use JSP Tagged Cache.....	7-4
7.1.15	Use well_known_taglib_loc To Share Tag Libraries	7-5
7.1.16	Use JSP-Timeout for Efficient Memory Utilization	7-5
7.1.17	Workarounds for the 64K Size Limit for the Generated Java Method.....	7-5
7.1.18	Workarounds for the Size Limit	7-6
7.1.19	Hiding JSP Pages.....	7-6
7.2	Sessions Best Practices.....	7-6
7.2.1	Persist Session State if Appropriate	7-6
7.2.2	Replicate Sessions if Persisting is Not an Option.....	7-7
7.2.3	Do Not Store Shared Resources in Sessions.....	7-7
7.2.4	Set Session Timeout Appropriately	7-8
7.2.5	Monitor Session Memory Usage	7-8
7.2.6	Always Use Islands, But Keep Island Size Small.....	7-8
7.2.7	Use a Mix of Cookie and Sessions	7-8
7.2.8	Use Coarse Objects Inside HTTP Sessions	7-8
7.2.9	Use Transient Data in Sessions Whenever Appropriate.....	7-9
7.2.10	Invalidate Sessions.....	7-9
7.2.11	Miscellaneous Guidelines.....	7-9
7.3	Enterprise Java Bean Best Practices.....	7-9
7.3.1	Local, Remote, and Message Driven EJBs.....	7-10
7.3.2	Use EJB Judiciously	7-10
7.3.3	Use Service Locator Pattern.....	7-10
7.3.4	Cluster Your EJBs.....	7-11
7.3.5	Index Secondary Finder Methods	7-11
7.3.6	Understand EJB Lifecycle	7-11
7.3.7	Use Deferred Database Constraints	7-11

7.3.8	Create a Cache with Read Only EJBs	7-12
7.3.9	Pick an Appropriate Locking Strategy	7-12
7.3.10	Understand and Leverage Patterns.....	7-12
7.3.11	When Using Entity Beans, Use Container Managed Aged Persistence Whenever Possible	7-12
7.3.12	Entity Beans using Local interfaces Only	7-13
7.3.13	Use a Session Bean Facade for Entity Beans	7-13
7.3.14	Enforce Primary Key Constraints at the Database Level.....	7-13
7.3.15	Use Foreign Key for 1-1 and 1-M Relationships.....	7-13
7.3.16	Avoid findAll Method on Entities Based on Large Tables	7-14
7.3.17	Set prefetch-size to Reduce Round Trips to Database.....	7-14
7.3.18	Use lazy-loading with Caution	7-14
7.3.19	Avoid Performing O-R Mapping Manually	7-14
7.4	Data Access Best Practices	7-15
7.4.1	Datasources Connections Caching and Handling	7-15
7.4.1.1	DataSource Connection Caching Strategies	7-16
7.4.2	Datasource Initialization.....	7-16
7.4.3	Disable Escape Processing for Better Performance.....	7-16
7.4.4	Defining Column Types.....	7-16
7.4.5	Prefetching Rows Improves Performance.....	7-17
7.4.6	Update Batching Improves Performance	7-17
7.4.6.1	Oracle Update Batching.....	7-17
7.4.6.2	Standard Update Batching	7-18
7.4.7	Use Emulated and Non-Emulated Data Sources Appropriately.....	7-19
7.4.8	Use the EJB-Aware Location Specified in Emulated Data Sources	7-19
7.4.9	Set the Maximum Open Connections in Data Sources.....	7-20
7.4.10	Set the Minimum Open Connections in Data Sources	7-20
7.4.11	Setting the Cache Connection Inactivity Timeout in Data Sources.....	7-21
7.4.12	Set the Wait for Free Connection Timeout in Data Sources	7-21
7.4.13	Set the Connection Retry Interval in Data Sources.....	7-21
7.4.14	Set the Maximum Number of Connection Attempts in Data Sources	7-22
7.4.15	Use JDBC Connection Pooling and Connection Caching	7-22
7.4.16	Use JDBC Statement Caching	7-22
7.4.17	Avoid Using More Than One Database Connection Simultaneously in the Same Request	7-23
7.4.18	Tune the Database and SQL Statements.....	7-23
7.4.18.1	JDBC Tuning	7-23
7.4.18.2	JDBC Connection Caching	7-24
7.4.18.3	JDBC Statement Caching.....	7-24
7.4.18.4	JDBC Cached Rowsets	7-24
7.5	J2EE Class Loading Best Practices	7-24
7.5.1	Avoid Duplicating Libraries	7-25
7.5.2	Load Resources Appropriately	7-25
7.5.3	Setting Class Loading Search Order within Web Modules	7-25
7.5.4	Declare and Group Dependencies.....	7-26
7.5.5	Minimize Visibility	7-26
7.5.6	Keep Configurations Portable.....	7-26
7.5.7	Do not Use the lib Directory for Container Wide Shared Libraries	7-26

7.6	Oracle Application Server TopLink Best Practices	7-26
7.6.1	OracleAS TopLink Mapping	7-27
7.6.2	Team Development	7-27
7.6.2.1	Team Working with Metadata.....	7-27
7.6.2.2	Large and/or Geographically Diverse Project Development	7-28
7.6.3	Caching.....	7-29
7.6.3.1	OracleAS TopLink Cache Refreshing Policies.....	7-30
7.6.3.2	Avoiding Stale Cache Content.....	7-31
7.6.3.3	Cache Synchronization	7-32
7.6.4	Sequencing.....	7-32
7.6.5	Performance Options	7-32
7.6.5.1	Performance Diagnostics.....	7-32
7.6.5.2	Tuning.....	7-33
7.7	Oracle Application Server XML Developer's Kit Best Practices	7-36
7.7.1	Choosing XML Parsers.....	7-36
7.7.2	High-Performance XSLT Transformations	7-37
7.7.3	Streaming XML Schema Validations	7-37
7.8	Java Message Service Best Practices	7-37
7.8.1	Set the Correct time_to_live Value	7-38
7.8.2	Do Not Grant Execute Privilege of the AQ PL/SQL Package to a User or Role	7-38
7.8.3	Close JMS Resources No Longer Needed	7-38
7.8.4	Reuse JMS Resources Whenever Possible.....	7-38
7.8.5	Use Debug Tracing to Track Down Problems	7-39
7.8.6	Understand Handle/Interpret JMS Thrown Exceptions	7-39
7.8.7	Ensure You Can Connect to the Server and Database From the Client Computer	7-39
7.8.8	Tune Your Database Based on Load	7-40
7.8.9	OJMS.....	7-40
7.8.10	OracleAS JMS Best Practices	7-41

8 Oracle Application Server Portal

8.1	Installation, Configuration, Administration, and Troubleshooting Best Practices	8-1
8.1.1	Use OracleAS RepCA.....	8-1
8.1.2	Use the Dependency Settings File and Tool	8-2
8.1.3	Configure the Diagnostic Log File for Improved Diagnostics	8-2
8.1.4	Review the <i>Oracle Application Server Portal Configuration Guide</i> for Installation and Configuration Troubleshooting Advice	8-2
8.2	OracleAS Portal Performance	8-2
8.2.1	Use Appropriate Caching Strategy	8-3
8.2.2	Use Web and Database Providers Judiciously	8-5
8.2.3	Improve Availability and Scalability	8-5
8.2.4	Scale OracleAS Portal by Tuning.....	8-5
8.2.5	mod_plsql Tuning Impacts Performance	8-6
8.2.6	Leverage Web Provider Session Caching.....	8-6
8.2.7	Increase Execution Speed of Slow Portlet	8-6
8.2.8	Reduce Page Complexity to Improve Cachability	8-6
8.2.9	Measure Tuning Effectiveness to Improve Performance.....	8-7
8.3	Performance Features for OracleAS Portal	8-7

8.3.1	Managed Portlet Execution per Page.....	8-7
8.3.2	Content Pruning.....	8-7
8.3.3	Search Key Invalidation.....	8-8
8.4	Content Management and Publishing	8-8
8.4.1	Use a Single Page Group for Delegating Administration.....	8-8
8.4.2	Research Your Taxonomy Before Building Up a Page Hierarchy	8-9
8.4.3	Use Page Templates for Consistency	8-9
8.4.4	Use Navigation Pages to Manage Template Content.....	8-9
8.4.5	Categories, Perspectives, and Custom Attributes.....	8-10
8.4.6	Understand how Multilingual Content is Managed	8-11
8.4.7	Use Unstructured User Interface Templates	8-11
8.4.8	Use Content Management APIs to Migrate Existing Content	8-12
8.4.9	Use WebDAV Capabilities to Support Desktop Application Centric Users.....	8-12
8.5	Export and Import Best Practices	8-13
8.5.1	Review Supported Use Cases Before Performing an Export or Import	8-13
8.5.2	Follow the Guidelines for Export and Import of Portal Objects.....	8-13

9 Oracle Application Server Wireless

9.1	Deploying Multiple Tiers for High-Volume Environments	9-1
9.2	Firewall Settings.....	9-1
9.3	Deploying Content Sources	9-2
9.4	Choice of Voice Gateway	9-2
9.5	Deploying Messaging Applications	9-2

10 Business Intelligence

10.1	Oracle Reports	10-1
10.1.1	Differences Between Paper and Web Reporting	10-1
10.1.2	Dynamic Environment Switching to Consolidate Reports Servers.....	10-2
10.2	Oracle Application Server Discoverer Best Practices	10-2

Index

Send Us Your Comments

Oracle Application Server 10g Best Practices, 10g (9.0.4)

Part No. B12223-01

Oracle welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: appserverdocs_us@oracle.com
- FAX: 650-506-7365 Attn: Oracle Application Server Documentation Manager
- Postal service:

Oracle Corporation
Oracle Application Server Documentation
500 Oracle Parkway, M/S 10p6
Redwood Shores, CA 94065
USA

If you would like a reply, please give your name, address, telephone number, and electronic mail address (optional).

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This preface includes the following topics:

- [Documentation Accessibility](#)
- [Related Documents](#)
- [Conventions](#)

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. 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. 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 additional information, visit the Oracle Accessibility Program Web site at

<http://www.oracle.com/accessibility/>

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, 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, JAWS 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.

Related Documents

For more information, see these Oracle resources:

- Oracle Application Server Documentation on Oracle Application Server Disk 1
- Oracle Application Server Documentation Library 10g (9.0.4)

Printed documentation is available for sale in the Oracle Store at

<http://oraclestore.oracle.com>

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at:

<http://otn.oracle.com/membership>

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at:

<http://otn.oracle.com/docs>

Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- [Conventions in Text](#)
- [Conventions in Code Examples](#)
- [Conventions for Microsoft Windows Operating Systems](#)

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an index-organized table .
<i>Italics</i>	Italic typeface indicates book titles or emphasis.	<i>Oracle9i Database Concepts</i> Ensure that the recovery catalog and target database do <i>not</i> reside on the same disk.
UPPERCASE monospace (fixed-width) font	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a NUMBER column. You can back up the database by using the BACKUP command. Query the TABLE_NAME column in the USER_TABLES data dictionary view. Use the DBMS_STATS.GENERATE_STATS procedure.
lowercase monospace (fixed-width) font	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter sqlplus to open SQL*Plus. The password is specified in the orapwd file. Back up the datafiles and control files in the /disk1/oracle/dbs directory. The department_id, department_name, and location_id columns are in the hr.departments table. Set the QUERY_REWRITE_ENABLED initialization parameter to true. Connect as oe user. The JRepUtil class implements these methods.

Convention	Meaning	Example
<i>lowercase</i> <i>italic</i> <i>monospace</i> (<i>fixed-width</i>) <i>font</i>	Lowercase italic monospace font represents placeholders or variables.	You can specify the <i>parallel_clause</i> . Run <i>Uold_release</i> .SQL where <i>old_release</i> refers to the release you installed prior to upgrading.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
[]	Brackets enclose one or more optional items. Do not enter the brackets.	DECIMAL (<i>digits</i> [, <i>precision</i>])
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	{ENABLE DISABLE}
	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	{ENABLE DISABLE} [COMPRESS NOCOMPRESS]
...	Horizontal ellipsis points indicate either: - That we have omitted parts of the code that are not directly related to the example - That you can repeat a portion of the code	CREATE TABLE ... AS <i>subquery</i> ; SELECT <i>col1</i> , <i>col2</i> , ... , <i>coln</i> FROM employees;
.	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	
Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	acctbal NUMBER(11,2); acct CONSTANT NUMBER(4) := 3;
<i>Italics</i>	Italicized text indicates placeholders or variables for which you must supply particular values.	CONNECT SYSTEM/ <i>system_password</i> DB_NAME = <i>database_name</i>
UPPERCASE	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	SELECT last_name, employee_id FROM employees; SELECT * FROM USER_TABLES; DROP TABLE hr.employees;

Convention	Meaning	Example
lowercase	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files. Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	<pre>SELECT last_name, employee_id FROM employees; sqlplus hr/hr CREATE USER mjones IDENTIFIED BY ty3MU9;</pre>

Conventions for Microsoft Windows Operating Systems

The following table describes conventions for Microsoft Windows operating systems and provides examples of their use.

Convention	Meaning	Example
Choose Start >	How to start a program.	To start the Oracle Database Configuration Assistant, choose Start > Programs > Oracle - <i>HOME_NAME</i> > Configuration and Migration Tools > Database Configuration Assistant.
File and directory names	File and directory names are not case sensitive. The following special characters are not allowed: left angle bracket (<), right angle bracket (>), colon (:), double quotation marks ("), slash (/), pipe (), and dash (-). The special character backslash (\) is treated as an element separator, even when it appears in quotes. If the file name begins with \\, then Windows assumes it uses the Universal Naming Convention.	c:\winnt "\"system32 is the same as C:\WINNT\SYSTEM32
C:\>	Represents the Windows command prompt of the current hard disk drive. The escape character in a command prompt is the caret (^). Your prompt reflects the subdirectory in which you are working. Referred to as the <i>command prompt</i> in this manual. The backslash (\) special character is sometimes required as an escape character for the double quotation mark (") special character at the Windows command prompt. Parentheses and the single quotation mark (') do not require an escape character. Refer to your Windows operating system documentation for more information on escape and special characters.	<pre>C:\oracle\oradata> C:\>exp scott/tiger TABLES=emp QUERY=\"WHERE job='SALESMAN' and sal<1600\" C:\>imp SYSTEM/password FROMUSER=scott TABLES=(emp, dept)</pre>
<i>HOME_NAME</i>	Represents the Oracle home name. The home name can be up to 16 alphanumeric characters. The only special character allowed in the home name is the underscore.	C:\> net start Oracle <i>HOME_NAME</i> TNSListener

Convention	Meaning	Example
<i>ORACLE_HOME</i> and <i>ORACLE_</i> <i>BASE</i>	In releases prior to Oracle8i release 8.1.3, when you installed Oracle components, all subdirectories were located under a top level <i>ORACLE_HOME</i> directory that by default used one of the following names: ■ C:\orant for Windows NT ■ C:\orawin95 for Windows 95 ■ C:\orawin98 for Windows 98 This release complies with Optimal Flexible Architecture (OFA) guidelines. All subdirectories are not under a top level <i>ORACLE_HOME</i> directory. There is a top level directory called <i>ORACLE_BASE</i> that by default is C:\oracle. If you install Oracle9i release 1 (9.0.1) on a computer with no other Oracle software installed, then the default setting for the first Oracle home directory is C:\oracle\ora90. The Oracle home directory is located directly under <i>ORACLE_BASE</i>. All directory path examples in this guide follow OFA conventions. Refer to <i>Oracle9i Database Getting Starting for Windows</i> for additional information about OFA compliances and for information about installing Oracle products in non-OFA compliant directories.	Go to the <i>ORACLE_BASE\ORACLE_HOME\rdbsms\admin</i> directory.

Deployment

This chapter describes deployment best practices for Oracle Application Server. It includes the following topics:

- [Section 1.1, "Deployment Considerations"](#)
- [Section 1.2, "Infrastructure Deployment considerations"](#)
- [Section 1.3, "Middle Tier Deployment Considerations"](#)
- [Section 1.4, "Use Remote Caching with Remote OracleAS Portal Instances"](#)

1.1 Deployment Considerations

In an enterprise environment, there is demand for highly available, scalable and secured enterprise architecture for global application deployment to support worldwide customers. Oracle Application Server 10g users expect high availability for:

- Middle-tier components such as OracleAS Web Cache, OracleAS Portal, OracleAS Wireless.
- Identity Management components such as OracleAS Single Sign-On, Delegated Administration Services, and Oracle Internet Directory.
- Custom J2EE applications.
- Backend OracleAS Metadata Repository.

Optimally, there should not be a single point of failure at any time, with any tier, and with any component.

The following is a description of a Portal and Wireless deployment in an enterprise environment.

The Portal and Wireless install type components require an OracleAS Infrastructure 10g. The infrastructure includes:

- Identity Management components including Oracle Internet Directory, Oracle Application Server Single Sign-On (OracleAS Single Sign-On), Oracle Delegated Administration Services (DAS), Oracle Directory Integration and Provisioning (DIP), and Oracle Application Server Certificate Authority (OCA).
- An Identity Management database consisting of an Oracle Application Server Metadata Repository (OracleAS Metadata Repository).

If there is a DMZ or multiple DMZs in the network topology, all the Application and Web tier components must reside in the Web tier DMZ. The data tier components reside in the intranet or data tier DMZ.

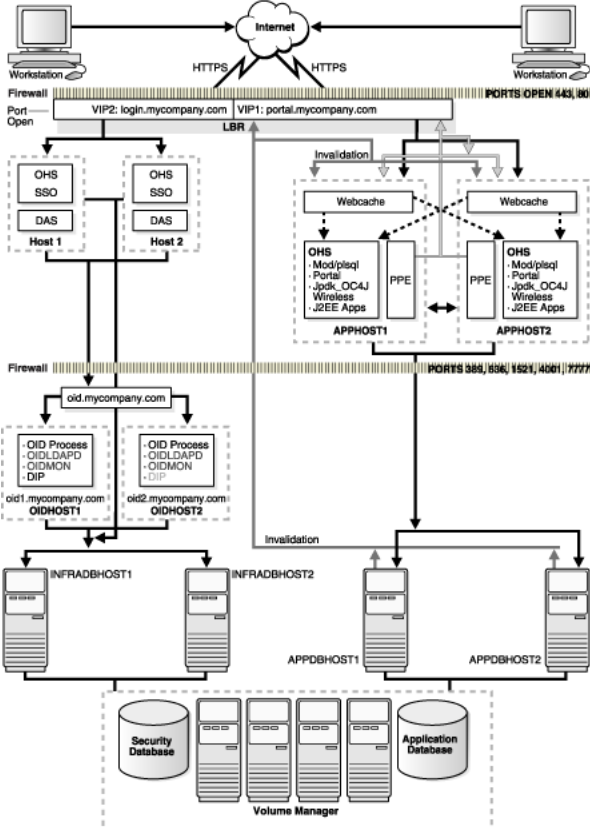
1.1.1 Security Requirements for Deployment

The following are security requirements for deployment:

- Distribute Oracle Application Server components between your Web and data tiers. Your Web application components should reside in the external DMZ and your database components should reside behind the internal or external DMZ. Components such as Oracle Internet Directory and DIP are considered as data tier components and they must reside in the data tier DMZ. Other Identity Management components such as Oracle HTTP Server, OracleAS Single Sign-On, and DAS will be in the Web tier DMZ.
- Communication between different components across the DMZ will be restricted for Port and protocol level, monitored by different firewall rules and intrusion detection.
- Isolate each DMZ as a zone and allow only required traffic/protocol between the DMZs on specific port.
- Do not allow any communication between two firewalls at a time.
- If communication starts in one firewall zone it must end in the next firewall zone.
- External direct communication from end-users must end at the external load-balancer (LBR) level.
- There should not any communication for the end user to the internal data tier (DMZ).

[Figure 1–1](#) is an example of component level High Availability architecture.

Figure 1–1 Security Deployment



1.2 Infrastructure Deployment considerations

This section describes infrastructure deployment considerations. It includes the following topics:

- Section 1.2.1, "Infrastructure Data Tier Components"
- Section 1.2.2, "Infrastructure Midtier or Identity Management Server"

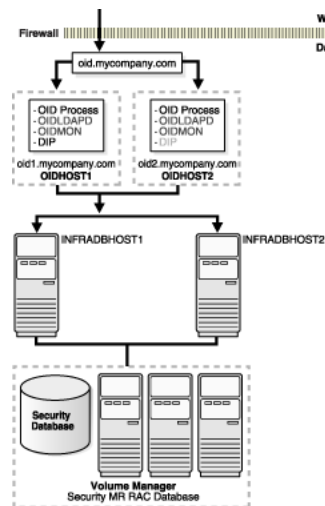
1.2.1 Infrastructure Data Tier Components

A highly available enterprise deployment requires end-to-end architecture for availability and scalability. The OracleAS Metadata Repository, Oracle Internet

Directory, and optionally, the DIP Server, are considered components of the data tier. You should deploy these components in the data tier DMZ.

Figure 1–2 is an example data tier infrastructure component deployment. In this example, the data tier High Availability is provided with Real Application Clusters (RAC), and multiple Oracle Internet Directory servers.

Figure 1–2 Data Tier Infrastructure



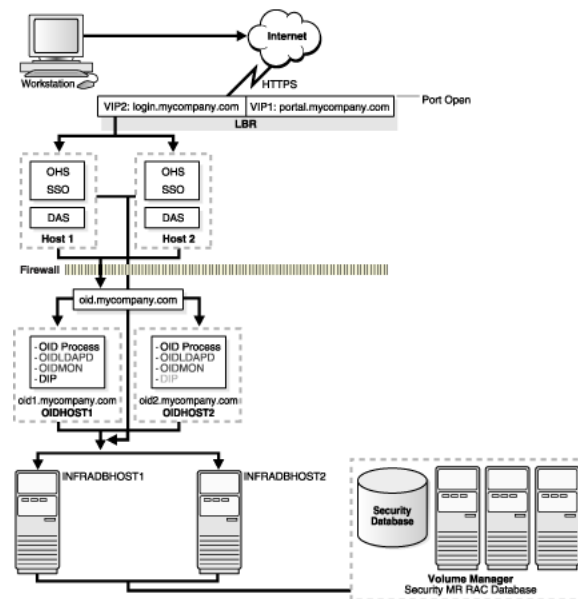
1.2.2 Infrastructure Midtier or Identity Management Server

Identity management components including Oracle HTTP Server, OracleAS Single Sign-On, and DAS should reside on the Web-tier or external DMZ.

For highly available deployment, you can have multiple identity management servers configured with same backend metadata repository. Identity Management servers can be configured with an external load balancer for external access.

If you have two Oracle Internet Directory servers running in your deployment, you can have a load balancer for LDAP traffic from OracleAS Single Sign-On and DAS to the two Oracle Internet Directories.

Figure 1–3 is an example of a highly available Identity Management installation.

Figure 1–3 High Availability Identity Management Installation

1.3 Middle Tier Deployment Considerations

Your application server middle tier should be installed and deployed in the Web tier or external DMZ. If your enterprise uses multiple middle-tier applications for high availability and scalability, you must deploy with a load balancer. When you use the load balancer consider the following:

- **Parallel Page Engine (PPE) Loopback:** PPE loops or routes back to OracleAS Web Cache to access `mod_plsql` for database connection. If OracleAS Web Cache is unresponsive, the PPE should be able to loopback to OracleAS Web Cache on the existing middle-tier. This is accomplished by looping back to LBR via Network Address Translations (NAT). For more information refer to the *Oracle Application Server Portal User's Guide*.
- **Invalidation:** Database sends invalidation message to OracleAS Web Cache, if there are multiple OracleAS Web Cache nodes Invalidation has to be load balanced across multiple OracleAS Web Cache nodes for High Availability. For more information refer to the *Oracle Application Server Portal User's Guide*.
- **SSL implementation:** Oracle AS will allow the customer to run their entire Application server in SSL mode using either software SSL or Hardware based SSL also called SSL accelerator (nCypher).

The following are characteristics of the end to end enterprise architecture for highly available, scalable and secured application server deployment:

- Highly available, scalable and secured architecture.
- Component separation to Web tier DMZ and data tier DMZ
- Encryption enforcement (SSL) for component communication outside the firewall. For ex: Communication from LBR to end-user or vice versa.
- SSL accelerator(nCypher) configuration with application server for performance improvement.
- No communication between two firewalls at a time.

Note: There may an exception for invalidation messages if there are multiple firewalls between LBR and database. But, this communication originates from data tier DMZ to Web tier DMZ in one direction, Reverse communication must be restricted.

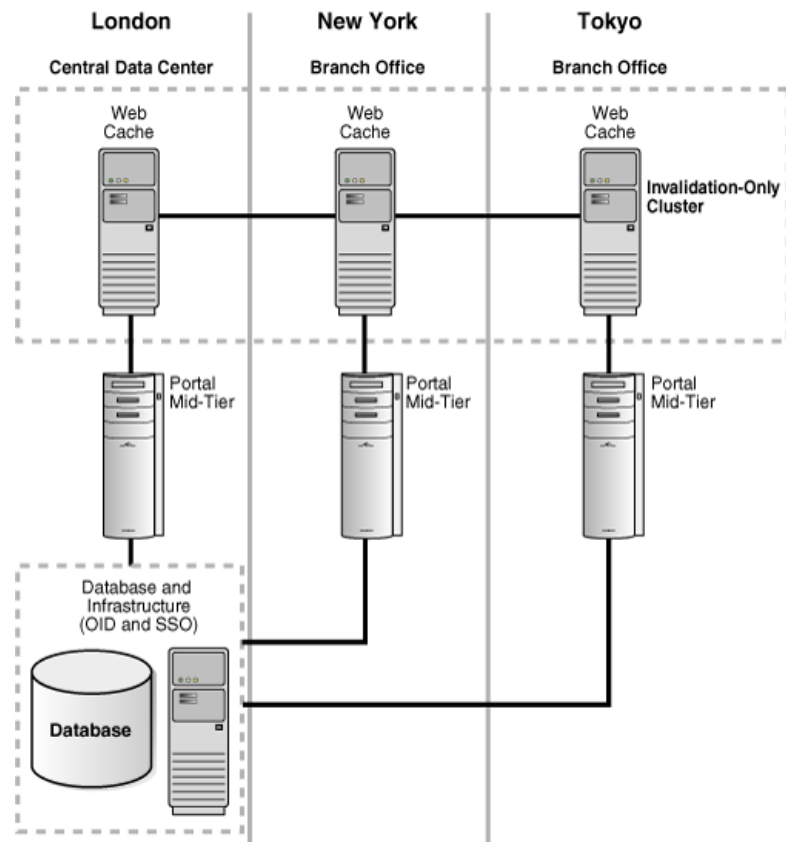
- Component level HA
- OracleAS Web Cache Clustering

1.4 Use Remote Caching with Remote OracleAS Portal Instances

If you want to deploy OracleAS Web Cache and OracleAS Portal in remote locations, consider deploying an instance of OracleAS Web Cache and an instance of the OracleAS Portal mid-tier at each remote location. This deployment can improve performance because many pages will be served from local caches and will not require any communication over the WAN.

In this deployment, each cache contacts the local OracleAS Portal mid-tier instance, instead of all caches contacting one OracleAS Portal instance at the data center. Each remote cache should cache content for the application Web servers. In addition, you can configure the remote caches as an invalidation-only cache cluster so that you can easily propagate invalidation across the caches in all locations.

[Figure 1–4](#) shows a topology that consists of OracleAS Web Cache and OracleAS Portal deployed in the middle tier at each location.

Figure 1–4 OracleAS Web Cache and OracleAS Portal Deployed at Remote Locations

Management and Monitoring

This chapter describes management and monitoring best practices for Oracle Application Server. It includes the following topics:

- [Section 2.1, "Oracle Enterprise Manager Best Practices"](#)
- [Section 2.2, "Oracle Process Manager and Notification Server Best Practices"](#)
- [Section 2.3, "Distributed Configuration Management Best Practices"](#)
- [Section 2.4, "Dynamic Monitoring Services Best Practices"](#)

2.1 Oracle Enterprise Manager Best Practices

This section describes best practices for Oracle Enterprise Manager. It features the following topics:

- [Section 2.1.1, "Select the Framework Options That Best Suit Your Needs"](#)
- [Section 2.1.2, "Monitor and Diagnose Performance Bottlenecks and Availability Problems"](#)
- [Section 2.1.3, "Monitor Application Performance During Application Development or Test Cycles"](#)
- [Section 2.1.4, "Monitor Rate and Aggregated Performance Metrics"](#)
- [Section 2.1.5, "Diagnose Web Application Problems in OC4J"](#)
- [Section 2.1.6, "Monitor End-User Response Times of Web Pages"](#)
- [Section 2.1.7, "Monitor the Availability of a Web Application"](#)
- [Section 2.1.8, "Proactively Monitor Web Application Transactions"](#)
- [Section 2.1.9, "Tune Application SQL"](#)
- [Section 2.1.10, "Use the Host Home Page to Help Diagnose Performance Issues"](#)
- [Section 2.1.11, "Use Alerts and Notifications to Proactively Monitor System Availability"](#)
- [Section 2.1.12, "Change Configurations"](#)
- [Section 2.1.13, "Use Clusters for Application Deployment and Configuration Management"](#)
- [Section 2.1.14, "Use the Deployment Wizard to Deploy Applications"](#)
- [Section 2.1.15, "Use Job System to Schedule a Deployment"](#)
- [Section 2.1.16, "Use Job System to Periodically Back Up Your Configuration"](#)

- [Section 2.1.17, "Managing Both Oracle Application Server and the Oracle Database"](#)

2.1.1 Select the Framework Options That Best Suit Your Needs

There are ways to deploy Oracle Enterprise Manager in order to give you the flexibility to select the configuration that best suits your needs. If you are working in a simple development or test environment, or if you have a single Oracle Application Server 10g instance to manage, you can use Oracle Enterprise Manager Application Server Control (Application Server Control), which is available with any Oracle Application Server middle-tier installation. Application Server Control allows you to directly access all the pages for managing and monitoring the instance.

In a production environment, you typically manage a wider variety of software and hardware components. For example, you need to manage the databases and host computers that support your Web applications. For your production environment, you should use Oracle Enterprise Manager Grid Control. The Grid Control Console provides you with a central location from which you can manage your Oracle Application Server instances, your databases, and your entire Oracle environment. Oracle Enterprise Manager Grid Control also supports sharing of information between administrators.

2.1.2 Monitor and Diagnose Performance Bottlenecks and Availability Problems

Once you have set up Oracle Enterprise Manager Grid Control to monitor for availability and performance issues, you will be alerted when a problem is detected. If Oracle Enterprise Manager detects that an application server component is unavailable, you can use Application Server Control to check the status of the component and restart it if desired. If a performance issue was detected, with a component or application, you can drill down to the component home page and view detailed performance and diagnostic information. You can also drill down from the Oracle Application Server Containers for J2EE (OC4J) home page to find applications, modules, and methods. Using these drill downs, you can diagnose and resolve performance issues.

2.1.3 Monitor Application Performance During Application Development or Test Cycles

During application development and testing, you can use the Application Server Control to monitor the application's resource usage and identify bottlenecks. For example, during a performance or load test you can view memory and CPU use for the Oracle Application Server instance overall and for the application. You can also drill down to find sessions, modules, EJBs, and methods that may be bottlenecks in the application.

2.1.4 Monitor Rate and Aggregated Performance Metrics

Application Server Control home pages and drill downs include rate and aggregated performance data that are not available via command line or other tools. For example, you can use Oracle Enterprise Manager to view average processing time for a HTTP request, allowing you to zero in on specific requests that may be slow.

Oracle Enterprise Manager also displays performance information, such as average processing time for a servlet for the most recent 5 minutes, in addition to averages since startup. This allows you to more easily diagnose problems in real-time.

2.1.5 Diagnose Web Application Problems in OC4J

Oracle Enterprise Manager provides comprehensive diagnostics that enable you to quickly pinpoint J2EE performance problems within the middle tier. To determine performance bottlenecks within your application, use the Web Application Page Performance page to identify the slowest URLs by OC4J processing time. Each URL is broken down by JSP, servlet, EJB and JDBC processing times. By traversing through the invocation paths of the URL call stack down to the SQL statement level, you can quickly identify the source of the bottlenecks causing application slowdowns. Use the application correlation feature to determine whether other system level problems have attributed to performance bottlenecks.

In addition, you can trace all invocation paths of a Web application starting at the transaction level on an on-demand basis, and diagnose performance problems across all tiers: from the network, through the middle tier (including JSP servlet, EJB, and JDBC times) down to the SQL statement level.

The mentioned features are enabled through configuration of Oracle Application Server through Oracle Enterprise Manager. Refer to the *Oracle Enterprise Manager Advanced Configuration Guide* for information on how to configure and enable this option.

2.1.6 Monitor End-User Response Times of Web Pages

To monitor the actual performance of your Web application as experienced by your end-users, use the End-User Performance Monitoring feature for Web applications. The Application Service Level Management feature of Oracle Enterprise Manager allows you to view and analyze the actual page response times for all URLs accessed by all your end-users. You can assess the impact of a performance problem on your end user base, or view page performance data by visitor, domain, region, or Web server, or by a combination of these axes. Also, you can highlight the monitoring of the most critical pages of your Web application by setting up a Watch List.

The End-User Performance Monitoring option requires configuration of the OracleAS Web Cache to instrument end-user performance data. Refer to the *Oracle Enterprise Manager Advanced Configuration Guide* for information on how to configure and enable this option.

2.1.7 Monitor the Availability of a Web Application

The definition of Web application availability varies depending on the application itself. With Oracle Enterprise Manager, you have the flexibility to define what constitutes the availability of your Web application. Web application availability is defined by a designated availability transaction and the key representative user communities of the application. Availability of a Web application is determined by the monitoring of the availability transaction from various user communities (also known as beacons) at specified intervals. Alerts are generated to immediately inform you of when your Web application is considered down. For more information, refer to the *Oracle Enterprise Manager Advanced Configuration Guide*.

2.1.8 Proactively Monitor Web Application Transactions

Oracle Enterprise Manager provides a proactive approach to monitoring Web applications through transaction performance monitoring. Synthetic business transactions are created using the transaction recorder, and are then replayed and monitored at specified intervals from key representative user communities called beacons. Measure the response times of key business transactions from various

geographical user communities using this feature. Use transaction performance monitoring to:

- isolate server-side problems from network delays
- profile how much time is spent connecting to the server
- document its first byte time
- time spent serving HTML and non-HTML content

Alerts will notify you when transaction response time thresholds have been exceeded. Refer to the *Oracle Enterprise Manager Advanced Configuration Guide* on how to configure and enable this option.

2.1.9 Tune Application SQL

Applications that access the database using SQL can be tuned using the Oracle Tuning Pack. Oracle Tuning Pack offers a cost-effective and easy-to-use solution that automates the entire application tuning process. Automatic SQL tuning is exposed through two new Oracle 10g Database components: SQL Tuning Advisor and SQL Access Advisor. Both components are seamlessly integrated with Oracle Enterprise Manager Grid Control and Database Control.

The SQL Tuning Advisor takes one or more SQL statements as input and applies the automatic SQL tuning process on it. The output of the SQL Tuning Advisor is in the form of recommendations, along with a rationale for each recommendation and its expected benefit.

The SQL Access Advisor provides comprehensive advice on how to optimize schema design in order to maximize application performance.

These two SQL advisors automate all manual tuning techniques currently practiced and form the core of automatic tuning solution.

2.1.10 Use the Host Home Page to Help Diagnose Performance Issues

The Application Server Control home page not only displays critical performance data and resource usage for the application server instance, it also includes a link to information for the host. For example, if your application server is performing poorly you can first drill down to the related Host home page to determine if the underlying problem is due to resource problems with the host and other processes, or to services running on the computer.

2.1.11 Use Alerts and Notifications to Proactively Monitor System Availability

Oracle Enterprise Manager Grid Control allows you to monitor your systems for specific conditions, such as loss of service or poor performance. When such a condition exists, Oracle Enterprise Manager generates an alert, which displays automatically on the appropriate Oracle Enterprise Manager home pages. In addition, you can also be notified via email or Web page. Minimally, you should set up Oracle Enterprise Manager Grid Control to alert you when your critical or production application servers are unavailable.

You can also configure Grid Control to notify specific administrators when an event condition occurs. This simplifies cooperation between administrators who share responsibility for the same systems.

2.1.12 Change Configurations

When you edit the configuration of Oracle Application Server components including Oracle HTTP Server, OC4J, or OPMN, you should do so using Application Server Control. Oracle Enterprise Manager ensures that your configuration changes are updated in the repository. If you edit these configuration files manually, you must use the DCM command-line utility, `dcmctl`, to notify the DCM repository of the changes.

2.1.13 Use Clusters for Application Deployment and Configuration Management

Using Oracle Application Server clusters simplifies management and maintenance of your application servers. Clustering enforces consistent configurations across all members of the cluster. If you want to make a configuration change in every instance, you only need to make the change once. The clustering mechanism ensures that the new configuration is propagated to all members.

Similarly, clustering also enforces consistency of deployed applications across all application server instances. If you wish to deploy a new application or update an existing deployment on every application server instance in the cluster, you only need to deploy or update the application once. The clustering mechanism ensures that the application is properly deployed to all members.

2.1.14 Use the Deployment Wizard to Deploy Applications

A simple way to deploy an application is to use the Oracle Enterprise Manager deployment wizard, which can be accessed from the Application Server Control. The wizard walks you systematically through all the essential deployment options to ensure that your application is deployed correctly.

2.1.15 Use Job System to Schedule a Deployment

In some cases, you may want to deploy an application during off-hours or at a certain scheduled time. You can use the Oracle Enterprise Manager job system to schedule a deployment to occur at a selected time. Simply create a script containing the DCM command-line `dcmctl deploy application` command and schedule the script via the Oracle Enterprise Manager job system.

2.1.16 Use Job System to Periodically Back Up Your Configuration

Periodically you should back up your application server configuration. By saving your configurations, you can restore the backed up settings if you ever need to undo configuration changes made. You can use the DCM command-line utility's `dcmctl createArchive instance` command in a script to save the configuration and application information for an application server instance. You can then schedule the backup script to run periodically using the Oracle Enterprise Manager job system. This ensures that backups of your configurations are taken on a regular basis.

2.1.17 Managing Both Oracle Application Server and the Oracle Database

If you plan to manage both your Oracle Application Server instances and your Oracle database from the same management console, install the latest version of Oracle Enterprise Manager 10g Grid Control. This will ensure that you have the most up-to-date functionality for managing both types of targets.

2.2 Oracle Process Manager and Notification Server Best Practices

This section describes Oracle Process Manager and Notification (OPMN) Server best practices. It includes the following topics:

- [Section 2.2.1, "Starting and Stopping OPMN Server"](#)
- [Section 2.2.2, "Never Start or Stop OPMN Managed Components Manually"](#)
- [Section 2.2.3, "Review stdout and stderr Logs If A Component Does Not Start"](#)
- [Section 2.2.4, "Increase Timeout For Components That Take A Long Time To Start or Stop"](#)
- [Section 2.2.5, "Set Retry to High Values For Components Running on an Overloaded System"](#)
- [Section 2.2.6, "Leverage Additional Logging to Aid in Debugging"](#)
- [Section 2.2.7, "Start Order Dependencies"](#)
- [Section 2.2.8, "Event Scripts"](#)
- [Section 2.2.9, "Using OPMN to Manage External Components"](#)

2.2.1 Starting and Stopping OPMN Server

The OPMN server should be started as soon as possible after turning on the host. OPMN must be running whenever OPMN-managed components are turned on or off. OPMN must be the last service turned off whenever you reboot or turn off your computer.

2.2.2 Never Start or Stop OPMN Managed Components Manually

Oracle Application Server components managed by OPMN should never be started or stopped manually. Do not use command line scripts or utilities from previous versions of Oracle Application Server for starting and stopping Oracle Application Server components. Use the Application Server Control or the `opmnctl` command line utility to start or stop Oracle Application Server components.

2.2.3 Review stdout and stderr Logs If A Component Does Not Start

The standard output (`stdout`) and standard error (`stderr`) of OPMN managed processes are reported in the log file in available in the `ORACLE_HOME/opmn/logs` directory. OPMN creates a log file for each component and assigns a unique concatenation of the Oracle Application Server component with a number. For example, the standard output log for OracleAS Web Cache may be `WebCache~WebCacheAdmin~1`. The process specific console logs are the first and best resource for investigating problems related to starting and stopping components.

The `stdout` and `stderr` log files are reused and appended to when a component is restarted so these files can contain output from multiple invocations of a component.

2.2.4 Increase Timeout For Components That Take A Long Time To Start or Stop

The time it takes to execute an `opmnctl` command is dependent on the type of Oracle Application Server process and available computer hardware. Because of this the time it takes to execute an `opmnctl` command may not be readily apparent. For example, the default start time out for OC4J is approximately five minutes. If an OC4J process does not start-up after an `opmnctl` command, OPMN will wait approximately an hour before timing out and aborting the request.

Increase the start element timeout attribute for the component that takes a long time to start. Set the timeout in the opmn.xml file at a level that will allow OPMN to wait for process to come up. This functionality is also available with the startproc command. Similarly increase the stop element timeout attribute in opmn.xml for the component that takes a long time to stop.

2.2.5 Set Retry to High Values For Components Running on an Overloaded System

Pings occur periodically between OPMN and the components that it manages to ensure that each component is not unresponsive and is capable of servicing requests. Ping failures result in a certain number of retry attempts and multiple failures in a row result in a restart of the component. On overloaded systems, it may be necessary to increase the number of retry attempts made before restarting the component.

Both `<start>` and `<restart>` elements, for each OPMN managed component in the `ORACLE_HOME/opmn/conf/opmn.xml` configuration file, accept an attribute named `retry`, which is the number of times to retry a ping attempt before a component is considered hung. Reasonable default retry values have been chosen, but when necessary, explicitly set this attribute in the appropriate element to a value greater than what is needed for the component to be pinged successfully.

2.2.6 Leverage Additional Logging to Aid in Debugging

OPMN provides different levels of logging. In a typical production mode, the log level should be set to a minimum. However, the following are steps should be performed, prior to contacting technical support, when having a problem related to OPMN:

- Set both log levels, the level attribute of `<log-file>` element for both `<notification-server>` and `<process-manager>` elements in the opmn.xml file, to 8 or 9.
- Execute the `$ORACLE_HOME/opmn/bin/opmnctl debug` command and save the output to a file.
- Save a copy of all logs in the `$ORACLE_HOME/opmn/logs` directory.

The file at this log level contains valuable information to assist in debugging.

2.2.7 Start Order Dependencies

OPMN is configured at installation with default start order dependencies, which allows you to start all of the components in an instance in a specific order with a single command. But if a specific component requires that other components and services are up and running before it starts, you can configure additional dependencies according to the environment.

2.2.8 Event Scripts

You can configure OPMN to execute your own custom event scripts whenever a particular component starts, stops, or crashes. You will find it useful to use one or more of the following event types:

- **pre-start:** OPMN runs the pre-start script after any configured dependency checks have been performed and passed, and before the Oracle Application Server component starts. For example, the pre-start script can be used for site-specific initialization of external components.

- **pre-stop:** OPMN runs the pre-stop script before stopping a designated Oracle Application Server component. For example, the pre-stop script can be used for collecting Java Virtual Machine stack traces prior to stopping OC4J processes.
- **post-crash:** OPMN runs the post-crash script after the Oracle Application Server component has terminated unexpectedly. For example, a user could learn of component crashes by supplying a script or program to be executed at post-crash events which sends a notification to the administrator's pager."

Refer to the *Oracle Process Manager and Notification Server Administrator's Guide* for a sample pre-start event script.

2.2.9 Using OPMN to Manage External Components

OPMN has the ability to manage arbitrary daemon processes that are not part of your Oracle Application Server installation. Even more sophisticated process management services can be created by supplying the `opmn.xml` file the optional paths to scripts for stopping, restarting, and pinging the daemon process.

Here is a simple example of an `opmn.xml` configuration for a custom component. The following lines load and identify the custom process module:

```
<module path="%ORACLE_HOME%/opmn/lib/libopmncustom.so">
  <module-id id="CUSTOM" />
</module>
```

The following lines represent the minimum configuration for a custom process:

```
<ias-component id="Custom">
  <process-type id="Custom" module-id="CUSTOM">
    <process-set id="Custom" numprocs="1">
      <module-data>
        <category id="start-parameters">
          <data id="start-executable" value="Your start executable here" />
        </category>
      </module-data>
    </process-set>
  </process-type>
</ias-component>
```

For complete configuration details refer *Oracle Process Manager and Notification Server Administrator's Guide*.

2.3 Distributed Configuration Management Best Practices

This section describes best practices for Distributed Configuration Management (DCM). It contains the following topics:

- [Section 2.3.1, "Use Distributed Configuration Management Archiving"](#)
- [Section 2.3.2, "Specify a Single Instance in a Cluster as the Management Point"](#)
- [Section 2.3.3, "Do not Perform Concurrent Administration Operations"](#)
- [Section 2.3.4, "Do not Run updateConfig Concurrently with any Other Configuration Operation"](#)
- [Section 2.3.5, "Restart Application Server Control after Joining or Leaving a Farm or Cluster in a File Based Repository"](#)
- [Section 2.3.6, "Use High Availability Features for Infrastructure Repository"](#)

- [Section 2.3.7, "dcmctl Usage"](#)

2.3.1 Use Distributed Configuration Management Archiving

The DCM archive feature provides a convenient and easy means of managing snapshots of the DCM managed portions of Oracle Application Server system configuration. Archives are useful for staging changes, recovering from errors, and to provision DCM managed configuration information associated with one Oracle Application Server instance to another.

DCM managed system configuration includes configuration for a farm, clusters, Oracle HTTP Server, OPMN, OC4J, and JAZN. For OC4J, in addition to configuration information related to the container itself, DCM manages all deployed J2EE applications.

If you use DCM clusters, DCM assures that any change to the configuration is automatically distributed to all members of the cluster. As an alternative to using clusters, an archive of a staged configuration can be applied manually to non-clustered instances in a farm.

A hybrid staging solution is to first stage and test changes to a non-clustered instance, archive the changes, and finally apply the archive to a DCM cluster. These changes are then automatically propagated to all members of the cluster.

For example, to create an archive prior to deploying a new J2EE application named "foo" use the command:

```
dcmctl createArchive -arch PriorToDeployingFoo -comment "prior to foo deploy V1"
```

When using `createArchive`, it is a good practice to use an archive name and a corresponding comment that identifies the version of configuration that the archive is associated with.

2.3.2 Specify a Single Instance in a Cluster as the Management Point

Oracle Application Server instances grouped in a cluster can be managed, as a single point of administration, using Application Server Control or `dcmctl` on any instance in the cluster. One instance should be used as the administrative point for the entire cluster.

Specifying a single instance in a cluster as the management point ensures that operations are executed in the correct order and are properly serialized.

2.3.3 Do not Perform Concurrent Administration Operations

When changing a instance specific configuration, (for example port numbers, host names or virtual hosts), on a particular instance in the cluster, you must ensure that there are no other administrative changes occurring concurrently in the cluster. You want to avoid conflicting changes to configuration settings. The result may be an unusable configuration.

Concurrent administration within a cluster is strongly discouraged. If multiple administrative operations are issued at the same time in a cluster, this can lead to errors and associated error messages.

2.3.4 Do not Run `updateConfig` Concurrently with any Other Configuration Operation

Do not run the `dcmctl updateConfig` command concurrently with any other `dcmctl` commands or Application Server Control configuration operations from

multiple Oracle Application Server instances in a farm or cluster. If `updateConfig` is being executed concurrently with other configuration operation, there is a risk of conflicting changes being placed in the metadata repository. This could leave the configuration stored in the metadata repository in a non-functional state and could require a restore from the archive.

2.3.5 Restart Application Server Control after Joining or Leaving a Farm or Cluster in a File Based Repository

When using a File-Based Repository, you should stop and then start Application Server Control after issuing the following `dcmctl` commands:

- `joinCluster`
- `joinFarm`
- `leaveCluster`
- `leaveFarm`

Use following commands to restart Application Server Control:

- `% emctl stop iasconsole`
- `% emctl start iasconsole`

2.3.6 Use High Availability Features for Infrastructure Repository

The infrastructure repository houses all the configuration information for the Oracle Application Server instances in a farm. This information is critical during startup, since DCM ensures that the local configuration of any node is synchronized with the configuration in this central repository. Therefore, it is a good idea to employ the high availability features for the infrastructure instance.

However, it is also important to understand that the database-based repository (in the case of a J2EE and Web Cache installation) is used for management operations and OracleAS Single Sign-On. Thus, if a site is not using single sign-on capabilities, then the repository is primarily required to be up when performing configuration management operations such as deploying new applications or, joining a, or moving from a cluster.

2.3.7 dcmctl Usage

The following are best practices when using `dcmctl`:

- Always use `-d` and `-v` options with `dcmctl` commands.

By default, the `dcmctl` script is configured for programmatic usage. Instead of displaying lengthy messages that can differ across releases and languages, error codes are displayed, such as ADMN-90605. Scripting tools can use these error codes to perform different activities based upon the result of commands.

Unfortunately a message like ADMN-906005 does not mean much by itself. In order to see an explanation of the error code, the `-d` and `-v` switches should be used whenever possible.

- Always use `dcmctl getreturnstatus` to determine whether a command failed after timeout

Long running operations will often timeout but continue to execute asynchronously. This is indicated by `dcmctl` with an ADMN-906005 error code:

Using the `dcmctl deployApplication` command with the `-v` option as an example, the following message will be displayed.

"The specified command "deployApplication", is being executed asynchronously. The maximum wait time of n seconds has been reached. This operation will continue to execute to completion. Use the "getReturnStatus" command to determine if/when the operation completes successfully."

Once this timeout message is received, you can invoke the `dcmctl getReturnStatus` command periodically until the operation has completed.

- Use `dcmctl` shell mode for multiple commands.

When you need to perform a number of `dcmctl` commands, the `dcmctl` shell or the `dcmctl` command file options should be used. See the *Distributed Configuration Management Reference Guide* for complete documentation on `dcmctl`.

Each initialization of `dcmctl` requires creation of a Java Virtual Machine and the parsing of a number of XML documents. This initialization only has to occur once if using a `dcmctl` shell versus multiple times if executing a set of `dcmctl` commands individually.

2.4 Dynamic Monitoring Services Best Practices

This section describes Dynamic Monitoring Services (DMS) best practices. It includes the following topics:

- [Section 2.4.1, "Monitor Your System Regularly"](#)
- [Section 2.4.2, "Take Regular Dumps of Metrics"](#)
- [Section 2.4.3, "Instrument Applications with DMS"](#)
- [Section 2.4.4, "Isolate Expensive Intervals Using PhaseEvent Metrics"](#)
- [Section 2.4.5, "Organize Performance Data"](#)
- [Section 2.4.6, "DMS Naming Conventions"](#)
- [Section 2.4.7, "DMS Coding Recommendations"](#)
- [Section 2.4.8, "Validate New Metrics"](#)

2.4.1 Monitor Your System Regularly

It is a good practice to monitor Oracle Application Server regularly. Monitoring Oracle Application Server and obtaining performance data can assist you in tuning the system and debugging applications with performance problems.

Refer to *Oracle Application Server 10g Performance Guide* for available monitoring tools.

2.4.2 Take Regular Dumps of Metrics

Run the `dmstool` command with the `-dump` option periodically, such as every 15 to 20 minutes, to capture and save a record of performance data for your Oracle Application Server installation. If you save performance data over time, it can assist you if you need to analyze system behavior to improve performance or if problems occur. Using `dmstool -dump` reports all the available metrics on the standard output. The `-dump` option also supports the `format=xml` query. Using this query at the end of the command line supplies the metric output in XML format.

2.4.3 Instrument Applications with DMS

Consider instrumenting applications with DMS metrics. Adding performance instrumentation to Java applications will help developers, system administrators and support analysts understand system performance and monitor system status. DMS instrumentation refers to the process of inserting DMS calls into application code. Using the DMS API is a simple and efficient way to enable your application to measure, collect, and save performance information.

To create DMS metrics, developers add calls that notify DMS when events occur, when important intervals begin and end, or when pre-computed values change their state. At runtime, DMS stores performance information, called DMS metrics, in memory and allows you to save or view the metrics.

Refer to *Oracle Application Server 10g Performance Guide* for available monitoring tools.

2.4.4 Isolate Expensive Intervals Using PhaseEvent Metrics

Carefully consider the requirements for new metrics when you add DMS instrumentation. It is important to add a sufficient number of metrics to validate that your code is behaving as desired but not so much that the useful statistics become buried in too much detail. As a guide, try to observe the following rules when you add DMS metrics:

- Add metrics only to provide an overview of the time the system spends in your block of code or module. You do not need to collect performance data for every method call, or for every distinct phase of your code or module.
- When your code calls external code that you do not control, and you expect that this could take a significant amount of time, add a PhaseEvent Sensor to track the start and the completion of the external code.

2.4.5 Organize Performance Data

The DMS metrics are organized in a tree, with leaf nodes acting as Sensor metrics and branching nodes acting as Nouns. Define DMS Nouns to organize Sensors and their associated metrics. Use Noun types for Nouns that directly contain Sensors. When a Noun contains only Nouns, and does not directly contain Sensors, AggreSpy displays the Noun type as a metric table, with no metrics.

2.4.6 DMS Naming Conventions

Follow the guidelines for defining DMS name for users viewing DMS metric reports. This allows users to easily understand metrics across applications and across Oracle Application Server components. In applying the naming convention rules, try to be as clear as possible, if there is a conflict, you might need to make an exception.

In general, try to use only alphanumeric and underscore characters for naming and avoid using the "/" character.

Refer to *Oracle Application Server 10g Performance Guide* for different naming conventions.

2.4.7 DMS Coding Recommendations

Use the following coding recommendations for working with DMS:

- When you create a new Noun or Sensor (`PhaseEvent`, `Event`, or `State`), its full name must not conflict with names in use by Oracle built-in metrics, or by other applications.
- Avoid frequently creating and destroying Nouns and Sensors.
- The DMS API calls are thread safe; they provide sufficient synchronization to prevent races and access bugs.
- Avoid creating `PhaseEvents` which do not measure a section of code that is expensive under some set of conditions.
- Be sure all `PhaseEvents` are stopped. Put the `PhaseEvents start()` in a try block with the `stop()` in the finally block.
- Avoid creating any DMS Sensor or Noun more than once. You should define Sensors and Nouns during static initialization, or in the case of a Servlet, in the `init()` method.
- Assign a type for each Noun. Nouns with no type specified are not shown in the `Spy` or `AggreSpy` display.

2.4.8 Validate New Metrics

You should test and verify the accuracy of the metrics that you add to Java applications. Use the `dmstool` and the other available DMS monitoring tools to verify and test new metrics. Try the following to validate new metrics:

- Do expected metrics appear in the display?
- Do unexpected metrics appear in the display?
Verify that you have only added the metrics that you planned to add.
- Are the metric values you see within reasonable ranges?
For example, a `size of pool` metric should never report a negative value.
- Are metric values accurate?
This can be difficult to test; however, if an alternate means of measuring a particular metric is available then use it to verify metric values. For example, you can verify an Event Sensor count metric by examining records that you write to a log file or to the console.
- When integrating DMS instrumentation with an existing package or when implementing a new feature, consider insulating a previously working system.
For example, you could include an option to enable and disable new DMS metrics.

Platform Security and Identity Management

This chapter describes security and management best practices for Oracle Application Server. It includes the following topics:

- [Section 3.1, "General Best Practices"](#)
- [Section 3.2, "JAAS Best Practices"](#)
- [Section 3.3, "J2EE Security Best Practices"](#)
- [Section 3.4, "OracleAS Single Sign-On Best Practices"](#)
- [Section 3.5, "Oracle Internet Directory Deployment Best Practices"](#)

3.1 General Best Practices

This section describes general best practices for security and management. It includes the following topics:

- [Section 3.1.1, "Best Practices for HTTPS Use"](#)
- [Section 3.1.2, "Assign Lowest Level Privileges Adequate for the Task"](#)
- [Section 3.1.3, "Best Practices for Cookie Security"](#)
- [Section 3.1.4, "Best Practices in Systems Setup"](#)
- [Section 3.1.5, "Best Practices for Certificates Use"](#)
- [Section 3.1.6, "Review Code and Content Against Already Known Attack"](#)
- [Section 3.1.7, "Follow Common Sense Firewall Practices"](#)
- [Section 3.1.8, "Leverage Declarative Security"](#)
- [Section 3.1.9, "Use Switched Connections in DMZ"](#)
- [Section 3.1.10, "Place Application Server in the DMZ"](#)
- [Section 3.1.11, "Secure Sockets Layer"](#)
- [Section 3.1.12, "Tune the SSL SessionCacheTimeout Directive"](#)
- [Section 3.1.13, "Plan Out The Final Topology Before Installing Oracle Application Server Security Components"](#)

3.1.1 Best Practices for HTTPS Use

The following are recommended for using HTTPS with Oracle Application Server:

- Configure **Oracle Application Server** to fail attempts that use weak encryption. Oracle Application Server can be configured to use only specific encryption

ciphers for HTTPS connections. Connections from all old Web browsers that have not upgraded the client-side secure sockets layer (SSL) library to 128-bit can be rejected. This functionality is especially useful for banks and other financial institutions because it provides server-side control of the encryption strength for each connection.

- **Use HTTPS to HTTP appliances for accelerating HTTP over SSL.** Huge performance overhead of HTTPS forces a trade-off in some situations. Use of HTTPS to HTTP appliances can change throughput from 20 to 30 transactions per second on a 500MHz Unix to 6000 transactions per second for a relatively low cost, making this trade-off decision easier. This is a better solution than math/crypto cards, which can be added to UNIX/NT/Linux computers.
- **Ensure that sequential HTTPS transfers are requested through the same Web server.** Expect 40/50 milliseconds CPU time for initiating SSL sessions on a 500 MHz computer. Most of this CPU time is spent in the key exchange logic, where the bulk encryption key is exchanged. Caching the bulk encryption key will significantly reduce CPU overhead on subsequent access, provided that the access is routed to the same Web server.
- **Keep secure pages and pages not requiring security on separate servers.** While it may be easier to place all pages for an application on one HTTPS server, the resulting performance cost is very high. Reserve your HTTPS server for pages that require SSL. Put pages that do not require SSL on an HTTP server.

If secure pages are composed of many .GIF, .JPEG, or other files that would be displayed on the same screen, it is probably not worth the effort to segregate secure from non-secure static content. The SSL key exchange (a major consumer of CPU cycles) is likely to be called exactly once in any case, and the overhead of bulk encryption is not that high

3.1.2 Assign Lowest Level Privileges Adequate for the Task

When assigning privileges to module(s), use the lowest levels adequate to perform the module(s) function(s). This is essentially "fault containment" which means if security is compromised, it is contained within a small area of the network and cannot invade the entire intranet.

3.1.3 Best Practices for Cookie Security

Use the following as guidelines for cookies:

- **Make sure that cookies have proper expiration dates.** Permanent cookies should have relatively short expiration dates of about three months or less. This will avoid cluttering client Web browsers, which may cause errors if the Web browser cannot transmit all the valid cookies. Non-permanent cookies should be set to expire when the relevant application exits.
- **Make sure that information in cookies contains Method Authentication.** Method Authentication should be used to ensure that cookie data has not been changed since the application set the data. This helps ensure that the cookie cannot be modified and deceive the application. Also, this helps prevent application failures if the cookie is inadvertently corrupted.
- **Make sure that the size and varieties of cookies are kept low.** There is a finite number and aggregate size of cookies that Web browsers support. If this is exceeded, then the Web browsers will not send all the relevant cookies leading to application failures. Also, very large cookies can result in performance degradation.

- **Carefully use cookie domain name facilities.** Use of cookie domains should ensure that the domain is the smallest possible. Making the domain `oracle.com`, for instance, would mean that any host in `oracle.com` would get the cookie. With hundreds of applications on different parts of `oracle.com`, a domain of `oracle.com` for each of them results in attempts to send hundreds of cookies for each HTTP input operation.

3.1.4 Best Practices in Systems Setup

Use the following as guidelines for system setup:

- **Apply all relevant security patches.** Check MetaLink (<http://metalink.oracle.com>) and OTN (<http://otn.oracle.com>) for current security alerts. Many of these patches address publicly announced security issues.
- **When deploying software, change all default passwords and close accounts used for samples and examples.**
- **Remove unused services from all hosts.** Examples of unused services are FTP, SNMP, NFS, BOOTP, and NEWS. HTTP or WebDAV may be good alternatives.
- **Limit the number of people with root and administrative privileges.**
- **In UNIX, disable the "r" commands if you do not need them.** For example, `rhost`, `rcp`.

3.1.5 Best Practices for Certificates Use

Use the following guidelines when using certificates:

- **Ensure that certificate organization unit plus issuer fields uniquely identify the organization across the Internet.** One way to accomplish this would be to include the Dun and Bradstreet or IRS identification as identification for the issuer and the organizational unit within the certificate.
- **Ensure that certificate issuer plus distinguished name uniquely identify the user.** If the combination of issuer and distinguished name is used as identification, there is no duplication risk.
- **Include expiring certificates in tests of applications using certificates.** Expiration is an important consideration for a number of reasons. Unlike most username/password-based systems, certificates expire automatically. With longer duration certificates, fewer re-issues are required, but revocation lists become larger.

In systems where certificates replace traditional usernames/passwords, expiring certificate situations may result in unexpected bugs. Careful consideration of the effects of expiration is required and new policies will have to be developed because most application and infrastructure developers have not worked in systems where authorization might change during transactions.

- **Use certificate re-issues to update certificate information.** Because certificates expire, infrastructure for updating expired certificates will be required. Take advantage of the re-issue to update organizational unit or other fields. In cases of mergers, acquisitions, or status changes of individual certificate holders, consider re-issuing even when the certificate has not yet expired. But pay attention to key management. If the certificate for a particular person is updated before it expires, for example, put the old certificate on the revocation list.

- **Audit certificate revocations.** Revocation audit trails can help you reconstruct the past when necessary. An important example is replay of a transaction to ensure the same results on the replay as during the original processing. If the certificate of a transaction participant was revoked between the original and the replay, failures may occur which would not have occurred when the original transaction was processed. For these cases, the audit trail should be viewed to simulated authentication at the time when the transaction was initially processed.

3.1.6 Review Code and Content Against Already Known Attack

It is quite common for viruses or known attacks to resurface in slightly altered shape or form. Thus, just because a threat has been apparently eliminated does not mean it will not resurface. Use the following as guidelines to minimize the recurrence of the threat:

- **Ensure that programs are reviewed against double encoding attacks.** There are many cases where special characters, such as <, >, | are encoded to prevent cross-site scripting attacks or for other reasons. For example, "<" might be substituted for ">". In a double encoding, the attacker might encode the "&" so that later decoding might involve the inadvertent processing of a >, <, or | character as part of a script. Prevention of this attack, unfortunately, can only be provided by careful program review, although some utilities can be used to filter escape characters that might result in double encoding problems in later processing.
- **Ensure that programs are reviewed against buffer overflow for received data.**
- **Ensure that programs are reviewed against cross-site scripting attacks.** This attack typically tricks HTML and XML processing via input from Web browsers (or processes which act like Web browsers) to invoke scripting engines inappropriately. However, it is not limited to the Web technologies, and all code should be evaluated for this.

3.1.7 Follow Common Sense Firewall Practices

The following are some common recommended practices pertaining to firewalls; while not unique to Oracle Application Server, these are important to overall Oracle Application Server security:

- Place servers providing Internet services behind an exterior firewall of the stateful inspection type. Stateful inspection means that the firewall keeps track of various sessions by protocol and ensures that illegal protocol transitions are disallowed through the firewall. This blocks the types of intrusion that exploit illegal protocol transitions.
- Set exterior firewall rules to allow Internet-initiated traffic only through specific IP and PORT addresses where SMTP, POP3, IMAP, or HTTP services are running. Some protocols (for example, IIOP) leave ports open without receiving processes. PORT and IP combinations that are not assigned to running programs should not be permitted.
- Set interior firewall rules to allow messages through to the intranet only if they originate from servers residing on the perimeter network. All incoming messages must first be processed in the perimeter network.
- Send outgoing messages through proxies on the perimeter network.
- Do not store the information of record on bastion hosts. Bastion hosts are fortified servers on the perimeter network. Information and processing should be segmented such that the bastion hosts provide initial protocol server processing

and generally do not contain information of a sensitive nature. The database of record and all sensitive processing should reside on the intranet.

- Disallow all traffic types unless specifically allowed. allow only the traffic required by Oracle Application Server for better security. For example, HTTP, AJP, OCI, LDAP.

3.1.8 Leverage Declarative Security

Oracle HTTP Server has several features that provide security to an application without requiring the application to be modified. These should be leveraged and/or evaluated before programming similar functionality as those features into the application. Specifically:

- **Authentication:** Oracle HTTP Server can authenticate users and pass the authenticated user-id to an application in a standard manner. It also supports single sign-on, thus reusing existing login mechanisms.
- **Authorization:** Oracle HTTP Server has directives that can allow access to your application only if the end user is authenticated and authorized. Again, no code change is required.
- **Encryption:** Oracle HTTP Server can provide transparent SSL communication to end customers without any code change on the application.

These three features should be leveraged heavily before designing any application specific security mechanisms.

3.1.9 Use Switched Connections in DMZ

Oracle recommends that all DMZ attached devices be connected by switched, not bussed connections. Furthermore, devices such as the Cisco 11000 series devices, which can provide IP, port, and protocol rules between each pair of connected devices are preferred.

3.1.10 Place Application Server in the DMZ

Application servers should exist in the DMZ. In this architecture Oracle Application Server Web Cache only forwards requests to computers containing Web servers. Web servers only forward requests to application servers or via PL/SQL to database servers. The application servers only forward inward requests to the database or, perhaps, special message processing processors in the intranet. This provides excellent fault containment because a compromised Web server must somehow compromise an application server before the database can be attacked.

3.1.11 Secure Sockets Layer

SSL encryption can be used to secure both LDAP and HTTP traffic that passes between the various components of the Oracle Application Server. To ensure that all LDAP queries being sent to Oracle Internet Directory are SSL-encrypted, you need to configure your Oracle Internet Directory instance to run with a configuration set that supports only SSL-encrypted LDAP connections. The default mode installed with Oracle Application Server allows a given Oracle Internet Directory instance to be configured to listen on both SSL and non-SSL ports. Refer to the *Oracle Internet Directory Administrator's Guide* for more details on configuring Oracle Internet Directory instances with SSL.

SSL encryption is unrelated to the installation or use of HTTPS, which allows users to access Oracle Application Server components over HTTP while using SSL to encrypt Web client packets."

3.1.12 Tune the SSL SessionCacheTimeout Directive

The Apache server in Oracle Application Server caches a client SSL session information by default. With session caching, only the first connection to the server incurs high latency.

In a simple test to connect and disconnect to an SSL-enabled server, the elapsed time for 5 connections was approximately 11.4 seconds without SSL session caching as opposed to approximately 1.9 seconds when session caching was enabled.

The default `SSLSessionCacheTimeout` is 300 seconds. Note that the duration of a SSL session is unrelated to the use of HTTP persistent connections. You can change the `SSLSessionCacheTimeout` directive in `httpd.conf` file to meet your application needs.

3.1.13 Plan Out The Final Topology Before Installing Oracle Application Server Security Components

Consult the *Oracle Application Server 10g Advanced Topologies for Enterprise Deployments* and the *Oracle Identity Management Concepts and Deployment Planning Guide* documents when planning out the final target topology. Identify the steps in installing and configuring the various Oracle Application Server components consistent with the options of the Oracle Universal Installer, rather than approaching the desired topology on an ad-hoc basis.

3.2 JAAS Best Practices

Oracle Application Server provides an implementation of Java Authentication and Authorization Service (JAAS) for J2EE applications that is fully integrated with J2EE declarative security. This allows J2EE applications to take advantage of the JAAS constructs such as principal-based security and pluggable login modules. Optionally, the Oracle JAAS implementation allows J2EE applications running on OC4J to leverage the central security services of Oracle Identity Management.

3.3 J2EE Security Best Practices

This section describes J2EE security best practices. It includes the following topics:

- [Section 3.3.1, "Avoid Writing Custom User Managers"](#)
- [Section 3.3.2, "Authentication Mechanism with the JAAS Provider"](#)
- [Section 3.3.3, "Use Fine-Grained Access Control"](#)
- [Section 3.3.4, "Use Oracle Internet Directory as the Central Repository"](#)
- [Section 3.3.5, "Develop Appropriate Logout Functionality for J2EE Applications"](#)

3.3.1 Avoid Writing Custom User Managers

The OC4J container continues to provide several methods and levels of extending security providers. The `UserManager` class can be extended to build a custom user manager that allows you to leverage the functionality provided by the JAAS Provider. Both Oracle Application Server Single Sign-On and Oracle Internet Directory provide

APIs to integrate with external authentication servers and directories respectively, thus allowing developers more time to focus on actual business logic instead of infrastructure code.

3.3.2 Authentication Mechanism with the JAAS Provider

OC4J allows different authentication options for J2EE applications. Oracle recommends leveraging the OracleAS Single Sign-On server whenever possible for the following reasons:

- It is the default mechanism for most Oracle Application Server components such as OracleAS Portal, OracleAS Forms Services, OracleAS Reports Services, and OracleAS Wireless.
- It is easy to setup in a declarative fashion and does not require any custom programming.
- It provides a seamless way for PKI integration.

For environments where OracleAS Single Sign-On is not available, and custom authentication is required, one should use JAAS compliant `LoginModules` to extend OC4J authentication. When using `LoginModules`, it is important to only use application relevant principals (roles) associated with the authenticated subject to preserve least privilege.

3.3.3 Use Fine-Grained Access Control

Unlike the coarse-grained J2EE authorization model as it exists today, the JAAS Provider integrated with OC4J allows any protected resource to be modeled using Java permissions. The Java permission model (and associated `Permission` class) is extensible and allows a flexible way to define fine-grained access control.

For example, a servlet can be written with `Subject.doAs` or `Subject.doPrivileged` to control code that executes sensitive operations.

3.3.4 Use Oracle Internet Directory as the Central Repository

Although the JAAS Provider supports a flat-file XML-based repository useful for development and testing environments, it should be configured to use Oracle Internet Directory for production environments. Oracle Internet Directory provides LDAP standard features for modeling administrative metadata and is built on the Oracle database platform inheriting all of the database properties of scalability, reliability, manageability, and performance. To optimize performance, adjust the caching configurations appropriate for your environment.

3.3.5 Develop Appropriate Logout Functionality for J2EE Applications

Simple J2EE applications using HTTP Basic authentication do not support the concept of logout, relying instead on the user to close the Web browser. When using other forms of authentication, including OracleAS Single Sign-On, it is important to plan out various logout and timeout flows. OC4J has an adjustable HTTP session inactivity parameter that is set to 20 minutes by default. If J2EE applications are leveraging OracleAS Single Sign-On and want to support full logout functionality, they should be written with the appropriate logout dynamic directives as described in the *Oracle Application Server Single Sign-On Application Developer's Guide*.

3.4 OracleAS Single Sign-On Best Practices

This section describes OracleAS Single Sign-On best practices. It features the following topics:

- [Section 3.4.1, "Configure for High Availability"](#)
- [Section 3.4.2, "Leverage Oracle Application Server Single Sign-On"](#)
- [Section 3.4.3, "Use an Enterprise-Wide Directory in Place"](#)
- [Section 3.4.4, "Use OracleAS Single Sign-On Instead of Writing Custom Authentication Logic"](#)
- [Section 3.4.5, "Always Use SSL with Oracle Application Server"](#)
- [Section 3.4.6, "Username and Password Only on Login Screen"](#)
- [Section 3.4.7, "Log Out So Cookies Do Not Remain Active"](#)

3.4.1 Configure for High Availability

Single sign-on failure is catastrophic since it means no single sign-on protected application can be accessed. Two recommendations for high availability of OracleAS Single Sign-On are:

- Carefully consider inclusion of any other types of processing on the single sign-on servers since this can make instability more likely.
- Consider deploying multiple single sign-on servers fronted by load balancing hardware to protect against failures in single sign-on listeners. In this case, the address of the load balancer is used as the single sign-on address and the single sign-on listener configuration information is replicated. It is also recommended that the database be a RAC configured for additional improvements in availability. Configuration details for multiple single sign-on servers can be found at OTN (<http://otn.oracle.com>).

3.4.2 Leverage Oracle Application Server Single Sign-On

OracleAS Single Sign-On should be used as the primary point of security. This is a benefit administratively and a major convenience to application customers. Also, OracleAS Single Sign-On is well integrated with the rest of OracleAS Infrastructure 10g and can, via Oracle Internet Directory and other means, be integrated with non-Oracle application and infrastructure. Also, as single sign-on becomes a single point for authentication, opportunities to attack the multiple authentication entities of sites today are reduced.

OracleAS Single Sign-On single authenticated user for all applications allows better control for more uniform authorization.

3.4.3 Use an Enterprise-Wide Directory in Place

In order to deploy an effective single sign-on solution, the user population must be centralized in a directory, preferably an LDAP-based directory such as Oracle Internet Directory. Having users represented in multiple systems (for example, in multiple Microsoft Windows NT domains) makes setting up the infrastructure for a common identity more difficult. In addition, clearly defining and automating the user provisioning process makes managing the single sign-on environment much easier.

3.4.4 Use OracleAS Single Sign-On Instead of Writing Custom Authentication Logic

OracleAS Single Sign-On provides the infrastructure to validate credentials and allows for various different authentication mechanisms such as username, password, X.509 certificates. Moreover, since these can be shared across different applications and Web sites, end users do not have to create a new username, password for each different corporate application.

3.4.5 Always Use SSL with Oracle Application Server

The OracleAS Single Sign-On server simplifies user interaction by providing a mechanism to have a single username and password that can be used by multiple partner applications. However, with this ease of use, comes the caution that the single sign-on server should always be accessed in the correct fashion; a breach of the common password can now put all partner applications at risk. Hence, the single sign-on server should always be configured to allow connections in SSL mode only. This protects the end user's credentials going across the wire. Applications where security and data confidentiality is important should also be protected by SSL. From a performance perspective, use of SSL hardware accelerators is recommended.

3.4.6 Username and Password Only on Login Screen

The OracleAS Single Sign-On server provides a standard login screen. This login page is serviced from the single sign-on server, which typically is installed on a different computer from the one the end user is trying to access. Thus, it is critical that before the end user enters their login and password, that a valid single sign-on screen is observed. This prevents users from unknowingly providing their username or password to inappropriate servers.

3.4.7 Log Out So Cookies Do Not Remain Active

Most users do not log out of Internet applications and this creates problems at two levels:

1. A security risk. Another person accessing the work station can now reuse the cookie. Also, since the session remains valid until it times out, a hacker from another machine has a longer time window to guess the session id/cookie value.
2. The system resources on the server associated with the cookie are not released until the session is ended or invalidated.

For application developers and administrators, single sign-on session duration and inactivity timeouts should be configured appropriately (for example, one hour inactivity timeouts for sensitive applications).

For external applications, OracleAS Single Sign-On is unable cannot logout users. Therefore, closing all Web browser windows is important.

3.5 Oracle Internet Directory Deployment Best Practices

This section describes Oracle Internet Directory deployment best practices. It includes the following topics:

- [Section 3.5.1, "Use bulkload.sh Utility"](#)
- [Section 3.5.2, "Replicate for High Availability"](#)
- [Section 3.5.3, "Use SSL Binding"](#)

- [Section 3.5.4, "Use Backup and Restore Utilities"](#)
- [Section 3.5.5, "Monitoring and Auditing Oracle Internet Directory"](#)
- [Section 3.5.6, "Assign Oracle Internet Directory Privileges"](#)
- [Section 3.5.7, "Change Access Control Policies"](#)
- [Section 3.5.8, "Best Practice for Directory Integration Platform"](#)
- [Section 3.5.9, "Recommendations for Migrating Oracle9iAS Applications to an Existing Oracle Internet Directory"](#)
- [Section 3.5.10, "Configuration of the Self-Service Console"](#)
- [Section 3.5.11, "Use opmnctl instead of oidmon and oidctl"](#)
- [Section 3.5.12, "Configure Active Directory Synchronization"](#)
- [Section 3.5.13, "Use User Attributes and Password Hints for Resets"](#)

Additionally, Oracle also recommends the following documentation for deployment of Oracle Internet Directory:

- *Oracle Internet Directory Administrator's Guide*
- *Oracle Application Server 10g Administrator's Guide*
- *Oracle Identity Management Concepts and Deployment Planning Guide*
- *Oracle Process Manager and Notification Server Administrator's Guide*
- *Oracle Application Server Single Sign-On Administrator's Guide*

3.5.1 Use bulkload.sh Utility

The `bulkload.sh` utility checks standard LDIF formatted files for schema violations and duplicates, and generates SQL*Loader intermediate files for fast loading into the database tables underlying Oracle Internet Directory. Use the `bulkload.sh` utility whenever there is an initial bootstrap required. For example, when setting up synchronization with Microsoft Active Directory or other LDAP directory servers.

Oracle recommends passing the LDIF file output from third-party LDAP directories into `bulkload.sh -check` mode, which will alert you to any problems with your existing LDAP schema.

Most third-party LDAP directories (including Oracle Internet Directory) support output to LDIF without any operational attributes (which typically cannot be loaded into another vendor's directory). If you are loading data into Oracle Internet Directory from another directory which does not support this, you will have to manually remove any operational attributes prior to sending the LDIF file to `bulkload.sh -generate` mode.

If your input LDIF file is from another Oracle Internet Directory instance, then you must use the `-restore` option to `bulkload.sh` to preserve these operational attributes as is during the bulkload.

For more information on the `bulkload.sh` utility refer to the *Oracle Internet Directory Administrator's Guide*.

3.5.2 Replicate for High Availability

Oracle Internet Directory supports both multimaster and fan out styles of directory replication. Refer to the *Oracle Identity Management Concepts and Deployment Planning Guide* for guidelines.

For high availability, consider placing an Oracle Internet Directory multimaster replication group behind a network load balancer to provide a single IP address to your LDAP client applications. If a replicated node becomes unavailable, the load balancer can be configured to re-route requests automatically to an available server.

Additionally, each Oracle Internet Directory node can run on Oracle Application Server RAC, further improving availability through increased database uptime and data availability. Other high availability solutions deployable with Oracle Internet Directory are listed in the *Oracle Identity Management Concepts and Deployment Planning Guide*.

3.5.3 Use SSL Binding

SSL is considered the Internet standard protocol for highly secure transportation of data. In addition to the strong PKI authentication using digital certificates, SSL also provides multiple data integrity and data encryption layers to protect your communication channels. SSL provides multiple cipher suites with varieties of encryption algorithms for many security levels.

Oracle Internet Directory supports three SSL authentication modes:

1. Confidentiality mode (no-authentication mode)

In this mode, SSL cipher suites use the Diffie-Hellman algorithm to generate a session key for client or server at run time. The session key will be used to encrypt the communication channel. No server or user SSL wallet is necessary. In this mode, the channel will be encrypted using a Diffie-Hellman key.

2. Server Authentication only mode

This mode essentially uses certificates for authentication. The client needs to verify the server certificate. This mode is most commonly used in the Internet environment since any client that needs to communicate with an SSL server does not require a certificate. A client can use their user and password identification to authenticate itself to the server. The username and password are protected by SSL encryption when being transferred on the wire.

3. Server and Client Authentication mode (Mutual authentication)

In this mode, both client and server use RSA certificates to authenticate each other. First, the client authenticates the server by validating its certificate. In return, the server also requires the client to send its certificate to prove its authenticity.

In addition to choosing an authentication mode, you should choose appropriate security algorithms. For more information refer to the *Oracle Internet Directory Administrator's Guide*

3.5.4 Use Backup and Restore Utilities

Depending on your Oracle Application Server enterprise topology, you may want to consider backing up Oracle Internet Directory as part of backing up your entire application server environment. For more information, refer to the *Oracle Application Server 10g Administrator's Guide* and the *Oracle Internet Directory Administrator's Guide* before deciding on an overall backup and recovery strategy for all of your Oracle Identity Management Infrastructure components.

3.5.5 Monitoring and Auditing Oracle Internet Directory

You can monitor and audit Oracle Internet Directory in one of three ways:

1. The Oracle Enterprise Manager LDAP page provides a very simple way to monitor the LDAP service and determine if it is up and running under its associated load.
2. You can also check the log files of various LDAP processes to ensure there are no errors showing up.
3. LDAP audit log service provides more granular information such as security violation information or sensitive events. The audit log can be further customized to specific directory operations and events.

Oracle recommends that you perform, at the very least, a weekly review of the audit and error logs. System administrators can do a more regular review via Oracle Enterprise Manager to provide better availability.

3.5.6 Assign Oracle Internet Directory Privileges

While it is possible to install Oracle Application Server as an Oracle Internet Directory super user, Oracle recommends that this not be done as it imparts more privileges than required.

To install Oracle Application Server, a user needs to be a member and owner of the Oracle Application Server Administrator's group.

When installing Oracle Application Server, the directory administrator should add the installation user as a member and owner of the Administrator's group. The administrator should then remove the member as the owner once the installation has completed.

3.5.7 Change Access Control Policies

Oracle Internet Directory administrators should change the default access control policies to better control user administration as required.

Oracle Internet Directory administrators should adjust the default access control and password policies using Oracle Directory Manager, in accordance with specific administrative policies for directory access and passwords. This includes both value and state parameters. Refer to the *Oracle Internet Directory Administrator's Guide* for specific parameters affecting both Access Control and Password Policies."

3.5.8 Best Practice for Directory Integration Platform

This section includes the following topics:

- [Section 3.5.8.1, "Use Identity Management Realms"](#)
- [Section 3.5.8.2, "Configuring DIP Synchronization Service"](#)
- [Section 3.5.8.3, "Oracle HR Synchronization"](#)

3.5.8.1 Use Identity Management Realms

Directory Integration Platform (DIP) should be used to build connectivity between Oracle Internet Directory and third party directories. This provides seamless integration with other Oracle products. It enables the Oracle products to work in the presence of third party directories in the enterprise and also provides sharing with the same identities in other directories.

The different identities for the same enterprise user from multiple LDAP directories can be joined or unified into a single global identity in Oracle Internet Directory using

DIP. This facilitates a true single sign-on environment in an enterprise using Oracle Internet Directory and Oracle Application Server Single Sign-On.

Oracle Internet Directory supports representation of multiple applications and multiple realms or administration Contexts in the Oracle Internet Directory. Various enterprise applications can be provisioned for a single or multiple realms. There are automated tools to create new realms and to provision applications for various realms. These tools setup the various levels of access required by the application to manage the realm.

User definitions from third party identity management systems should be synchronized via the DIP into the appropriate realms to create an enterprise view of all relevant user namespaces and their defined services.

Refer to the *Oracle Internet Directory Administrator's Guide* for more information.

3.5.8.2 Configuring DIP Synchronization Service

When configuring DIP, specify only the containers and attributes which are required in the connected directory or in Oracle Internet Directory. LDAP filters can be used as part of mapping configuration profiles to screen out unwanted attribute data and keep synchronization simple.

Each connector and its associated mapping configuration file should be set to an appropriate scheduling interval. No connector needs to fire at the same time or at the same interval as any another, as they are completely independent of one another.

When synchronizing external users and groups into Oracle Internet Directory for use with Oracle Application Server, be sure to establish connectors to the appropriate Identity Management Realm `cn=users` and `cn=groups` container. DIP will then provision all inbound user entries with the Oracle-specific attributes needed to enable users to interact with their deployed Oracle applications.

A synchronization Profile has to be disabled before altering any status attributes through the Oracle Directory Manager. After the change, it needs to be enabled once again.

Refer to the *Oracle Internet Directory Administrator's Guide* for more information.

3.5.8.3 Oracle HR Synchronization

Since the `Last Successful Execution Time` connector profile attribute is used to fetch the desired changes from connected directories at a given time, set it initially to some date in the past. Then enable the profile. Note this technique will potentially cause all entries in the connected directory to be synchronized all at once into Oracle Internet Directory. If this is not desirable, use the `bulkload.sh` technique for bootstrapping Oracle Internet Directory and then set the last change number appropriately to begin synchronizing incrementally from the connected directory instead.

It is a good idea to synchronize user data from connected directories to the public `cn=users` container within an Oracle Internet Directory Identity Management realm. This way, all users are immediately accessible to OracleAS Single Sign-On and Delegated Administration Services such as the Self-Service Console.

The nickname attribute should be synchronized from the connected directory or derived from some attribute which is unique in the connected directory, so that the user can use this identifier with OracleAS Single Sign-On.

Since the `Last Successful Execution Time` connector needs appropriate privilege to read and write to the `cn=users` container under the Identity Management

Realm, the profile distinguished name (DN) should be added to the groups `DASCreateUserGroup`, `DASEditUserGroup`, and `DASDeleteUserGroup` for that realm. Refer to the *Oracle Internet Directory Administrator's Guide* for more information.

3.5.9 Recommendations for Migrating Oracle9iAS Applications to an Existing Oracle Internet Directory

Oracle Application Server 10g installs an Identity Management infrastructure (including Oracle Internet Directory) as part of each Standard or Enterprise Edition install. However, some organizations may need to delay the upgrade of their Identity Management Infrastructure (Oracle Internet Directory, OracleAS Single Sign-On, DAS, DIP, OCA) and thus need to upgrade mid-tier components from Oracle9iAS to Oracle Application Server 10g while maintaining an existing Oracle Internet Directory installation from previous versions of Oracle9iAS or Oracle9i DB. In this case, the following points need to be considered:

- Before starting the upgrade process, the user keys in the older repository must be made consistent with the keys used to identify users in Oracle Internet Directory. This will enable the upgrade process to correlate the private keys with those present in the production Oracle Internet Directory system.
- The upgrade process will install its own local infrastructure (including a local Oracle Internet Directory instance) against which the Oracle Application Server 10g mid-tier component(s) you are installing can be validated before switching to the previously deployed instance of Oracle Internet Directory.
- Once the upgrade is completed, the upgraded component should be verified for correctness against the newly installed, local Oracle Internet Directory instance.
- Once the verification is complete, simply redirect applications from using the new, local Oracle Internet Directory to using the production Oracle Internet Directory.

As with the upgrade case, the presence of a corporate directory in a deployment influences the process by which the deployment can roll out new services against existing Identity Management infrastructure. In case the existing corporate directory is replicated, some special steps need to be taken by administrators to create a test replica of the production Oracle Internet Directory. Then, install and verify the components against the test replica before switching to the production Oracle Internet Directory service. Refer to the *Oracle Application Server 10g Administrator's Guide* for more information.

3.5.10 Configuration of the Self-Service Console

Rather than creating users and assigning them to groups as separate steps, consider incorporating the group assignment step during user creation. To do this:

1. Log in to the Oracle Internet Directory Self-Service Console as a DAS privileged user (`orcladmin` or designate).
2. Select the Configuration tab.
3. Select **User Entry > Add Role**.
4. Search for and select any commonly-subscribed group entries.

Now whenever you or any other DAS privileged user performs a **Create User** sequence, the list of specified groups will appear in the next-to-last step, in a section called **Roles Assignment**. Simply click whichever checkboxes are relevant to the newly-created user, and that user will automatically be made a member of all the groups you specify.

3.5.11 Use opmnctl instead of oidmon and oidctl

In Oracle Application Server, you no longer need to run `oidmon` and `oidctl` to start and stop Oracle Internet Directory processes. OPMN stores the proper sequences and controls these services. Refer to the *Oracle Process Manager and Notification Server Administrator's Guide* for more information.

3.5.12 Configure Active Directory Synchronization

Prior to configuring Windows Native Authentication, be sure to first configure the Active Directory DIP Connector and bootstrapped the appropriate `cn=users` and `cn=groups` containers within your desired Oracle Identity Management Realm. Do not configure the External Authentication Plug-in for Active Directory if your goal is to enable Windows Native Authentication

See Also: ■ *Oracle Application Server Single Sign-On Administrator's Guide*

- *Oracle Internet Directory Administrator's Guide*

3.5.13 Use User Attributes and Password Hints for Resets

Users that forget their OracleAS Single Sign-On passwords can reset them on their own by using the Oracle Internet Directory Self-Service Console. You must authenticate yourself in one of the following ways:

- If, while previously changing their password, a user specified a password hint question, then the Confirm Additional Personal Information window will ask them for the correct answer to the reminder question when attempting a password reset.
- Users who have not previously set a password hint will be presented with the Confirm Additional Personal Information window, which prompts them for other personal data as configured the DAS administrator, via the Password Reset Validation field in the **Add New Attributes** Window of the User Management section of the Self-Service Console.

Refer to the *Oracle Internet Directory Administrator's Guide* for more information.

High Availability

This chapter describes high availability best practices for Oracle Application Server. It includes the following topics:

- [Section 4.1, "Distribute Identity Management Components"](#)
- [Section 4.2, "Use OPMN for Crash Handling and Monitoring"](#)
- [Section 4.3, "Analyze High Availability using iHAT or Topology Viewer"](#)
- [Section 4.4, "Use Metric Based Load Balancing to Tune High Availability"](#)

4.1 Distribute Identity Management Components

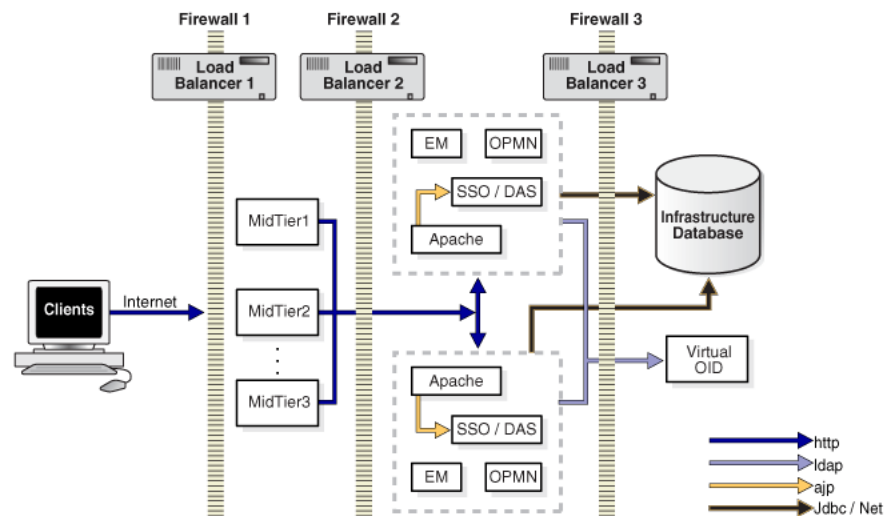
With Oracle Application Server 10g, Oracle Identity Management can be deployed in a variety of high availability configurations such as Oracle Application Server Cold Failover Clusters (OracleAS Cold Failover Cluster) and Active Failover Cluster (AFC).

AFC refers to the deployment of Oracle Identity Management and the OracleAS Metadata Repository on hardware clusters where each node of the cluster runs the database instances as well as the ID processes and all of them access a shared RAC database as their repository.

OracleAS Cold Failover Cluster solution for the infrastructure uses a two node hardware cluster (one active node and one passive node) that in case of failure of one of the nodes, relocates the execution of an application from a failed node to a designated standby node.

For both type of configurations, the "out-of-the-box" installation for Oracle Application Server places the OracleAS Single Sign-On service, Delegated Administration Service (DAS) and DIP in the same tier as the infrastructure database itself.

Both for scalability and security reasons, Oracle recommends separating the OracleAS Single Sign-On, DAS, and DIP services from the node or nodes where the infrastructure database resides as shown in [Figure 4-1](#). Since users access most enterprise applications from both within the company intranet as well as Internet, their access needs to be authenticated by ID in both cases. A distributed Oracle Identity Management deployment where the OracleAS Single Sign-On and DAS components reside in the DMZ meets this requirement. It is also important that while users are allowed to authenticate and access applications, their authentication data is well protected from unauthorized access from the Internet. Locating Oracle Internet Directory and the database repository for the identity data in the intranet ensures this. If necessary, the OracleAS Single Sign-On and DAS tier servers can be co-located with the middle tier servers (in separate Oracle homes) in the DMZ allowing them to be scaled in tandem with middle-tier growth.

Figure 4–1 Distributed Identity Management

In the specific case of AFC, the infrastructure database is a RAC database and a group of Oracle Internet Directories exposes a common virtual IP by using a Hardware Load Balancer where Firewall 3 is placed.

In the case of CFC, the clusterware running on the active and standby nodes where the infrastructure database resides expose the same virtual IP to OracleAS Single Sign-On, DAS, and middle tiers.

4.2 Use OPMN for Crash Handling and Monitoring

Oracle Application Server 10g allows you to configure Oracle Process Manager and Notification server (OPMN) to execute your own custom event scripts whenever a particular component starts, stops, or crashes. The code associated with crash events should:

- release any resources used by a component that has crashed
- log special characteristics associated with each crash event
- take possible actions that enhance the resiliency of the system

If the component being monitored by OPMN uses any special resources not controlled by Oracle Application Server, the crash event script should release them. A good example is any type of code in an OC4J container that uses JNI invocations. In this type of cases, OC4J does not control the resources under the JNI invocation, so that those resources may become trapped in the case of a Java Virtual Machine (JVM) crash. Common resources to control include sockets in use, memory in the JNI code, connections to database, and locked files.

The same script may take actions about the application that generated the crash. If it is likely that the crash was due to the overloading of the specific component, it could be useful to start a new instance of such a component in another node. For instance, the event script could be used for starting a new OC4J container in another application server instance that is mounting the same URL on Oracle HTTP Server. Or it could be used to relocate the repository in a file-based configuration when the DCM daemon of that specific instance crashes.

You can use OPMN to monitor your custom external process in a consistent mode with the default stack controlled by OPMN. Use OPMN to start, stop and monitor any

additional processes that are used by your system. For example, if your application uses an external resource such as an HTTP Server different from Oracle HTTP Server, you can add it to the OPMN control list so that your system High Availability is monitored in a consistent way.

Refer to the *Oracle Process Manager and Notification Server Administrator's Guide* for more information.

4.3 Analyze High Availability using iHAT or Topology Viewer

Oracle Application Server Hi-Av Tool (iHAT) and Oracle Enterprise Manager Topology Viewer are real-time, graphical monitors based on topology information maintained by all the OPMN servers running in an Oracle Application Server Farm.

iHAT is a standalone java application that runs from a local display computer. iHAT connects to a single OPMN server to monitor and control an entire Oracle Application Server Farm.

The Topology Viewer is a J2EE application that can be deployed on Oracle Application Server Control to provide a real-time view of the application server processes managed by OPMN. You can view all the instances in a Farm, including clusters and their members.

Both iHat and the Topology Viewer display performance metrics for each component.

It is a good practice to verify the system High Availability prior to the publishing of its functionality to the end-user (pre-production testing). You can leverage either iHAT or Topology Viewer capabilities, to verify:

- what is installed and configured to be managed by OPMN across an entire Oracle Application Server Farm
- what is the current status of all the processes which are part of the Oracle Application Server Farm

By using various load simulators you can stress your High Availability topology and verify from the graphical tools that any component in the stack can be started, stopped, or disabled without any impact in the system response. You should verify that the number of requests sent to the system are all completed correctly even when one of the subcomponents is stopped. This verification is very simple from iHat or Topology Viewer. They both allow you to start, stop, or disable any process controlled by OPMN and they both show the number of completed requests. You should verify that the response to all requests are correct (this should be provided by your load simulator) and that the total number of requests is being completed. This information is available on iHat and Topology Viewer.

You can learn how to use Topology Viewer and iHat at the Oracle Application Server Utilities area on OTN (<http://otn.oracle.com>).

4.4 Use Metric Based Load Balancing to Tune High Availability

mod_oc4j is the module that routes requests from the Oracle HTTP Server to OC4J through the AJP 1.3 protocol. mod_oc4j provides the load balancing mechanism between Oracle HTTP Server and the OC4J containers that share a common OC4J mount point or are accessed through the same base URL. Thus, the high availability of a configuration where several OC4J nodes are being load balanced behind an Oracle HTTP Server, is totally dependent on the mod_oc4j configuration. A load balancing policy that mod_oc4j supports in Oracle Application Server 10g is the metric based policy. When this type of policy is specified in the mod_oc4j.conf file, mod_oc4j routes

requests as per the run time metrics from OC4J processes that indicate how much load is currently being placed on the OC4J process. For this to happen, each one of the OC4J nodes continuously sends metric values to mod_oc4j. mod_oc4j then designates the node for routing of the HTTP request. However, this policy incurs a penalty on performance since the OC4J nodes are obtaining and sending metrics on a real time basis.

If you want to maximize the usage of resources but without penalizing the performance of your system, you can use the metric based load balancing to tune your routing weights and then change the load balancing to a weighted round-robin protocol. Set up your load balancing protocol to metric-based and place the system in production or, if possible, in production simulation. Based on the number of the requests delivered to each node on a usage cycle you can make a good estimation of the weights that you should assign to each one of the nodes. Obviously, this tuning is more accurate when the number of requests and the load in each node due to external influences to the application being load balanced, do not change much during the usage cycle. You can then optimize the usage of each node with a better performance.

For more information refer to the *Oracle HTTP Server Administrator's Guide*.

Performance and Scalability

This chapter describes performance and scalability best practices. The topics include:

- [Section 5.1, "OracleAS Web Cache Best Practices"](#)

5.1 OracleAS Web Cache Best Practices

This chapter describes Oracle Application Server Web Cache best practices. The topics include:

- [Section 5.1.1, "Improve Performance, Scalability, and Availability"](#)
- [Section 5.1.2, "Planning and Deployment"](#)
- [Section 5.1.3, "OracleAS Web Cache Security"](#)
- [Section 5.1.4, "Configuring OracleAS Web Cache"](#)
- [Section 5.1.5, "Increasing Cache Hits"](#)
- [Section 5.1.6, "Invalidation and Expiration"](#)
- [Section 5.1.7, "Optimizing Response Times"](#)

5.1.1 Improve Performance, Scalability, and Availability

OracleAS Web Cache improves the scalability, performance and availability of e-business Web sites. Using OracleAS Web Cache, your applications benefit from higher throughput, shorter response times and lower infrastructure costs.

- Unlike legacy cache servers that only handle static data, OracleAS Web Cache combines caching, compression and assembly technologies to accelerate the delivery of both static and dynamically generated Web content.
- OracleAS Web Cache provides support for partial-page caching with Edge-Side Includes (ESI), personalization, and dynamic content assembly at the network edge. See [Section 5.1.5.4](#) for more information.
- OracleAS Web Cache includes patent-pending clustering functionality that increases capacity for content storage and ensures scalability and availability for cacheable content, even when a member cache experiences a failure or is taken offline for maintenance. See [Section 5.1.2.2](#) for more information.
- OracleAS Web Cache also provides back-end origin server load balancing, failover, and surge protection features that ensure consistent application performance and greater overall reliability.
- OracleAS Web Cache is designed to run with commodity hardware, reducing the cost.

Using OracleAS Web Cache and its ESI features, your business application performance can improve by several orders of magnitude with very little development effort. The return on investment is also significant, both in terms of developer resources (you no longer need to build your own dynamic caching solution) and hardware cost savings.

5.1.2 Planning and Deployment

The following sections describe best practices related to planning for and deploying OracleAS Web Cache:

- [Section 5.1.2.1, "Use Two CPUs and Consider Deploying on Dedicated Hardware"](#)
- [Section 5.1.2.2, "Cluster Cache Instances for Better Availability, Scalability, and Performance"](#)
- [Section 5.1.2.3, "Use a Network Load Balancer in Front of OracleAS Web Cache"](#)
- [Section 5.1.2.4, "Use OracleAS Web Cache Built-In Load Balancing for Availability and Scalability of Origin Servers"](#)
- [Section 5.1.2.5, "Deploy Caches in Remote Offices for Faster Response Times and Reduced WAN Traffic"](#)
- [Section 5.1.2.6, "Use the Latest Version"](#)
- [Section 5.1.2.7, "Test Application Upgrades and Patches to Ensure Existing Cache and Session Rules Still Function Correctly"](#)

5.1.2.1 Use Two CPUs and Consider Deploying on Dedicated Hardware

You can deploy OracleAS Web Cache on the same node as the application Web server or on a separate node. When making your decision, consider system resources, such as the number of CPUs. OracleAS Web Cache is designed to use one or two CPUs. Because OracleAS Web Cache is an in-memory cache, it is rarely limited by CPU cycles. Additional CPUs do not increase performance significantly. However, the speed of the processors is critical; use the fastest CPUs you can afford.

If other resources are competing with OracleAS Web Cache for CPU usage, then you should take the requirements of those resources into account when determining the number of CPUs needed. You can derive a significant performance benefit from OracleAS Web Cache running on the same node as the application Web server, although a separate node for OracleAS Web Cache is often optimal.

For a Web site with more than one OracleAS Web Cache instance, consider installing each instance on a separate two-CPU node, either as part of a cache cluster or as standalone instances. When OracleAS Web Cache instances are on separate nodes, you are less likely to encounter operating system limitations, particularly in network throughput. For example, two caches on two separate two-CPU nodes are less likely to encounter operating system limitations than two caches on one four-CPU node.

5.1.2.2 Cluster Cache Instances for Better Availability, Scalability, and Performance

To increase the availability, scalability, and performance of your Web site, you can configure multiple instances of OracleAS Web Cache to run as members of a cache cluster. A cache cluster is a loosely coupled collection of cooperating OracleAS Web Cache instances working together to provide a single logical cache.

Cache clusters provide failure detection and failover of caches, increasing the availability of your Web site. If a cache fails, other members of the cache cluster detect the failure and take over ownership of the cached content of the failed cluster member.

By distributing the Web site's content across multiple OracleAS Web Cache instances, more content can be cached and more client connections can be supported, expanding the overall capacity of your Web site and improving its performance.

For more information about cache clusters, see the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.2.3 Use a Network Load Balancer in Front of OracleAS Web Cache

Many customers deploy a single instance of OracleAS Web Cache in front of their application Web server farm. In such deployments, the OracleAS Web Cache acts as the virtual IP address for the application, in addition to providing caching and load balancing services. This deployment is both functionally sufficient and cost-effective for customers that do not require 100 percent application uptime. The OracleAS Web Cache is highly stable and, in the event of a failure, a process monitor will automatically restart the cache.

For customers who cannot tolerate a single point of failure, Oracle Corporation recommends that two or more nodes running OracleAS Web Cache be deployed behind a third-party hardware load balancing device. In turn, customers should use the built-in load balancing functionality in OracleAS Web Cache to distribute cache miss traffic over the application Web server farm, as described in [Section 5.1.2.4](#). Refer to the *Oracle Application Server Web Cache Administrator's Guide* for more information about using network load balancers.

5.1.2.4 Use OracleAS Web Cache Built-In Load Balancing for Availability and Scalability of Origin Servers

Situated between Web browser clients and the origin servers, OracleAS Web Cache includes built-in weighted load balancing and failover detection features to ensure that cache misses are directed to the most available, highest performing origin server in the application Web server farm. The cache supports both stateless and stateful load balancing mechanisms, including the use of cookies and URL parameters to maintain server affinity when required. (OracleAS Web Cache can be configured to generate its own session-binding cookie, allowing you to use sessions without having to modify your applications.)

In addition, OracleAS Web Cache maintains a pool of HTTP connections between the cache and the origin Web servers to reduce connection establishment overhead and improve cache miss performance.

To avoid a single point of failure, two or more nodes running OracleAS Web Cache should be deployed behind a third-party hardware load-balancing device. However, Oracle Corporation also recommends that customers use the built-in load balancing and failure detection functionality in OracleAS Web Cache to route cache miss requests to origin servers. Deploying additional load balancing hardware between the OracleAS Web Cache and origin server tiers is not recommended for the following reasons:

- **Cost:** Using another tier of load balancing hardware adds significant cost to a deployment, in part because these devices must also be deployed in pairs for high availability reasons.
- **Complexity:** Another tier of load balancing hardware is another set of systems to configure, manage, and troubleshoot.
- **Features:** OracleAS Web Cache includes patent-pending performance assurance and surge protection features that enable customers to sustain higher loads with

less application and database server hardware. These features depend on the capacity-based load balancing algorithms in OracleAS Web Cache.

For more information on load balancing, performance assurance and surge protection functionality, see the *Oracle Application Server Web Cache Administrator's Guide* and various technical white papers on OracleAS Web Cache available on OTN (<http://otn.oracle.com>).

5.1.2.5 Deploy Caches in Remote Offices for Faster Response Times and Reduced WAN Traffic

OracleAS Web Cache offers hierarchical caching features that enable customers to easily create Content Delivery Networks (CDNs). For high availability and performance, many Internet businesses mirror their Web sites in strategic geographical locations. Caching is an excellent low-cost alternative to full-scale mirroring. Caching may also be used to serve local markets in order to shorten response times to these markets, and to reduce bandwidth and rack space costs for the content provider. Within the corporate intranet, so-called "Enterprise" CDNs (eCDNs) provide shorter response times for branch office users of e-business applications. Compared to application mirroring and database replication, eCDN is a more manageable and cost-effective model of distributed computing. Using OracleAS Web Cache, customers can distribute the content assembly and delivery functions of their applications to key network access points, while maintaining centralized management of application logic and data.

In setting up an eCDN, customers typically deploy OracleAS Web Cache in each branch office data center as well as in the central office where the application and database are maintained. For example, a U.S.-based company might deploy instances of OracleAS Web Cache in its U.S., Japanese, and Australian offices. The central cache residing in the U.S. serves as the origin server for the caches in Japan and Australia. Using various commercially available DNS routing techniques, requests are handled by the cache that is closest to the end user. A Web browser request made by a Japanese employee, for instance, is handled by the cache instance in Japan, thereby reducing WAN traffic and eliminating long-haul network latencies. In a distributed cache hierarchy, the central cache is aware of the branch office caches. As a result, any content invalidation messages sent to the central cache automatically propagate to the remote caches. This invalidation propagation feature ensures content consistency across the CDN and simplifies the management of cache hierarchies.

5.1.2.6 Use the Latest Version

Make sure that you are using the latest version of OracleAS Web Cache, including any available patches. Use the Upgrade tool to upgrade the old configuration to the new configuration. To check for the latest patches, go to Oracle *MetaLink*.

5.1.2.7 Test Application Upgrades and Patches to Ensure Existing Cache and Session Rules Still Function Correctly

Although there is a growing trend to specifying the caching rules dynamically with the Surrogate-Control response header, some sites continue to use OracleAS Web Cache Manager for configuring the rules statically. Typically, this configuration is done at the start of the deployment cycle. After adequate testing in a staging area to validate the rules, OracleAS Web Cache is deployed in a production environment. Problems may arise when the backend application is upgraded for patches or with new versions and some or all of the earlier statically configured rules become not applicable and void. For example, if a site uses a session-related caching rule and, after applying a patch, the name of the session cookie or session-embedded URL parameter changes, all the

pages related to that rule will no longer be cacheable, resulting in poor performance for the site.

When applying application upgrades and patches, it is important to understand the extent of the application changes and then verify and tune the related caching rules in OracleAS Web Cache. By periodically checking the cache-hit percentage and ensuring that it remains more or less constant, you can guard against unexpected behavior. Whenever there is a major change in the database or the mid-tier layer, such as for upgrades or application patches, you should validate caching rules much the same way as you did during the initial deployment cycle, including, but not limited, to using debug-level event logging. And if possible, include OracleAS Web Cache in your application regression test cycle.

5.1.3 OracleAS Web Cache Security

The following sections describe best practices related to security issues:

- [Section 5.1.3.1, "Route All HTTP and HTTPS Traffic Through OracleAS Web Cache"](#)
- [Section 5.1.3.2, "Secure Administration, Invalidation, and Statistics Monitoring Using HTTPS"](#)
- [Section 5.1.3.3, "Use Web Caching to Help Defend Against Denial-of-Service Attacks"](#)
- [Section 5.1.3.4, "Change Passwords Frequently"](#)

5.1.3.1 Route All HTTP and HTTPS Traffic Through OracleAS Web Cache

In general, you should route all HTTP and HTTPS requests through OracleAS Web Cache. Ensure documents, especially HTTPS documents, are sent to authorized users through careful use of caching rules.

Depending on the application, you may or may not want requests for secure pages to go through the cache. If every HTTPS request is a non-cacheable page that is unique for each user session or is too sensitive for caching a copy, then route this traffic directly to the origin server. Because no traffic will be cached in this case, routing traffic to the origin server avoids extra encryption/decryption processing time at the OracleAS Web Cache layer.

5.1.3.2 Secure Administration, Invalidation, and Statistics Monitoring Using HTTPS

The default configuration for OracleAS Web Cache does not enable HTTPS for administration, invalidation, or statistics monitoring requests. Instead, these ports are configured for HTTP basic authentication. On an insecure network, the passwords for the `ias_admin`, `administrator`, and `invalidator` users can be decoded if they are sniffed out of the HTTP traffic. To avoid breach of security information for unprotected and insecure networks, set the communication protocol to HTTPS for administration and invalidation operations in the Operation Ports page (Ports > Operations Ports) of OracleAS Web Cache Manager.

5.1.3.3 Use Web Caching to Help Defend Against Denial-of-Service Attacks

OracleAS Web Cache was designed to provide high performance, reliability and scalability on low-cost commodity hardware. A single OracleAS Web Cache instance can be configured to support thousands of concurrent inbound HTTP connections. The throughput (requests/second) that a single cache instance can sustain scales linearly with CPU speed. Additionally, OracleAS Web Cache fully supports the HTTP/1.0 and HTTP/1.1 header fields, including `Keep-Alive`. The `Keep-Alive` header field

reduces the frequency of connection establishment and improves performance and scalability under heavy load.

When clustered, the capacity—the amount of content stored in RAM—of the cache tier scales linearly, as well. Cache clustering achieves high availability by failing over among cluster nodes, as well as high scalability by utilizing memory and processing power of multiple inexpensive computing hardware units in parallel.

Not surprisingly, many customers have reported the successful use of OracleAS Web Cache to prevent distributed and single-source denial-of-service attacks. Denial-of-service attacks attempt to prevent access to Web sites either by flooding sites with traffic volumes that surpass throughput and connection capacities or by sending malicious requests intended to exploit software flaws that cause servers to crash. By caching responses to denial-of-service requests, OracleAS Web Cache helps address the throughput and scalability limitations of Web sites, while creating a crucial barrier between malicious queries and a site origin application and database servers.

The guidelines for configuring connection limits, caching rules, and cache clusters are described in the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.3.4 Change Passwords Frequently

You should change the administration and invalidation passwords for OracleAS Web Cache from the default passwords during or after installation. You should change the passwords on a regular basis, at least once a month.

Consider using a secure password that is at least 8 characters long. At least one of the characters must be a number.

5.1.4 Configuring OracleAS Web Cache

The following sections describe best practices related to configuring OracleAS Web Cache:

- [Section 5.1.4.1, "Use the OracleAS Web Cache Manager to Avoid Configuration Problems"](#)
- [Section 5.1.4.2, "Configure Enough Memory"](#)
- [Section 5.1.4.3, "Allocate Sufficient Network Bandwidth"](#)
- [Section 5.1.4.4, "Set a Reasonable Number of Network Connections"](#)
- [Section 5.1.4.5, "Create Custom Error Pages"](#)

5.1.4.1 Use the OracleAS Web Cache Manager to Avoid Configuration Problems

The OracleAS Web Cache Manager is a graphical user interface for administering, configuring, and managing caches. Oracle Corporation recommends that you use it rather than manually editing the configuration files to make configuration changes.

In cache clusters, it is especially important to use OracleAS Web Cache Manager to modify the configuration and to propagate it to the other cluster members. If you manually edit the configuration files, you may encounter problems.

For example, if you do not set the site name properly for each cluster member, identical objects could be cached in each cluster member, but the objects may be known by a different name in the different caches. For example, you could have the same object cached as:

```
mysite:7777:document1.html  
myste:7777:document1.html
```

If the site names are not consistent, the objects with the variant site name will not be invalidated even if you use invalidation propagation.

5.1.4.2 Configure Enough Memory

To avoid swapping objects in and out of the cache, it is crucial to configure enough memory for the cache. Generally, the amount of memory (maximum cache size) for OracleAS Web Cache should be set to at least 256 MB.

To determine the maximum amount of memory required, take the following steps:

1. Determine what objects you want to cache, how many are smaller than 2 KB and how many are larger than 2 KB. Determine the average size of the objects that are larger than 2 KB. Determine the expected peak load—the maximum number of objects to be processed concurrently. Determine the average number of connections.
2. Calculate the amount of memory needed based on the number and size of the objects. The *Oracle Application Server Web Cache Administrator's Guide* provides a formula to use in calculating the amount of memory needed to cache your objects.

5.1.4.3 Allocate Sufficient Network Bandwidth

When you use OracleAS Web Cache, make sure that each node has sufficient network bandwidth to accommodate the throughput load. Otherwise, the network may be saturated even though OracleAS Web Cache has additional capacity. For example, if your application generates more than 100 megabits of data per second, 10/100 Megabit Ethernet will likely be saturated.

If the network is saturated, consider using Gigabit Ethernet rather than 10/100 Megabit Ethernet. Gigabit Ethernet provides the most efficient deployment scenario to avoid network collisions, retransmissions, and bandwidth starvations.

Additionally, consider using two separate network interface cards (NIC): one for incoming client requests and one for requests from the cache to the application Web server.

If system monitoring tools reveal that the network is under utilized and throughput is less than expected, check whether or not the CPUs are saturated.

5.1.4.4 Set a Reasonable Number of Network Connections

It is important to specify a reasonable number for the maximum connection limit for the OracleAS Web Cache server. If you set a number that is too high, performance can be affected, resulting in slower response time. If you set a number that is too low, fewer requests will be satisfied. You must strike a balance between response time and the number of requests processed concurrently.

For information about setting the number of network connections, see the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.4.5 Create Custom Error Pages

By default, OracleAS Web Cache ships with and is configured to serve the following error pages:

- `network_error.html`: This file is served when OracleAS Web Cache encounters network problems while connecting, sending, or receiving a response from an origin server for a cache miss request.

- `busy_error.html`: This file is served when origin server capacity has been reached.
- `esi_fragment_error.txt`. This file is served when OracleAS Web Cache is unable to fetch the `src` specified in an `<esi:include>` tag and the `alt` attribute, `onerror` attribute, or the `try | attempt | except` block are either not present or fail.

The ESI language, which provides three specific elements for fine-grain control over content assembly in error scenarios:

- The `alt` attribute of the `<esi:include>` tag
- The `onerror` attribute of the `<esi:include>` tag
- The `try | attempt | except` block
- Default page in place of the fragment

When the fragment specified for the `src` attribute of the `<esi:include>` tag cannot be fetched, the fragment specified with the `alt` attribute is used as an alternative. If the `alt` attribute is not used or the fragment cannot be fetched, the `onerror` attribute is used. The `onerror` attribute is used before the `try | attempt | except` block. If the `try | attempt | except` block does not exist, the exception handling is propagated to the parent or template page. If all the ESI language controls fail, OracleAS Web Cache displays a default page in place of the fragment.

For a production environment, Oracle Corporation advises that you modify the defaults or create entirely new error pages to be consistent with other error pages generated by your application.

For information on creating or modifying default error pages, see the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.5 Increasing Cache Hits

The following sections provide tips in increasing the cache hit rate:

- [Section 5.1.5.1, "Use Cookies and URL Parameters to Increase Cache Hit Ratios"](#)
- [Section 5.1.5.2, "Use Redirection to Cache Entry Pages"](#)
- [Section 5.1.5.3, "Use Surrogate-Control Headers Instead of Caching Rules"](#)
- [Section 5.1.5.4, "Use Partial Page Caching Where Possible"](#)
- [Section 5.1.5.5, "Use ESI Variables for Improved Cache Hit Ratio for Personalized Pages"](#)
- [Section 5.1.5.6, "Use the <esi:environment> Tag for Authentication or Authorization Callbacks"](#)
- [Section 5.1.5.7, "Use esi:inline and esi:include Tags Appropriately"](#)
- [Section 5.1.5.8, "Leverage JESI Over Hand-Generating the ESI Tags"](#)

5.1.5.1 Use Cookies and URL Parameters to Increase Cache Hit Ratios

OracleAS Web Cache can cache different versions of a document with the same URL based on request cookies or headers. To use this feature, applications may need to implement some simple change, such as creating a cookie or header that differentiates the pages.

On the opposite side of the spectrum, some applications contain some insignificant URL parameters that lead to different URLs representing essentially the same content. If the documents are cached under their full URLs, then the cache-hit ratio becomes

very low. You can configure OracleAS Web Cache to ignore the non-differentiating URL parameter values when composing the "cache key" for documents, so a single document will be cached for different URLs, greatly increasing cache hit ratios.

Sometimes the content for a set of pages is nearly identical, but not exactly the same. For example, the pages may contain hyperlinks composed of the same URL parameters with different session-specific values, or they may include some personalized strings in the page text, such as welcome greetings and shopping cart totals. In this case, OracleAS Web Cache can still store one single copy of the document with placeholders for the embedded URL parameters and/or the personalized strings, and dynamically substitute the correct values into the placeholders when serving the document to clients.

You can also control whether a cached document can be served to a client based on its session state.

For more information on multiple version documents, sessions, ignoring URL parameter values, simple personalization, and how to control whether a cached document can be served to a request based on sessions, see Chapter 2, "Caching Concepts," in the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.5.2 Use Redirection to Cache Entry Pages

For some popular site entry pages, such as "/", that typically require session establishment, session establishment effectively makes the page non-cacheable to all new users without a session.

To cache these pages while preserving session establishment, you can create a blank page that provides session establishment for all initial requests and redirects to the real popular page. This way, subsequent redirected requests to the popular page will carry the session, enabling the popular page to be served out of the cache.

For more information on configuring caching rules for pages requiring session establishment, see Chapter 9, "Creating Caching Rules," in the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.5.3 Use Surrogate-Control Headers Instead of Caching Rules

There are two ways to specify the caching properties of an HTTP response using OracleAS Web Cache. You can use one or both of the following:

- Administrators can configure caching rules using the OracleAS Web Cache Manager interface.
- Application developers can set caching policies via the Surrogate-Control response header. If a given property is set in both a response header and the configuration, the value set by Surrogate-Control overrides rules specified in the configuration.

Although caching rules support the setting of more properties than the Surrogate-Control header, it is generally more manageable to set properties in the Surrogate-Control header whenever possible. For example, if you need to set the expiration policy and the multiple-version property for a document, it is preferable to use the Surrogate-Control header.

If you define many different categories of cacheable and non-cacheable documents, you need to carefully define rule selectors and rule priorities so that the appropriate rule is used for any document. Because a Surrogate-Control response header is only associated with one response and overrides the configuration, the properties set in Surrogate-Control will not be mistakenly replaced by other configuration rules or

newly created configuration rules. If you are creating new applications, consider building in Surrogate-Control response headers.

On the other hand, sometimes the configuration approach is more convenient. If a small number of rules are sufficient to describe all the caching properties of all documents that OracleAS Web Cache can receive from an origin server, then editing the configuration using the OracleAS Web Cache Manager administration interface may be simpler than generating Surrogate-Control headers for many documents.

Often, a combination of the two approaches is best.

5.1.5.4 Use Partial Page Caching Where Possible

Many Web pages, such as portal pages, are composed of fragments with unique caching properties. For these pages, full-page caching is not feasible. However, OracleAS Web Cache provides a partial page caching feature that enables each Web page to be divided into a template and multiple fragments that can, in turn, be further divided into templates and lower-level fragments.

Each fragment or template is stored and managed independently; a full page is assembled from the underlying fragments upon request. Fragments can be shared among different templates, so that common fragments are not duplicated to waste cache space. Sharing can also greatly reduce the number of updates required when fragments expire. Depending on the application, updating a fragment can be cheaper than updating a full page. In addition, each template or fragment may have its own unique caching policies such as expiration, validation, and invalidation, so that each fragment in a full Web page can be cached as long as possible, even when some fragments are not cached or are cached for a much shorter period of time.

For example, a Portal page may include stock quotes that expire in 20 minutes, news that expires in three hours, and rotating ad banners that should not be cached. To serve consistent content, traditional full-page caches need to update the entire page at the highest change frequency of all its fragments. With partial page caching, particular fragments can be updated, instead of the entire page.

OracleAS Web Cache uses Edge Side Includes (ESI) to achieve flexible partial-page caching. ESI is a simple markup language for partial-page caching. Applications can mark up HTTP responses with two different kinds of tags, `<esi:inline>` and `<esi:include>`, that define the fragment/template structure in the response.

5.1.5.5 Use ESI Variables for Improved Cache Hit Ratio for Personalized Pages

Personalized information often appears in Web pages, making them unique for each user. For example, many Web pages contain tens or hundreds of hyperlinks embedding application session IDs.

OracleAS Web Cache allows application developers to use variables in an ESI template. Because variables can be resolved to different pieces of request information or response information, the uniqueness of templates and fragments can be significantly reduced when personal information abounds.

There are two kinds of ESI variables: **request variables** and **response variables**. When an ESI template is assembled, a request variable is instantiated to a piece of request information such as a query string parameter, a cookie, or an HTTP header. For example, when a request for a dynamic page carries an application session ID in a query string parameter, this page may contain many hyperlinks with ESI request variables accessing this session ID, so that generated hyperlinks can carry the session ID into the next clicked page.

A response variable is similar to a request variable, except that its value comes not from the request, but from a special fragment called ESI environment. An **ESI environment** is a type of fragment with a response that defines a set of variables that can be accessed by response variable occurrences in the enclosing template. The tag itself does not contribute to the final assembled output. For example, a dynamic page with a calendar may need to present personal appointments that cannot be stored in Web browser cookies due to cookie size limits. The application can instead refer to a “profile” environment fragment in the template, in effect referring to all appointments in the environment without making separate requests for each appointment. In addition, you can merge multiple small fragments into one environment, so that each fragment can be referenced through response variable instantiation. This reduces storage and retrieval overhead similarly.

5.1.5.6 Use the <esi:environment> Tag for Authentication or Authorization Callbacks

Some applications protect certain Web pages with authentication, authorization, or session validation in the HTTP request. Even though the page content can be cached, every HTTP request must be authenticated, authorized, or validated by the origin server. For these pages, it is not appropriate to cache the full page. While it is possible to utilize JavaScript to achieve authentication, authorization, and validation through a separate HTTP request, the <esi:environment> tag provides a better solution to this problem.

If a page has cacheable content but requires mandatory authentication, authorization, and session validation, you can enclose an <esi:environment> tag in the page referencing a non-cacheable environment, and cache the enclosing page. When the cached page is requested, an HTTP request that specifies the environment will always be sent to the origin server, making a callback to the application. In this callback request, if you want to validate a cookie in the enclosing page request for session validation, authorization, or authentication, specify the <esi:environment> tag to include that cookie in its request. You can also include other information from the page request in the environment request.

If authentication, authorization, and validation are passed, your application should return HTTP status code 200 and any ESI environment response. OracleAS Web Cache proceeds to finish assembling the page. If the authentication, authorization, or validation fail, your application should return an appropriate HTTP status code (at or above 500 to denote a server error, or between 400 and 499 to denote a client request error). Then, OracleAS Web Cache will recognize that this environment has failed, and resort to standard ESI exception handling to abort this page or output an alternative error page or login page.

For more information about using the <esi:environment> tag or implementing ESI exception handling, see Chapter 15, “Edge Side Includes (ESI) Language Tags,” in the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.5.7 Use esi:inline and esi:include Tags Appropriately

The <esi:inline> and <esi:include> tags enable applications to adopt ESI page fragmentation and assembly.

If a partial page fragment can be fetched independently, such as with a URLPortlet, refer to it with an <esi:include> tag. This enables OracleAS Web Cache to fetch the fragment independently of the rest of the page or fragments. However, if the fragment cannot be generated independently of the surrounding page, use the <esi:inline> tag, which enables OracleAS Web Cache to cache the inline fragment and re-use it in different contexts.

5.1.5.7.1 Using esi:inline for Non-Fetchable Fragmentation Most existing applications are designed to output an entire Web page to HTTP requests. These fragments and templates are **non-fetchable**, meaning they are not to be fetched independently from the origin server. If a cache needs any of these fragments or templates, the corresponding full Web page must be requested. To use ESI page assembly for non-fetchable fragments, an application can output the full-page response just as it does normally, with the exception that at the beginning and the end of each fragment, an `<esi:inline>` tag is inserted with a fragment name to demarcate the fragment. OracleAS Web Cache stores the enclosed portions as separate fragments and the original page as page templates without the enclosed fragments. Fragments are shared if their names are identical.

When an application uses non-fetchable `<esi:inline>` fragments, the full page must be requested for every cache miss. At first, it can appear that there is no apparent cache benefit for cache misses. However, non-fetchable `<esi:inline>` fragments improve overall caching by:

- Increasing the cache-hit ratio because shared cacheable fragments can be extracted and stored only once, reducing the size of the dynamic portion. A reduced space requirement results in a higher cache-hit ratio than full page caching.
- Reducing cache update frequency. Dynamic shared cacheable fragments require only one update. For example, a shared stock market fragment may expire much more frequently than any other parts of the page. With `<esi:inline>` fragmentation, only one cache update of any full page containing this fragment is enough to bring all full pages sharing this fragment current. Therefore, even non-fetchable `<esi:inline>` fragments can significantly reduce cache update frequency. The cost reduction is proportional to the degree of sharing.

The `<esi:inline>` tag is primarily intended for pages with cacheable fragments. If a page contains non-cacheable, non-fetchable fragments, then the use of `<esi:inline>` is not recommended. However, the update of this full page may still offer benefit if it contains some cacheable fragments that are shared with other pages.

5.1.5.7.2 Using esi:include for Fetchable Fragmentation The `<esi:include>` tag is another way to define fragments and templates in an HTTP output for dynamic content caching and assembly. The page including an `<esi:include>` tag is a template that refers to the defined fragment. However, because it has some key differences from `<esi:inline>`, its applicable scenarios are very different:

- An `<esi:include>` tag in a template only defines the reference to a fragment. It does not enclose an embedded fragment directly in the template.
- A fragment referenced by an `<esi:include>` tag must always be independently fetchable by HTTP or HTTPS.

The requested URL is the same as the fragment name. In contrast, an `<esi:inline>` tag name only identifies the uniqueness of the fragment and is not used to fetch the actual content. The attribute defining the fragment name in `<esi:include>` fragment is `src` instead of `name`.

There are at least two scenarios where using `<esi:include>` tags is beneficial:

- Some applications, such as a Web portal, assemble content from external sources. The application only provides a template that is used to fetch various fragments from third-party sources. In this case, the `<esi:include>` tags fetch and assemble directly, reducing one layer of redundancy.
- Some applications offer faster responses for template-only or fragment-only requests than full-page requests that use `<esi:inline>` tags. If `<esi:include>`

is used for page fragmentation and assembly, OracleAS Web Cache can miss only on the templates or fragments when most or all fragments are already cached, saving effective cache miss cost. In many cases, it is also valuable to cache the personalized templates because these seldom change.

5.1.5.8 Leverage JESI Over Hand-Generating the ESI Tags

In dynamic applications, you can use ESI tags for better caching in two ways: hand generating the tags, or using Edge Side Includes for Java (JESI) tags.

The JESI specification is a specification and custom JSP tag library that developers can use to automatically generate ESI code. JESI facilitates the programming of Java ServerPages (JSPs) using ESI. While developers can always use ESI tags directly within their JSP code, JESI represents an easy way to express the modularity of JSPs and the caching of those modules, without requiring developers to learn a new programming syntax. JESI generates the appropriate ESI tags and headers in the JSP output that instruct ESI processors, such as OracleAS Web Cache, to cache (or not) templates and fragments for the appropriate duration. JESI also facilitates the partial execution of JSPs when an ESI processor requests fragments and templates.

5.1.6 Invalidation and Expiration

The following sections provide tips about invalidating and expiring cache objects:

- [Section 5.1.6.1, "Use Basic Invalidation for Single Objects"](#)
- [Section 5.1.6.2, "Use Substring Matching for Multiple Objects in Advanced Invalidations"](#)
- [Section 5.1.6.3, "Build Programmatic Invalidation Into Application Logic"](#)
- [Section 5.1.6.4, "Combine Invalidation and Expiration Policies"](#)
- [Section 5.1.6.5, "Use Invalidation Propagation in Clusters and Hierarchies"](#)
- [Section 5.1.6.6, "Tune Invalidation Performance Using Indexes"](#)

5.1.6.1 Use Basic Invalidation for Single Objects

When you need to invalidate one object in the cache, send a basic rather than an advanced invalidation request to avoid cache traversal. Advanced invalidation requests should be reserved for invalidation of multiple objects.

To send a basic invalidation request, use the Basic Content Invalidation page (Operations > Basic Content Invalidation) in OracleAS Web Cache Manager or specify the BASICSELECTOR element in a manual invalidation request.

For example, the following request invalidates a document exactly matching /contacts/contacts.html using the BASICSELECTOR element:

```
<?xml version="1.0"?>
<!DOCTYPE INVALIDATION SYSTEM
"internal:///WCSinvalidation.dtd">
<INVALIDATION VERSION="WCS-1.1">
  <OBJECT>
    <BASICSELECTOR URI="http://www.company.com:80/contacts/contacts.html"/>
    <ACTION REMOVALTTL="0"/>
  </OBJECT>
</INVALIDATION>
```

5.1.6.2 Use Substring Matching for Multiple Objects in Advanced Invalidations

When multiple objects share a common URL, request POST body, or an embedded URL parameter, you can express the common elements in multiple ways:

- Common URL:
 - Use the **URL Expression** field in the Advanced Content Invalidation page.
 - Use the URIEXP attribute of the ADVANCEDSELECTOR element.
 - Use the ADVANCEDSELECTOR element OTHER element with a NAME attribute value of URI.
- Common Request POST Body:
 - Use the **HTTP Method** and **POST Body Expression** fields in the Advanced Content Invalidation page
 - Use the ADVANCEDSELECTOR element METHOD and BODYEXP attributes
 - Use the ADVANCEDSELECTOR element OTHER element with a NAME attribute value of BODY.
- Common Embedded URL Parameter:
 - Use the ADVANCEDSELECTOR element OTHER element with a NAME attribute value of QUERYSTRING_PARAMETER.

For the quickest invalidation, Oracle Corporation recommends using the OTHER element to specify a substring for literal matching rather than regular expression for pattern matching.

To send an advanced invalidation request, use the Advanced Content Invalidation page (Operations > Advanced Content Invalidation) in OracleAS Web Cache Manager or specify the ADVANCEDSELECTOR element in a manual invalidation request.

To send an advanced invalidation request with substring matching:

1. Specify the OTHER element in a manual invalidation request that uses the ADVANCEDSELECTOR element.
2. Specify the NAME attribute to use a value of URI, BODY, or QUERYSTRING_PARAMETER.
3. Specify the TYPE attribute to use a value of SUBSTRING.

For example, the request in the following example searches for any documents below `http://wc-cluster.us.oracle.com:1100/pls/portal/!PORTAL.wwpro_app_provider.execute_portlet/595897563/`, that match the following criteria:

- The HTTP request method is an HTTP POST request method.
- The URI is `showPortlet.Show`.
- The HTTP requests headers are `x-oracle-cache-user` and `x-oracle-cache-subid`.
- The embedded URL parameters are `_portlet_id` and `_provider_id`.

```
<?xml version="1.0"?>
<!DOCTYPE INVALIDATION SYSTEM "http://www.oracle.com/webcache/90200/WCSinvalidation.dtd">
<INVALIDATION VERSION="WCS-1.1">
  <OBJECT>
    <ADVANCEDSELECTOR
      URIPREFIX="/pls/portal/!PORTAL.wwpro_app_provider.execute_portlet/595897563/"
      HOST="wc-cluster.us.oracle.com:1100" METHOD="POST">
      <OTHER NAME="QUERYSTRING_PARAMETER" TYPE="SUBSTRING"
```

```

        VALUE="_portlet_id=2"/>
<OTHER NAME="QUERYSTRING_PARAMETER" TYPE="SUBSTRING"
        VALUE="_provider_id=595897563"/>
<HEADER NAME="x-oracle-cache-user" VALUE="PORTAL"/>
<HEADER NAME="x-oracle-cache-subid" VALUE="1"/>
<OTHER NAME="BODY" TYPE="SUBSTRING" VALUE="_language=EN-US"/>
<OTHER NAME="URI" TYPE="SUBSTRING" VALUE="showPortlet.Show"/>
</ADVANCEDSELECTOR>
<ACTION REMOVALTTL="0"/>
<INFO VALUE="Invalidate an old portlet based on portlet ID and provider ID"/>
</OBJECT>
</INVALIDATION>

```

5.1.6.3 Build Programmatic Invalidation Into Application Logic

OracleAS Web Cache is designed for caching highly dynamic Web pages. There are several ways to safeguard the correctness of the cached content. For those cached pages with content that changes following unpredictable actions by Web site users, the most effective way to ensure correct content is to build programmatic invalidation into application logic.

The application Web server and database are two areas that may benefit from embedded programmatic invalidation. On the application Web server, you can build invalidation into CGIs, JSPs and Java servlets using the OracleAS Web Cache Java invalidation API, `jawc.jar`. The `jawc.jar` file is located in the `$ORACLE_HOME/webcache/jlib` directory on UNIX and `ORACLE_HOME\webcache\jlib` directory on Windows. For example, page A displays information about a certain bike in stock. This page is cacheable. Page B provides users a way to reserve one bike for purchase. On the mid-tier, there is a Java servlet or JSP to service page B. To make this servlet cache-aware, use `jawc.jar` to invalidate page A.

Similarly, you can build invalidation into PL/SQL applications using the OracleAS Web Cache PL/SQL invalidation API `wxvutil.sql` and `wxvappl.sql`. This way, developers can embed the invocation of invalidation PL/SQL procedure calls into the PL/SQL Web page. The APIs, `wxvutil.sql` and `wxvappl.sql`, are located in the `$ORACLE_HOME/webcache/examples` directory on UNIX and `ORACLE_HOME\webcache\examples` directory on Windows.

To facilitate the caching of JSPs, developers can use the JESI custom tag library included with OC4J and Oracle JDeveloper. One of the tags, `<jesi:invalidate>`, enables programmatic invalidation when the JSP engine processes a page containing this tag. For more information about JESI, see the *Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference*.

Alternatively, developers can tie invalidation logic to database updates. In the same bike example, you can use a PL/SQL invalidation procedure call to invalidate pages that rely on inventory data stored in the database. In other words, you can apply a database trigger to the row or table containing information about the bike.

For more information about invalidation, see Chapter 11, "Sending Invalidation Requests," in the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.6.4 Combine Invalidation and Expiration Policies

With expiration rules, cached objects are marked as invalid after a certain amount of time in the cache. Expirations are useful if it can be accurately predicated when content will change on an origin server or database. To prevent documents from remaining in the cache indefinitely, Oracle Corporation recommends creating expiration rules for all cached documents.

With invalidation requests, an HTTP POST message specifies which objects to mark as invalid. Invalidation requests are intended for less predictable, more frequently changing content. Send invalidation requests when you know objects have been refreshed on the origin server. Invalidation policies can be automated, as described in [Section 5.1.6.3](#).

5.1.6.5 Use Invalidation Propagation in Clusters and Hierarchies

In a cache cluster or hierarchy, you can send invalidation messages to one cache, and that cache propagates the invalidation messages to the other caches in the cluster or hierarchy.

The benefits of invalidation propagation include data consistency across cluster members and ease of use for the administrator.

However, under the following circumstances, you may want to disable invalidation propagation and send the invalidation messages to each individual member of the cluster:

- When the cluster membership is in flux. For example, as you begin deployment, you may have made changes to cluster members but have not yet propagated the configuration changes to all members. In this case, the invalidation messages are not propagated to the members with different configurations.
- If you do not want to invalidate data for all cluster members. For example, because of time zone differences, you want to send invalidation messages to only some of the cluster members at one time.

Note that if you do not invalidate data for all cluster members, the cached data may become inconsistent. In addition, cluster members may serve stale data, not only in response to requests from clients, but also in response to requests from their peers.

For more information about invalidation propagation, see the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.6.6 Tune Invalidation Performance Using Indexes

To improve performance of advanced invalidation requests that use `QUERYSTRING_PARAMETER` to match objects with the same embedded URL parameters, manually create an invalidation index. Because an invalidation index creates more depth to an internal invalidation tree used by OracleAS Web Cache, OracleAS Web Cache is able to categorize objects in the cache for faster lookup and traversal. To use this feature for a URL or a request POST body, consider moving the URI or BODY value to `QUERYSTRING_PARAMETER` instead.

To specify an invalidation index, add an `INVALIDATIONINDEX` element to the `webcache.xml` file that specifies the substring value used for `QUERYSTRING_PARAMETER`:

```
<SECURITY>
...
</SECURITY>
<INVALIDATIONINDEX>
  <INDEXPARAM VALUE="VALUE_of_QUERYSTRING_PARAMETER" />
  <INDEXPARAM VALUE="VALUE_of_QUERYSTRING_PARAMETER" />
</INVALIDATIONINDEX>
<WATCHDOG ENABLE="YES|NO" />
```

Oracle Corporation recommends using invalidation indexes to create more depth to flat directory structures.

For more information about invalidation, see Chapter 11, "Sending Invalidation Requests," in the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.7 Optimizing Response Times

The following sections describe best practices in optimizing response times:

- [Section 5.1.7.1, "Optimize Response Time By Tuning Origin Server and OracleAS Web Cache Settings"](#)
- [Section 5.1.7.2, "Improve Response Times and Reduce Network Bandwidth With Compression"](#)
- [Section 5.1.7.3, "Use Only Warning or Notification Logging Levels to Conserve Resources"](#)

5.1.7.1 Optimize Response Time By Tuning Origin Server and OracleAS Web Cache Settings

If you have not configured the origin server or the cache correctly, response time may be slower than anticipated. If the origin server is responding more slowly than expected or if the origin server is not responding to requests from the cache because it has reached its capacity, check the origin server and the OracleAS Web Cache settings.

First, check the values of the following settings in the application Web server configuration file (`httpd.conf`). (These particular parameter names are specific to the Oracle HTTP Server.)

- **KeepAlive**: Whether to allow persistent connections. Persistent connections allow a client to send multiple sequential requests through the same connection.
Make sure `KeepAlive` is enabled. This can improve performance because the connection is set up only once and is kept open for subsequent requests from the same client.
- **KeepAliveTimeout**: The time a connection is left open to wait for the next request from the same client. If requests are primarily from OracleAS Web Cache, you can set this value fairly high. A reasonable value is 30 seconds.
- **MaxKeepAliveRequests**: The maximum number of requests to allow during a persistent connection. Set to 0 to allow an unlimited number of requests.
- **MaxClients**: The maximum number of clients that can connect to the application Web server simultaneously.

If `KeepAlive` is enabled for the application Web server, you may require more concurrent `httpd` server processes, and you may need to set the `MaxClients` directive to a higher value.

If client requests have a short response time, you may be able to improve performance by setting `MaxClients` to a lower value. However, when this value is reached, no additional processes will be created, causing other requests to fail.

The `MaxClients` limit on the application Web server should be greater than or equal to the application Web server capacity as set through the OracleAS Web Cache Manager.

See the *Oracle Application Server 10g Performance Guide* for more information about setting limits for the origin server.

Then, if the origin server is still busier than anticipated, it may mean that the cache cannot process the requests and is routing more requests to the origin server. Check the following OracleAS Web Cache settings:

- The origin server capacity, as set through the OracleAS Web Cache Manager. See Chapter 7, "Basic Setup and Configuration," in the *Oracle Application Server Web Cache Administrator's Guide* for information about setting origin server capacity.
- The number of cache connections (Maximum Incoming Connections) in the Resource Limits page (Properties > Resource Limits) of the OracleAS Web Cache Manager. If you have specify too low of a limit and the cache reaches the limit, OracleAS Web Cache may drop connections.
- The memory size for the cache (Maximum Cache Size) in the Resource Limits page. See Chapter 7, "Basic Setup and Configuration," in the *Oracle Application Server Web Cache Administrator's Guide* for information about calculating the memory size.
- The cache cluster capacity. In a cluster, if cluster capacity is too low, a cache may not receive a response for owned content from a peer cache within the specified interval. As a result, the request is sent to the origin server.
- Keep-Alive Timeout in the Network Timeouts page (Properties > Network Timeouts). This is the amount of time a network connection is left open after OracleAS Web Cache sends a response to a Web browser. Keep-Alive allows an HTTP client to send multiple requests to OracleAS Web Cache using the same network connection. By default, the connection is left open for five seconds, which is typically enough time for the Web browser to send subsequent requests to OracleAS Web Cache using the same connection.

If the network between the Web browser and OracleAS Web Cache is slow, consider increasing the timeout, perhaps up to 30 seconds.

- Origin Server Timeout in the Network Timeouts page (Properties > Network Timeouts): This is the amount of time for the application Web server to generate a response to OracleAS Web Cache. If the application Web server or proxy server is unable to generate a response within that time, OracleAS Web Cache sends a network apology page to the Web browser.

Usually, this value should be equal to the response time of the slowest document served by the application Web Server. If the value is too low, long-running requests will timeout before the response is complete. If the value is too high and the application Web server hangs for some reason, it will take longer for OracleAS Web Cache to failover to another application Web server.

For information on specifying these settings, see the *Oracle Application Server Web Cache Administrator's Guide*.

If these resources are set reasonably, check the following:

- Caching rules. Make sure that you are caching the appropriate objects including popular objects.
- Priority rankings of the caching rules. Give the non-cacheable documents a higher priority than the cacheable documents.
- The number of rules. If you have a large number of rules, parsing of rules will take additional time.

If the settings for the origin server and OracleAS Web Cache are set correctly, but the response times are still higher than expected, check system resources, especially:

- Network bandwidth. See [Section 5.1.4.3, "Allocate Sufficient Network Bandwidth"](#)
- CPU usage. See [Section 5.1.2.1, "Use Two CPUs and Consider Deploying on Dedicated Hardware"](#)

- The TCP time-wait setting. This setting controls the amount of time that the operating system holds a port, not allowing new connections to use the same port. See the *Oracle Application Server 10g Performance Guide* for more information.

5.1.7.2 Improve Response Times and Reduce Network Bandwidth With Compression

OracleAS Web Cache features automatic compression of dynamically generated content. On average, using the standard GZIP algorithm, OracleAS Web Cache is able to compress text files, such as HTML and XML by a factor of four. Because compressed objects are smaller in size, they require less bandwidth to transmit and can be delivered faster to Web browsers. With compression, everyone benefits: Internet Service Providers (ISPs), Hosting Service Provider (HSPs), corporate networks and content providers reduce their transmission costs, while end-users enjoy more rapid response times. Since 1997, all major Web browsers support the expansion of GZIP encoded documents.

Most application Web servers on the market are capable of serving compressed pages, but few enable caching of compressed output. With OracleAS Web Cache, compression is a simple Yes/No option that an administrator selects when specifying a caching rule. Because OracleAS Web Cache supports regular expression for caching rules, compression can be applied to responses using criteria other than just file extension. Regular expression makes it very easy to select which pages to compress and which pages not to compress, as well as whether or not a particular Web browser should receive compressed content. Unlike the typical application Web server, OracleAS Web Cache offers compression and caching for pages that have been dynamically generated and for ESI fragments specified in a Surrogate-Control response header. By caching compressed output, OracleAS Web Cache reduces the processing burden on the application Web server, which would otherwise have to re-generate and compress dynamic pages each time they are requested. Because compressed objects are smaller in size, they are delivered faster to Web browsers with fewer round-trips, reducing overall latency. In addition, compressed objects consume less cache memory.

Do not compress images, such as GIFs and JPEGs, as well as executables and files that are already zipped with utilities like WinZip and GZIP. Compressing these files incurs additional overhead without the benefits of compression. Also, do not compress JavaScript includes (.js) and Cascading Style Sheets (.css), as some Web browsers have difficulty expanding these file types.

For more information on compression, see the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.7.3 Use Only Warning or Notification Logging Levels to Conserve Resources

You can specify the level of detail for the event log. The Trace or Debug levels are useful for debugging purposes. However, using either of these levels of event logging consumes system resources. For example, the log file might fill up disk space, causing OracleAS Web Cache to crash. Unless you need to diagnose problems, you should the Warning or Notification levels. With these levels, OracleAS Web Cache writes only typical events to the event log. Also, consider disabling access logging unless you are monitoring end-user performance and access.

For information on specifying these settings, see Chapter 12, "Logging Events, Diagnostics, and Access Information" in the *Oracle Application Server Web Cache Administrator's Guide*.

Oracle HTTP Server

This chapter describes best practices for Oracle HTTP Server. It includes the following topics:

- [Section 6.1, "Configure Appropriately For Modem Connections"](#)
- [Section 6.2, "Tune TCP/IP Parameters"](#)
- [Section 6.3, "Tune KeepAlive Directives"](#)
- [Section 6.4, "Tune MaxClients Directive"](#)
- [Section 6.5, "Avoid any DNS Lookup"](#)
- [Section 6.6, "Turn Off Access Logging"](#)
- [Section 6.7, "Use FollowSymLinks and Not SymLinkIfOwnerMatch"](#)
- [Section 6.8, "Set AllowOverride To None"](#)
- [Section 6.9, "Use mod_rewrite to Hide URL Changes for End Users"](#)
- [Section 6.10, "Sticky Routing at Load Balancer is not Required"](#)

6.1 Configure Appropriately For Modem Connections

Some Web sites have visitors connecting from slow modems. Sometimes, it takes longer for data to transfer over these slow connections than for data to be computed by the application server. Thus, an Oracle HTTP Server process can be blocked doing the transfer, and CPU processing power is not available for another request to perform computation.

If this is perceived to be a problem in your environment, you should front-end Oracle HTTP Server with either:

- OracleAS Web Cache which uses threaded architecture
- Oracle HTTP Server in reverse proxy mode, which can spawn more light weight processes to handle the transfer

In both cases, the backend Oracle HTTP Server is reserved to do the computation work. This separation of data computation and data transfer responsibilities buffers a site from latency due to slow modem connections.

6.2 Tune TCP/IP Parameters

Setting TCP/IP parameters can improve Oracle HTTP Server performance. Refer to the *Oracle Application Server 10g Performance Guide*.

6.3 Tune KeepAlive Directives

The `KeepAlive`, `KeepAliveTimeout`, and `MaxKeepAliveRequests` directives are used to control persistent connections. Persistent connections are supported in HTTP1.1 to allow a client to send multiple sequential requests through the same connection.

Setting `KeepAlive` to **On** allows Apache to keep the connection open for that client when the client requests it. This can improve performance, because the connection has to be set up only once. The trade-off is that the `httpd` server process cannot be used to service other requests until either the client disconnects, the connection times out (controlled by the `KeepAliveTimeout` directive), or the `MaxKeepAliveRequests` value has been reached.

You can change these `KeepAlive` parameters to meet your specific application needs, but you should not set the `MaxKeepAliveRequests` to 0. A value of 0 in this directive means there is no limit. The connection will be closed only when the client disconnects or times out.

You may also consider setting `KeepAlive` to **Off** if your application has a large population of clients who make infrequent requests.

6.4 Tune MaxClients Directive

The `MaxClients` directive controls the maximum number of clients who can connect to the server simultaneously. This value is set to 1024 by default. If your requests have a short response time, you may be able to improve performance by setting `MaxClients` to a lower value. However, when this value is reached, no additional processes will be created, causing other requests to fail. In general, increasing the value of `MaxClients` does not improve performance when the system is saturated.

If you are using persistent connections, you may require more concurrent `httpd` server processes, and you may need to set the `MaxClients` directive to a higher value. Tune this directive according to the `KeepAlive` parameters.

6.5 Avoid any DNS Lookup

Any DNS lookup can affect Oracle HTTP Server performance. The `HostNameLookups` directive in Apache informs Apache whether it should log information based on the IP address (if the directive is set to **Off**), or look up the hostname associated with the IP address of each request in the DNS system on the Internet (if the directive is set to **On**).

Performance degrades by a minimum of approximately 3% in tests with `HostNameLookups` set to **On**. Depending on the server load and the network connectivity to your DNS server, the performance cost of the DNS lookup could be much higher. Unless you really need to have host names in your logs in real time, it is best to log IP addresses and resolve IP addresses to host names off-line.

6.6 Turn Off Access Logging

It is useful to have access logs for your Web server, both for load tracking and for the detection of security violations. However, if you find that you do not need these data, you should turn it off and reduce the overhead of writing the data to this log file.

6.7 Use FollowSymLinks and Not SymLinkIfOwnerMatch

The `FollowSymLinks` and `SymLinkIfOwnerMatch` options are used by Apache to determine if it should follow symbolic links. If the `SymLinkIfOwnerMatch` option is

used, Apache will check the symbolic link and make sure the ownership is the same as that of the server.

6.8 Set AllowOverride To None

If the AllowOverride directive is not set to None, Apache will check for directives in the htaccess files at each directory level until the requested resource is found for each URL request.

6.9 Use mod_rewrite to Hide URL Changes for End Users

Oracle HTTP Server includes a component, `mod_rewrite`, that can transparently map the URLs visible to the end users to a different URL. This is accomplished without a round-trip via the Web browser, or any code change.

This feature makes it very easy to re-organize directories on the server side or perform other changes; you can do this even after an application has been developed and deployed. There is a slight performance impact, however, as this configuration change for `mod_rewrite` is preferred.

6.10 Sticky Routing at Load Balancer is not Required

`mod_oc4j` is able to perform sticky routing independent of any external router or OracleAS Web Cache. Thus, for J2EE requests, the external router need not be set to perform sticky routing. However, if there are other requests (non-J2EE) in the mix that require sticky routing, then the sticky routing setup has to be performed on the external load balancer.

J2EE Applications

This describes the best practices for J2EE Applications. It includes the following topics:

- [Section 7.1, "Java Server Pages Best Practices"](#)
- [Section 7.2, "Sessions Best Practices"](#)
- [Section 7.3, "Enterprise Java Bean Best Practices"](#)
- [Section 7.4, "Data Access Best Practices"](#)
- [Section 7.5, "J2EE Class Loading Best Practices"](#)
- [Section 7.6, "Oracle Application Server TopLink Best Practices"](#)
- [Section 7.7, "Oracle Application Server XML Developer's Kit Best Practices"](#)
- [Section 7.8, "Java Message Service Best Practices"](#)

7.1 Java Server Pages Best Practices

This section describes Java Server Pages (JSP) best practices. It includes the following topics:

- [Section 7.1.1, "Pre-Translate JSPs Before Deployment"](#)
- [Section 7.1.2, "Separate Presentation Markup From Java"](#)
- [Section 7.1.3, "Use JSP Template Mechanism"](#)
- [Section 7.1.4, "Set Sessions=False If Not Using Sessions"](#)
- [Section 7.1.5, "Always Invalidate Sessions When No Longer Used"](#)
- [Section 7.1.6, "Set Main_Mode Attribute To "justrun""](#)
- [Section 7.1.7, "Use Available JSP Tags In Tag Library"](#)
- [Section 7.1.8, "Minimize Context Switching Between Servlets and EJBs"](#)
- [Section 7.1.9, "Package JSP Files In EAR File For Deployment Rather Than Standalone"](#)
- [Section 7.1.10, "Use Compile-Time Object Introspection"](#)
- [Section 7.1.11, "Choose Static Versus Dynamic Includes Appropriately"](#)
- [Section 7.1.12, "Disable JSP Page Buffer If Not Used"](#)
- [Section 7.1.13, "Use Forwards Instead of Redirects"](#)
- [Section 7.1.14, "Use JSP Tagged Cache"](#)
- [Section 7.1.15, "Use well_known_taglib_loc To Share Tag Libraries"](#)

- [Section 7.1.16, "Use JSP-Timeout for Efficient Memory Utilization"](#)
- [Section 7.1.17, "Workarounds for the 64K Size Limit for the Generated Java Method"](#)
- [Section 7.1.18, "Workarounds for the Size Limit"](#)
- [Section 7.1.19, "Hiding JSP Pages"](#)

7.1.1 Pre-Translate JSPs Before Deployment

You can use the Oracle `ojspc` tool to pre-translate the JSPs and avoid the translation overhead that has to be incurred when the JSPs are executed the first time. You can pre-translate the JSPs on the production system or before you deploy them. Also, pre-translating the JSPs allows you the option to deploy only the translated and compiled class files, if you choose not to expose and compromise the JSP source files.

7.1.2 Separate Presentation Markup From Java

Separating presentation markup such as HTML from Java code is a good practice to get better performance from your application. The following are a few tips:

- Use JavaBeans for the business logic and JSPs only for the view. Thus, JSPs should primarily contain logic for HTML (or other presentation markup) generation only.
- Use stylesheets when appropriate to provide even more separation of the aspects of HTML that a user can control better.
- JSPs containing a large amount of static content, including large amounts of HTML code that does not change at runtime, which may result in slow translation and execution. Use dynamic includes, or better, enable the external resource configuration parameter to put the static HTML into a Java resource file.

7.1.3 Use JSP Template Mechanism

Using the JSP code `out.print("<html>")` requires more resources than including static template text. For performance reasons, it is best to reserve the use of `out.print()` for dynamic text.

7.1.4 Set Sessions=False If Not Using Sessions

The default for JSPs is `session="true"`. If your JSPs do not use any sessions, you should set `session="false"` to eliminate the overhead of creating and releasing these internal sessions created by the JSP runtime. To disable sessions, set the directive as follows:

```
<%@page session="false" %>
```

7.1.5 Always Invalidate Sessions When No Longer Used

Sessions add performance overhead to your Web applications. Each session is an instance of the `javax.servlet.http.HttpSession` class. The amount of memory used per session depends on the size of the session objects created.

If you use sessions, ensure that you explicitly cancel each session using the `invalidate()` method to release the memory occupied by each session when you no longer need it.

The default session timeout for OC4J is 30 minutes. You can change this for a specific application by setting the `<session-timeout>` parameter in the `<session-config>` element of the `web.xml` file.

7.1.6 Set Main_Mode Attribute To "justrun"

This attribute, found in the `global-web-application.xml` file, determines whether classes are automatically reloaded or JSPs are automatically recompiled. In a deployment environment set `main_mode` to `justrun`. The runtime dispatcher does not perform any timestamp checking, so there is no re-compilation of JSPs or reloading of Java classes. This mode is the most efficient mode for a deployment environment where code is not expected to change.

If comparing timestamps is unnecessary, as is the case in a production deployment environment where source code does not change, you can avoid all timestamp comparisons and any possible re-translations and reloads by setting the `main_mode` parameter to the value `justrun`. Using this value can improve the performance of JSP applications.

Note that before you set `main_mode` to `justrun`, make sure that the JSP is compiled at least once. You can compile the JSP by invoking it through a Web browser or by running your application (using the `recompile` value for `main_mode`). This assures that the JSP is compiled before you set the `justrun` flag.

7.1.7 Use Available JSP Tags In Tag Library

JSP tags make the JSP code cleaner, and more importantly, provide easy reuse. In some cases, there is also a performance benefit. Oracle Application Server ships with a very comprehensive JSP tag library that will meet most needs. In cases where custom logic is required or if the provided library is insufficient, you can build a custom tag library, if appropriate.

7.1.8 Minimize Context Switching Between Servlets and EJBs

Minimize context switching between different Enterprise JavaBeans (EJB) and servlet components especially when the EJB and Web container processes are different. If context switching is required, co-locate EJBs whenever possible.

7.1.9 Package JSP Files In EAR File For Deployment Rather Than Standalone

Oracle Application Server supports deploying of JSP files by copying them to the appropriate location. This is very useful when developing and testing the pages. However, this is not recommended for releasing your JSP-based application for production. You should always package JSP files into an Enterprise Archive (EAR) file so that they can be deployed in a standard manner - even across multiple application servers.

7.1.10 Use Compile-Time Object Introspection

Developers should try to rely on compile-time object introspection on the beans and objects generated by the tag library instead of request-time introspection.

7.1.11 Choose Static Versus Dynamic Includes Appropriately

JSP pages have two different include mechanisms:

- Static includes which have a page directive such as:

```
<%@ include file="filename.jsp" %>
```

- Dynamic includes which have a page directive such as:

```
<jsp:include page="filename.jsp" flush="true" />
```

Static includes create a copy of the include file in the JSP. Therefore, it increases the page size of the JSP, but it avoids additional trips to the request dispatcher. Dynamic includes are analogous to function calls. Therefore, they do not increase the page size of the calling JSP, but they do increase the processing overhead because each call must go through the request dispatcher.

Dynamic includes are useful if you cannot determine which page to include until after the main page has been requested. Note that a page that can be dynamically included must be an independent entity, which can be translated and executed on its own.

7.1.12 Disable JSP Page Buffer If Not Used

In order to allow part of the response body to be produced before the response headers are set, JSPs can store the body in a buffer.

When the buffer is full or at the end of the page, the JSP runtime will send all headers that have been set, followed by any buffered body content. This buffer is also required if the page uses dynamic content type settings, forwards, or error pages. The default size of a JSP page buffer is 8 KB. If you need to increase the buffer size, for example to 20KB, you can use the following JSP attribute and directive:

```
<%@page buffer="20kb" %>
```

If you are not using any JSP features that require buffering, you can disable it to improve performance; memory will not be used in creating the buffer, and output can go directly to the Web browser. You can use the following directive to disable buffering:

```
<%@ page buffer="none" %>
```

7.1.13 Use Forwards Instead of Redirects

For JSPs, you can pass control from one page to another by using forward or redirect, but forward is always faster. When you use forward, the forwarded target page is invoked internally by the JSP runtime, which continues to process the request. The Web browser is totally unaware that such an action has taken place.

When you use redirect, the Web browser actually has to make a new request to the redirected page. The URL shown in the Web browser is changed to the URL of the redirected page, but it stays the same in a forward operation.

Therefore, redirect is always slower than the forward operation. In addition, all request scope objects are unavailable to the redirected page because redirect involves a new request. Use redirect only if you want the URL to reflect the actual page that is being executed in case the user wants to reload the page.

7.1.14 Use JSP Tagged Cache

Using the Java Object Cache in JSP pages, as opposed to servlets, is particularly convenient because JSP code generation can save much of the development effort. OracleJSP provides the following tags for using the Java Object Cache:

- `ojsp:cache`

- `ojsp:cacheXMLObj`
- `ojsp:useCacheObj`
- `ojsp:invalidateCache`

Use the `ojsp:cacheXMLObj` or `ojsp:cache` tag to enable caching and specify cache settings. Use `ojsp:useCacheObj` to cache any Java serializable object. Use the `ojsp:invalidateCache` tag to invalidate a cache block. Alternatively, you can arrange invalidation through the `invalidateCache` attribute of the `ojsp:cacheXMLObj` or `ojsp:cache` tag.

7.1.15 Use `well_known_taglib_loc` To Share Tag Libraries

As an extension of standard JSP "well-known URI" functionality described in the JSP 1.2 specification, the OC4J JSP container supports the use of a shared tag library directory where you can place tag library JAR files to be shared across multiple Web applications. The benefits are:

- avoidance of duplication of tag libraries between applications
- allow easy maintenance as the TLDs can be in a single JAR file
- application size is minimized

OC4J JSP `well_known_taglib_loc` configuration parameter specifies the location of the shared tag library directory. The default location is `j2ee/home/jsp/lib/taglib/` under the `ORACLE_HOME` directory. If `ORACLE_HOME` is not defined, it is the current directory (from which the OC4J process was started).

The shared directory must be added to the server-wide CLASSPATH by specifying it as a library path element. The default location is set in the `application.xml` file in the OC4J configuration files directory (`j2ee/home/config` by default) and can be altered.

7.1.16 Use JSP-Timeout for Efficient Memory Utilization

Resource utilization is a key factor for any efficient application. Oracle Application Server introduces the `<orion-web-app>` attribute `jsp-timeout` that can be specified in the OC4J `global-web-application.xml` file or `orion-web.xml` file. The `jsp-timeout` attribute specifies an integer value, in seconds, after which any JSP page will be removed from memory if it has not been requested. This frees up resources in situations where some pages are called infrequently. The default value is 0, for no timeout.

Like other attributes use the `<orion-web-app>` element of the OC4J `global-web-application.xml` file to apply to all applications in an OC4J instance. To set configuration values to a specific application, use the `<orion-web-app>` element of the deployment-specific `orion-web.xml` file.

7.1.17 Workarounds for the 64K Size Limit for the Generated Java Method

The JVM limits the amount of code to 65536 bytes per Java method. Sometimes, as the JSPs grow larger, there is a possibility of hitting this limit. The following are some suggestions to workaround this limitation:

As a general rule, design smaller JSPs for your web application.

If your JSP uses tag libraries heavily, and if you are hitting the 64k limit, use the `reduce_tag_code` config parameter to reduce the size of generated code for custom tag usage. Note that this may impact performance.

7.1.18 Workarounds for the Size Limit

The JVM limits the amount of code to 65536 bytes per Java method. Sometimes, as JSPs grow larger, there is a possibility of reaching this limit. The following are some suggestions to workaround this limitation:

- Design smaller JSPs for your Web application.
- If your JSP primarily uses tag libraries, and if you are reaching the 64k limit, use the `reduce_tag_code` configuration parameter to reduce the size of generated code for custom tag usage. However, note that this may impact performance.
- The version of the JDK you are using can impact the size of the code generated. Generally, JDK 1.4.2_04-b05 generates far less code compared to JDK 1.4.1_03-b02.

7.1.19 Hiding JSP Pages

There are situations when you do not want to allow access to specific JSPs from a Web browser. For example, when a JSP is not presented in the Web browser but is part of application logic which gets accessed by other JSPs or servlets. Put the JSPs you want to hide into a `/WEB-INF` directory.

Access within your application code with the following:

```
<jsp: forward page = "/WEB-INF/forwarded.jsp"/>
```

7.2 Sessions Best Practices

This section describes Sessions best practices. It includes the following topics:

- [Section 7.2.1, "Persist Session State if Appropriate"](#)
- [Section 7.2.2, "Replicate Sessions if Persisting is Not an Option"](#)
- [Section 7.2.3, "Do Not Store Shared Resources in Sessions"](#)
- [Section 7.2.4, "Set Session Timeout Appropriately"](#)
- [Section 7.2.5, "Monitor Session Memory Usage"](#)
- [Section 7.2.6, "Always Use Islands, But Keep Island Size Small"](#)
- [Section 7.2.7, "Use a Mix of Cookie and Sessions"](#)
- [Section 7.2.8, "Use Coarse Objects Inside HTTP Sessions"](#)
- [Section 7.2.9, "Use Transient Data in Sessions Whenever Appropriate"](#)
- [Section 7.2.10, "Invalidate Sessions"](#)
- [Section 7.2.11, "Miscellaneous Guidelines"](#)

7.2.1 Persist Session State if Appropriate

HTTP Sessions are used to preserve the conversation state with a Web browser. As such, they hold information, which if lost, could result in a client having to start the conversation over.

Hence, it is always safe to save the session state in database. However, this imposes a performance penalty. If this overhead is acceptable, then persisting sessions is indeed the best approach.

There are trade-offs when implementing state safety that affect performance, scalability, and availability. If you do not implement state-safe applications, then:

- A single JVM process failure will result in many user session failures. For example, work done shopping online, filling in a multiple page form, or editing a shared document will be lost, and the user will have to start over.
- Not having to load and store session data from a database will reduce CPU overhead, thus increasing performance.
- Having session data clogging the JVM heap when the user is inactive reduces the number of concurrent sessions a JVM can support, and thus decreases scalability.

In contrast, a state safe application can be written so that session state exists in the JVM heap for active requests only, which is typically 100 times fewer than the number of active sessions.

To improve performance of state safe applications:

- Minimize session state. For example, a security role might map to detailed permissions on thousands of objects. Rather than store all security permissions as session state, just store the role id. Maintain a cache, shared across many sessions, mapping role id to individual permissions.
- Identify key session variables that change often, and store these attributes in a cookie to avoid database updates on most requests.
- Identify key session variables that are read often, and use `HttpSession` as a cache for that session data in order to avoid having to read it from the database on every request. You must manually synchronize the cache, which requires care to handle planned and unplanned transaction rollback.

7.2.2 Replicate Sessions if Persisting is Not an Option

For the category of applications where the HTTP session state information cannot be persisted and retrieved on each HTTP request (due to the performance overhead), OC4J provides an intermediate option, replication.

It can replicate the session state information across an island of servers (which are in the same cluster). This provides a performance improvement because the sessions remain in memory, and fault tolerance. This is because Oracle HTTP Server automatically routes the HTTP requests to a different server in the island, if the original OC4J (and the session it contains) is down.

Hence, the best practice here is to at least setup two servers in an island, so that they can back session state for each other.

7.2.3 Do Not Store Shared Resources in Sessions

Objects that are stored in the session objects will not be released until the session times out (or is invalidated). If you hold any shared resources that have to be explicitly released to the pool before they can be reused (such as a JDBC connection), then these resources may never be returned to the pool properly and can never be reused.

7.2.4 Set Session Timeout Appropriately

Set session timeout appropriately (`setMaxInactiveInterval()`) so that neither sessions timeout frequently nor does it live for ever this consuming memory.

7.2.5 Monitor Session Memory Usage

Monitor the memory usage for the data you want to store in session objects. Make sure there is sufficient memory for the number of sessions created before the sessions time out.

7.2.6 Always Use Islands, But Keep Island Size Small

Setting up an island of OC4J JVMs causes the sessions to be replicated across all JVMs. This provides better fault tolerance, since a server crash does not necessarily result in a lost session. Oracle Application Server automatically re-routes request to another server in the island; thus an end-user never finds out about a failure. However, this replication overhead increases as more servers are added to the island. For example, if your session object requires 100KB per user, and there are 100 users per server. This results in a 10MB memory requirement for session replication per server. If you have 5 servers in an island, the memory requirement increases five-fold. Since islands provide session replication, it is, in general, not prudent to exceed an island size beyond 3.

Hence, setting up multiple islands, with few servers in an island is a better choice compared to having a fewer number of larger sized islands.

7.2.7 Use a Mix of Cookie and Sessions

Typically, a cookie is set on the Web browser (automatically by the container), to track a user session. In some cases, this cookie may last a much longer duration than a single user session. (For example, one time settings, such as to determine the end-user geographic location).

Thus, a cookie that persists on the client disk could be used to save information valid for the long-term, while a server side session will typically include information valid for the short-term.

In this situation, the long-term cookie should be parsed on only the first request to the server, when a new session established. The session object created on the server should contain all the relevant information, so as not to require re-parsing the cookie on each request.

A new client side cookie should then be set that contains only an ID to identify the server side session object. This is automatically done for any JSP page that uses sessions.

This gives performance benefit since the session object contents do not have to be re-created from the long-term cookie. The other option is to save the user settings in a database on the server, and have the user login. The unique userid can then be used to retrieve the contents from the database and store the information in a session.

7.2.8 Use Coarse Objects Inside HTTP Sessions

Oracle Application Server automatically replicates sessions when session object is updated. If a session object contains granular objects, (for example, a person's name), it results in too many update events to all the servers in the island. Hence, it is recommended to use coarse objects, (for example the person object, as opposed to the name attribute), inside the session.

7.2.9 Use Transient Data in Sessions Whenever Appropriate

Oracle Application Server does not replicate transient data in a session across servers in the island. This reduces the replication overhead (and also the memory requirements). Hence, use the transient type liberally.

7.2.10 Invalidate Sessions

The number of active users is generally quite small compared to the number of users on the system. For example, of the 100 users on a Web site, only 10 may actually be doing something.

A session is typically established for each user on the system, which uses memory.

Simple things, like a logout button, provide an opportunity for quick session invalidation and removal. This avoids memory usage growth since the sessions on the system will be closer to the number of active users, as opposed to all those that have not timed out yet.

7.2.11 Miscellaneous Guidelines

- Use sessions as light-weight mechanism by verifying session creation state.
- Use cookies for long-standing sessions.
- Put recoverable data into sessions so that they can be recovered if the session is lost. Store non-recoverable data persistently (in file system or in database using JDBC). However, storing every data persistently is an expensive thing. Instead, one can save data in sessions and use `HttpSessionBindingListener` or other events to flush data into persistent storage during session close.
- Sticky versus Distributable Sessions
 - Distributable session data must be serialized and useful for failover. However it is expensive, as the data has to be serialized and replicated among peer processes.
 - Sticky sessions affect load-balancing across multiple JVMs, but are less expensive as there is no state replication.

7.3 Enterprise Java Bean Best Practices

This section describes Enterprise Java Beans (EJB) best practices. It includes the following topics:

- [Section 7.3.1, "Local, Remote, and Message Driven EJBs"](#)
- [Section 7.3.2, "Use EJB Judiciously"](#)
- [Section 7.3.3, "Use Service Locator Pattern"](#)
- [Section 7.3.4, "Cluster Your EJBs"](#)
- [Section 7.3.5, "Index Secondary Finder Methods"](#)
- [Section 7.3.6, "Understand EJB Lifecycle"](#)
- [Section 7.3.7, "Use Deferred Database Constraints"](#)
- [Section 7.3.8, "Create a Cache with Read Only EJBs"](#)
- [Section 7.3.9, "Pick an Appropriate Locking Strategy"](#)
- [Section 7.3.10, "Understand and Leverage Patterns"](#)

- [Section 7.3.11, "When Using Entity Beans, Use Container Managed Aged Persistence Whenever Possible"](#)
- [Section 7.3.12, "Entity Beans using Local interfaces Only"](#)
- [Section 7.3.12, "Entity Beans using Local interfaces Only"](#)
- [Section 7.3.13, "Use a Session Bean Facade for Entity Beans"](#)
- [Section 7.3.14, "Enforce Primary Key Constraints at the Database Level"](#)
- [Section 7.3.15, "Use Foreign Key for 1-1 and 1-M Relationships"](#)
- [Section 7.3.16, "Avoid findAll Method on Entities Based on Large Tables"](#)
- [Section 7.3.17, "Set prefetch-size to Reduce Round Trips to Database"](#)
- [Section 7.3.18, "Use lazy-loading with Caution"](#)
- [Section 7.3.19, "Avoid Performing O-R Mapping Manually"](#)

7.3.1 Local, Remote, and Message Driven EJBs

EJBs can be local or remote. If you envision calls to an EJB to originate from the same container as the one running the EJB, local EJBs are better since they do not entail the marshalling, unmarshalling, and network communication overhead. The local beans also allow you to pass an object-by-reference, thus, improving performance further.

Remote EJBs allow clients to be on different machines and/or different application server instances to talk to them. In this case, it is important to use the value object pattern to improve performance by reducing network traffic.

If you choose to write an EJB, write a local EJB over a remote EJB object. Since the only difference is in the exception on the EJB object, almost all of the implementation bean code remains unchanged.

Additionally, if you do not have a need for making synchronous calls, message driven beans are more appropriate.

7.3.2 Use EJB Judiciously

An EJB is a reusable component backed by component architecture with several useful services: persistence, transactions security, naming, etc. However, these additions make it "heavy."

If you just require abstraction of some functionality and are not leveraging the EJB container services, you should consider using a simple JavaBean, or implement the required functionality using JSPs or servlets.

7.3.3 Use Service Locator Pattern

Most J2EE services and/or resources require "acquiring" a handle to them via an initial Java Naming and Directory Interface (JNDI) call. These resources could be an EJB Home object, or, a JMS topic. This results in expensive calls to the server machine to resolve the JNDI reference, even though the same client may have gone to the JNDI service for a different thread of execution to fetch the same data!

Hence, it is recommended to have a **Service Locator**, which in some sense is a local proxy for the JNDI service, so that the client programs talk to the local service locator, which in turn talks to the real JNDI service, and that only if required.

The Java Object Cache bundled with the product may be used to implement this pattern.

This practice improves availability since the service locator can hide failures of the backend server or JNDI tree by having cached the lookup. Although this is only temporary since the results still have to be fetched.

Performance is also improved since trips to the back-end application server are reduced.

7.3.4 Cluster Your EJBs

Cluster your EJBs only when you require:

- **Load Balancing:** The EJB client(s) are load balanced across the servers in the EJB cluster.
- **Fault Tolerance:** The state (in case of stateful session beans) is replicated across the OC4J processes in the EJB cluster. If the proxy classes on the client cannot connect to an EJB server, they will attempt to connect to the next server in the cluster. The client does not see the failure.
- **Scalability:** Since multiple EJB servers behaving as one can service many more requests than a single EJB server, a clustered EJB system is more scalable. The alternative is to have stand-alone EJB systems, with manual partitioning of clients across servers. This is difficult to configure and does not have fault tolerance advantages.

In order to fully leverage EJB clustering you will need to use remote EJBs. Remote EJBs have some performance implications over local EJBs ([Section 7.3.1, "Local, Remote, and Message Driven EJBs"](#)). If you use local EJBs and save a reference to them in a servlet (or JSP) session, when the session is replicated the saved reference becomes invalid. Therefore, use EJB clustering only when you need the listed features.

7.3.5 Index Secondary Finder Methods

When finder methods, other than `findByPrimaryKey` and `findAll`, are created they may be extremely inefficient if appropriate indexes are not created that help to optimize execution of the SQL generated by the container.

7.3.6 Understand EJB Lifecycle

As a developer, it is imperative that you understand the EJB lifecycle. Many problems can be avoided by following the lifecycle and the expected actions during call backs more closely.

This is especially true with entity beans and stateful session beans. For example, in a small test environment, a bean may never get passivated, and thus a mis-implementation (or non-implementation) of `ejbPassivate()` and `ejbActivate()` may not show up until later. Moreover, since these are not used for stateless beans, they may confuse new developers.

7.3.7 Use Deferred Database Constraints

For those constraints that may be invalid for a short time during a transaction but will be valid at transaction boundaries, use deferred database constraints. For example, if a column is not populated during an `ejbCreate()`, but will be set prior to the completion of the transaction, then you may want to set the not null constraint for that column to be deferred. This also applies to foreign key constraints that are mirrored by EJB relationships with EJB 2.0.

7.3.8 Create a Cache with Read Only EJBs

For those cases where data changes very slowly or not at all, and the changes are not made by your EJB application, read-only beans may make a very good cache. A good example of this is a country EJB. It is unlikely that it will change very often and it is likely that some degree of stale data is acceptable.

To do this:

1. Create read-only entity beans.
2. Set `exclusive-write-access="true"`.
3. Set the validity timeout to the maximum acceptable staleness of the data.

7.3.9 Pick an Appropriate Locking Strategy

It is critical that an appropriate locking strategy be combined with an appropriate database isolation mode for properly performing and highly reliable EJB applications.

Use optimistic locking where the likelihood of conflict in updates is low. If a lost update is acceptable or cannot occur because of application design, use an isolation mode of read-committed. If the lost updates are problematic, use an isolation mode of serializable.

Use pessimistic locking where there is a higher probability of update conflicts. Use an isolation mode of read-committed for maximum performance in this case. Use read-only locking when the data will not be modified by the EJB application.

7.3.10 Understand and Leverage Patterns

With the wider industry adoption, there are several common (and generally) acceptable ways of solving problems with EJBs. These have been widely published in either books or discussion forums, etc. In some sense, these patterns are best practices for a particular problem. These should be researched and followed.

Here are some examples:

- Session Façade: Combines multiple entity bean calls into a single call on a session bean, thus reducing the network traffic.
- Message Façade: Use MDBs if you do not need a return status from your method invocation.
- Value Object Pattern: A value object pattern reduces the network traffic by combining multiple data values that are usually required to be together, into a single value object.

A full discussion on the large number of patterns available is outside the scope of this document, but the references section contains some useful books and/or Web sites on this subject.

7.3.11 When Using Entity Beans, Use Container Managed Aged Persistence Whenever Possible

Although there are some limitations to container-managed persistence (CMP), CMP has a number of benefits. One benefit is portability. With CMP, decisions like persistence mapping and locking model selection become a deployment activity rather than a coding activity. This allows deployment of the same application in multiple containers with no change in code. This is commonly not true for Bean Managed

Persistence (BMP) since SQL statements and concurrency control must be written into the entity bean and are therefore specific to the container and/or the data store.

Another benefit is that, in general, J2EE container vendors provide quality of service (QoS) features such as locking model variations, lazy loading, and performance and scalability enhancements, which may be controlled via deployment configuration rather than by writing code. Oracle Application Server includes features such as read-only entity beans, minimal writing of changes, and lazy loading of relations, which would have to be built into code for BMP.

A third benefit of CMP is container-managed relationships. Through declarations, not unlike CMP field mapping, a CMP entity bean can have relationships between two entity beans managed by the container with no implementation code required from application developers.

Last but not least, tools are available to aid in the creation of CMP entity beans so that minimal work is required from developers for persistence. This allows developers to focus on business logic, which allows them to be more efficient. JDeveloper9i is a perfect example where, through modeling tools and wizards, very little work is required to create CMP entity beans including creation of both the generic EJB descriptors and the Oracle Application Server specific descriptors.

Overall, there are cases where CMP does not meet the requirements of an application, but the development effort saved, and the optimizations that J2EE containers like OC4J provide make CMP much more attractive than BMP.

7.3.12 Entity Beans using Local interfaces Only

It is a good practice to expose your entity beans using only local interfaces because container managed relationship can only be used with local interfaces. Also local interfaces avoid expensive serialization and remote network calls.

7.3.13 Use a Session Bean Facade for Entity Beans

Avoid using entity beans directly from Web modules and client applications and use a session bean façade layer instead. Use of entity beans from client applications hardcodes the domain model in the client. It also introduces difficulty when managing both remote and local interfaces for entity beans.

Create a session bean façade layer by grouping together all natural use cases. This exposes operations to one or more entity beans. It provides finer grained access to the entity beans and reduces database interactions by acting as a transaction boundary. This also enables the entity beans to be accessed by Web services by exposing the stateless session bean as a Web service endpoint.

7.3.14 Enforce Primary Key Constraints at the Database Level

Enforce primary key constraint for the underlying table for your CMP entity beans instead of having the container execute an extra SQL statement to check for a duplicate primary key. You can switch this check by setting the `do-select-before-insert="false"` for your entity bean in the `orion-ejb-jar.xml` file.

7.3.15 Use Foreign Key for 1-1 and 1-M Relationships

Use a foreign key when completing the O-R mapping for 1-1 and 1-Many relationships between entity beans instead of using an association table. This allows you to avoid

maintaining an extra table and an extra SQL statement generated by container to maintain the relationships.

7.3.16 Avoid findAll Method on Entities Based on Large Tables

When you use the `findAll` method, the container tries to retrieve all rows of the table. You should try to avoid this type of operation on entities based on tables that have a large number of records. It will slowdown the operations of your database.

7.3.17 Set prefetch-size to Reduce Round Trips to Database

Oracle JDBC drivers have extensions that allows setting the number of rows to prefetch into the client while a result set is being populated during a query. This reduces the number of round trips to the server. This can drastically improve performance of finder methods that return a large number of rows. You can specify the `prefetch-size` attribute for your finder method in the `orion-ejb-jar.xml` file.

7.3.18 Use lazy-loading with Caution

If you turn on lazy loading, which is off by default, then only the primary keys of the objects retrieved within the finder are returned. Thus, when you access the object within your implementation, the OC4J container uploads the actual object based on the primary key.

You may want to turn on the lazy loading feature if the number of objects that you are retrieving is so large that loading them all into your local cache would decrease performance. If you retrieve multiple objects, but you only use a few of them, then you should turn on lazy loading. In addition, if you only use objects through the `getPrimaryKey` method, then you should also turn on lazy loading.

7.3.19 Avoid Performing O-R Mapping Manually

O-R mapping for CMP entity beans in the `orion-ejb-jar.xml` file is very complex and error prone. Any error in the mapping can cause deployment errors and generation of wrong SQL for EJB-QL statements. The following two approaches are recommended:

- Use JDeveloper 9.0.5.1 to perform the O-R mapping and to generate the mapping information in the `orion-ejb-jar.xml` file.
- Deploy the application in OC4J to generate the mappings and then modify the `orion-ejb-jar.xml` file to include the correct table-name and persistence-names.

7.4 Data Access Best Practices

This section describes data access best practices. It includes the following topics:

- [Section 7.4.1, "Datasources Connections Caching and Handling"](#)
- [Section 7.4.2, "Datasource Initialization"](#)
- [Section 7.4.3, "Disable Escape Processing for Better Performance"](#)
- [Section 7.4.4, "Defining Column Types"](#)
- [Section 7.4.5, "Prefetching Rows Improves Performance"](#)
- [Section 7.4.6, "Update Batching Improves Performance"](#)
- [Section 7.4.7, "Use Emulated and Non-Emulated Data Sources Appropriately"](#)
- [Section 7.4.8, "Use the EJB-Aware Location Specified in Emulated Data Sources"](#)
- [Section 7.4.9, "Set the Maximum Open Connections in Data Sources"](#)
- [Section 7.4.10, "Set the Minimum Open Connections in Data Sources"](#)
- [Section 7.4.11, "Setting the Cache Connection Inactivity Timeout in Data Sources"](#)
- [Section 7.4.12, "Set the Wait for Free Connection Timeout in Data Sources"](#)
- [Section 7.4.13, "Set the Connection Retry Interval in Data Sources"](#)
- [Section 7.4.14, "Set the Maximum Number of Connection Attempts in Data Sources"](#)
- [Section 7.4.15, "Use JDBC Connection Pooling and Connection Caching"](#)
- [Section 7.4.16, "Use JDBC Statement Caching"](#)
- [Section 7.4.17, "Avoid Using More Than One Database Connection Simultaneously in the Same Request"](#)
- [Section 7.4.18, "Tune the Database and SQL Statements"](#)

7.4.1 Datasources Connections Caching and Handling

Connections must not be closed within `finalize()` methods. This can cause the connection cache to run out of connections to use, since the connection is not closed until the object that obtained it is garbage collected.

The current connection cache does not provide any mechanism to detect "abandoned" connections, reclaim them, and return them to the cache. All connections can be explicitly closed by the application.

If a connection is declared as static, then it is possible that the same connection object is used on different threads at the same time. Do not declare connections as static objects.

Use the `FIXED_WAIT_SCHEME` when using the connection cache, especially when writing Web applications. This guarantees enforcement of the `MaxLimit` on the connection cache as well as retrieval of a connection from the cache when a connection is returned to the cache.

Always use Connection Cache Timeouts such as `CacheInactivityTimeout` to close unused physical connections in the cache and cause "shrinking" of the cache, thus releasing valuable resources.

7.4.1.1 DataSource Connection Caching Strategies

In order to minimize the lock up of resources for long periods of time but allow for recycling of connections from the connection cache, you should use the most appropriate strategy for obtaining and releasing connections as follows:

- Many clients, few connections - Open and close a connection in the same method that needs to use the connection. In order to ensure that connections are returned to the pool, all calls to this method should happen within try-catch, try-finally, or try-catch-finally blocks. This strategy is useful when you have a large number of clients sharing a few connections at the cost of the overhead associated with getting and closing each connection.
- Private client pool - Take advantage of the BMP life cycle. Get a connection within `setEntityContext()` and release the connection in `unsetEntityContext()`. Make connections available to all methods by declaring it a member instance.
- Combined strategy - You may take further advantage of BMP life cycle and implement a strategy which combines the two above.

7.4.2 Datasource Initialization

It is a good practice to put the JNDI lookup of a `DataSource` as part of the application initialization code, since `DataSource`s are simply connection factories.

For example, when using servlets, it is a good idea to put the `DataSource` lookup code into the `init()` method of the servlet.

7.4.3 Disable Escape Processing for Better Performance

Escape processing for SQL92 syntax is enabled by default, which results in the JDBC driver performing escape substitution before sending the SQL code to the database. If you want the driver to use regular Oracle SQL syntax, which is more efficient than SQL92 syntax and escape processing, then disable escape processing using the following statement:

```
stmt.setEscapeProcessing(false);
```

7.4.4 Defining Column Types

The Oracle-specific defining column types feature provides the following benefits:

- Saves a roundtrip to the database server.
- Defines the datatype for every column of the expected result set.
- For `VARCHAR`, `VARCHAR2`, `CHAR`, and `CHAR2`, specifies their maximum length.

The following example illustrates the use of this feature. It assumes you have imported the `oracle.jdbc.*` and `java.sql.*` interfaces and classes.

```
//ds is a DataSource object
Connection conn = ds.getConnection();
PreparedStatement pstmt = conn.prepareStatement("select empno, ename, hiredate
from emp");

//Avoid a roundtrip to the database and describe the columns
((OraclePreparedStatement)pstmt).defineColumnType(1, Types.INTEGER);

//Column #2 is a VARCHAR, we need to specify its max length
((OraclePreparedStatement)pstmt).defineColumnType(2, Types.VARCHAR, 12);
```

```

((OraclePreparedStatement)pstmt).defineColumnType(3,Types.DATE);
ResultSet rset = pstmt.executeQuery();
while (rset.next())
System.out.println(rset.getInt(1)+", "+rset.getString(2)+", "+rset.getDate(3));
pstmt.close();
...

```

7.4.5 Prefetching Rows Improves Performance

Row prefetching improves performance by reducing the number of round trips to a database server. For most database-centric applications, Oracle recommends the use of row prefetching as much as possible. The default prefetch size is 10.

The following example illustrates the use of row prefetching. It assumes you have imported the `oracle.jdbc.*` and `java.sql.*` interfaces and classes

```

//ds is a DataSource object Connection conn = ds.getConnection();

//Set the default row-prefetch setting for this connection
((OracleConnection)conn).setDefaultRowPrefetch(7);

//The following statement gets the default row-prefetch value for
//the connection, that is, 7 Statement stmt = conn.createStatement();
//Subsequent statements look the same, regardless of the row

//prefetch value. Only execution time changes.
ResultSet rset = stmt.executeQuery("SELECT ename FROM emp");
System.out.println( rset.next () );
while( rset.next () )
System.out.println( rset.getString (1) );

//Override the default row-prefetch setting for this

//statement
( (OracleStatement)stmt ).setRowPrefetch (2);
ResultSet rset = stmt.executeQuery("SELECT ename FROM emp");
System.out.println( rset.next () );
while( rset.next() )
System.out.println( rset.getString (1) );
stmt.close();
...
.
.

```

7.4.6 Update Batching Improves Performance

Update Batching sends a batch of operations to the database in one trip. When using it, always disable auto-commit mode with Update Batching. Use a batch size of around 10. Do not mix the standard and Oracle models of Update Batching.

7.4.6.1 Oracle Update Batching

The following example illustrates how you use the Oracle Update Batching feature. It assumes you have imported the `oracle.jdbc.driver.*` interfaces.

```

//ds is a DataSource object
Connection conn = ds.getConnection();

```

```
//Always disable auto-commit when using update batching
conn.setAutoCommit(false);
PreparedStatement ps =
conn.prepareStatement("insert into dept values (?, ?, ?)");
//Change batch size for this statement to 3
((OraclePreparedStatement)ps).setExecuteBatch (3);
//-----#1-----
ps.setInt(1, 23);
ps.setString(2, "Sales");
ps.setString(3, "USA");
ps.executeUpdate(); //JDBC queues this for later execution
//-----#2-----
ps.setInt(1, 24);
ps.setString(2, "Blue Sky");
ps.setString(3, "Montana");
ps.executeUpdate(); //JDBC queues this for later execution
//-----#3-----
ps.setInt(1, 25);
ps.setString(2, "Applications");
ps.setString(3, "India");
ps.executeUpdate(); //The queue size equals the batch value of
3

//JDBC sends the requests to the database
//-----#1-----
ps.setInt(1, 26);
ps.setString(2, "HR");
ps.setString(3, "Mongolia");
ps.executeUpdate(); //JDBC queues this for later execution
((OraclePreparedStatement)ps).sendBatch(); // JDBC sends the
//queued request
conn.commit();
ps.close();
...
```

7.4.6.2 Standard Update Batching

This example uses the standard Update Batching feature. It assumes you have imported the `oracle.jdbc.driver.*` interfaces.

```
//ds is a DataSource object
Connection conn = ds.getConnection();
//Always disable auto-commit when using update batching
conn.setAutoCommit(false);
Statement s = conn.createStatement();
s.addBatch("insert into dept values ('23','Sales','USA')");
s.addBatch("insert into dept values ('24','Blue
Sky','Montana')");
s.addBatch("insert into dept values
('25','Applications','India')");
//Manually execute the batch
s.executeBatch();
s.addBatch("insert into dept values ('26','HR','Mongolia')");
s.executeBatch();
conn.commit();
ps.close();
...
```

7.4.7 Use Emulated and Non-Emulated Data Sources Appropriately

For speed and performance reasons emulated data sources are preferred over non-emulated ones.

A non-emulated datasource provides JDBC v2.0 compliance and additional capabilities such as XA which may not be required for all applications.

Some of the performance related configuration options have different affects, depending on the type of the data source. OC4J supports two types of data sources, emulated and non-emulated.

The pre-installed default data source is an emulated data source. Emulated data sources are wrappers around Oracle data sources. If you use these data sources, your connections are extremely fast, because they do not provide full XA or JTA global transactional support. Oracle recommends that you use these data sources for local transactions or when your application requires access or update to a single database.

You can use emulated data sources for Oracle or non-Oracle databases. You can use the emulated data source to obtain connections to different databases by changing the values of the URL and connection-driver parameters.

The following is a definition of an emulated data source:

```
<data-source
class="com.evermind.sql.DriverManagerDataSource"
name="OracleDS"
location="jdbc/OracleCoreDS"
xa-location="jdbc/xa/OracleXADS"
ejb-location="jdbc/OracleDS"
connection-driver="oracle.jdbc.driver.OracleDriver"
username="scott"
password="tiger"
url="jdbc:oracle:thin:@localhost:5521:oracle"
inactivity-timeout="30"
/>
```

Non-emulated data sources are pure Oracle data sources. These are used by applications that want to coordinate access to multiple sessions within the same database or to multiple databases within a global transaction.

7.4.8 Use the EJB-Aware Location Specified in Emulated Data Sources

Each data source is configured with one or more logical names that allow you to identify the data source within J2EE applications. The EJB-location is the logical name of an EJB data source. In addition, use the EJB-location name to identify data sources for most J2EE applications, where possible, even when not using EJBs. The EJB-location only applies to emulated data sources. You can use this option for single phase commit transactions or emulated data sources.

Using the EJB-location, the data source manages opening a pool of connections, and manages the pool. Opening a connection to a database is a time-consuming process that can sometimes take longer than the operation of getting the data itself. Connection pooling allows client requests to have faster response times, because the applications do not need to wait for database connections to be created. Instead, the applications can reuse connections that are available in the connection pool.

Oracle recommends that you only use the EJB-location JNDI name in emulated data source definitions for retrieving the data source. For non-emulated data sources, you must use the location JNDI name.

7.4.9 Set the Maximum Open Connections in Data Sources

The `max-connections` option specifies the maximum number of open connections for a pooled data source. To improve system performance, the value you specify for the number `max-connections` depends on a combination of factors including the size and configuration of your database server, and the type of SQL operations that your application performs. The default value for `max-connections` and the handling of the maximum depends on the data source type, emulated or non-emulated.

For emulated data sources, there is no default value for `max-connections`, but the database configuration limits that affect the number of connections apply. When the maximum number of connections, as specified with `max-connections`, are all active, new requests must wait for a connection to be become available. The maximum time to wait is specified with `wait-timeout`.

For non-emulated data sources, there is a property, `cacheScheme`, that determines how `max-connections` is interpreted. The following lists the values for the `cacheScheme` property (`DYNAMIC_SCHEME` is the default value for `cacheScheme`).

FIXED_WAIT_SCHEME: In this scheme, when the maximum limit is reached, a request for a new connection waits until another client releases a connection.

FIXED_RETURN_NULL_SCHEME: In this scheme, the maximum limit cannot be exceeded. Requests for connections when the maximum has already been reached return null.

DYNAMIC_SCHEME: In this scheme, there is no maximum limit.

For some applications you can improve performance by limiting the number of connections to the database (this causes the system to queue requests in the middle-tier).

For example, for one application that performed a combination of updates and complex parallel queries into the same database table, performance was improved by over 35% by reducing the maximum number of open connections to the database by limiting the value of `max-connections`.

7.4.10 Set the Minimum Open Connections in Data Sources

The `min-connections` option specifies the minimum number of open connections for a pooled data source.

For applications that use a database, performance can improve when the data source manages opening a pool of connections, and manages the pool. This can improve performance because incoming requests don't need to wait for a database connection to be established; they can be given a connection from one of the available connections, and this avoids the cost of closing and then reopening connections.

By default, the value of `min-connections` is set to 0. When using connection pooling to maintain connections in the pool, specify a value for `min-connections` other than 0.

For emulated and non-emulated data sources, the `min-connections` option is treated differently.

For emulated data sources, when starting up the initial `min-connections` connections, connections are opened as they are needed and once the `min-connections` number of connections is established, this number is maintained.

For non-emulated data sources, after the first access to the data source, OC4J then starts the `min-connections` number of connections and maintains this number of connections.

Limiting the total number of open database connections to a number your database can handle is an important tuning consideration. You should check to make sure that your database is configured to allow at least as large a number of open connections as the total of the values specified for all the data sources `min-connections` options, as specified in all the applications that access the database.

7.4.11 Setting the Cache Connection Inactivity Timeout in Data Sources

The `inactivity-timeout` specifies the time, in seconds, to cache unused connections before closing them.

To improve performance, you can set the `inactivity-timeout` to a value that allows the data source to avoid dropping and then re-acquiring connections while your J2EE application is running.

The default value for the `inactivity-timeout` is 60 seconds, which is typically too low for applications that are frequently accessed, where there may be some inactivity between requests. For most applications, to improve performance, Oracle recommends that you increase the `inactivity-timeout` to 120 seconds.

To determine if the default `inactivity-timeout` is too low, monitor your system. If you see that the number of database connections grows and then shrinks during an idle period, and grows again soon after that, you have two options:

- You can increase the `inactivity-timeout`.
- You can increase the `min-connections`.

7.4.12 Set the Wait for Free Connection Timeout in Data Sources

The `wait-timeout` specifies the number of seconds to wait for a free connection if the connection pool does not contain any available connections (that is, the number of connections has reached the limit specified with `max-connections` and they are all currently in use).

If you see connection timeout errors in your application, increasing the `wait-timeout` can prevent the errors. The default `wait-timeout` is 60 seconds.

If database resources, including memory and CPU are available and the number of open database connections is approaching `max-connections`, you may have limited `max-connections` too stringently. Try increasing `max-connections` and monitor the impact on performance. If there are not additional machine resources available, increasing `max-connections` is not likely to improve performance.

You have several options in the case of a saturated system:

- Increase the allowable `wait-timeout`.
- Evaluate the application design for potential performance improvements.
- Increase the system resources available and then adjust these parameters.

7.4.13 Set the Connection Retry Interval in Data Sources

The `connection-retry-interval` specifies the number of seconds to wait before retrying a connection when a connection attempt fails.

If the `connection-retry-interval` is set to a small value, or a large number of connection attempts is specified with `max-connect-attempts` this may degrade performance if there are many retries performed without obtaining a connection.

The default value for the `connection-retry-interval` is 1 second.

7.4.14 Set the Maximum Number of Connection Attempts in Data Sources

The `max-connect-attempts` option specifies the maximum number of times to retry making a connection. This option is useful to control when the network is not stable, or the environment is unstable for any reason that sometimes makes connection attempts fail.

If the `connection-retry-interval` option is set to a small value, or a large number of connection attempts is specified with `max-connect-attempts` this may degrade performance if there are many retries performed without obtaining a connection.

The default value for `max-connect-attempts` is 3.

7.4.15 Use JDBC Connection Pooling and Connection Caching

Constant creation and destruction of resource objects can be very expensive in Java. Oracle suggests using a resources pool to share resources that are expensive to create. The JDBC connections are one of the most common resources used in any Web application that requires database access. They are also very expensive to create; overhead from hundreds of milliseconds to seconds (depending on the load) in establishing a JDBC connection on a mid-size system with 4 CPUs and 2 GB memory.

In JDBC 2.0, a connection-pooling API allows physical connections to be reused. A pooled connection represents a physical connection, which can be reused by multiple logical connections. When a JDBC client obtains a connection through a pooled connection, it receives a logical connection. When the client closes the logical connection, the pooled connection does not close the physical connection. It simply frees up resources, clears the state, and closes any statement objects associated with the instance before the instance is given to the next client. The physical connection is released only when the pooled connection object is closed directly.

The term pooling is extremely confusing and misleading in this context. It does not mean there is a pool of connections. There is just one physical connection, which can be serially reused. It is still up to the application designer to manage this pooled connection to make sure it is used by only one client at a time.

To address this management challenge, Oracle's extension to JDBC 2.0 also includes connection caching, which helps manage a set of pooled connections. It allows each connection cache instance to be associated with a number of pooled connections, all of which represent physical connection to the same database and schema. You can use one of Oracle's JDBC connection caching schemes (`DYNAMIC`, `FIXED_WITH_NO_WAIT`, or `FIXED_WAIT`) to determine how you want to manage the pooled connections, or you can use the connection caching APIs to implement your own caching mechanisms.

7.4.16 Use JDBC Statement Caching

Use JDBC statement caching to cache a JDBC `PreparedStatement` or `OracleCallableStatement` that is used repeatedly in the application to:

- prevent repeated statement parsing and recreation
- reduce the overhead of repeated cursor creation

The performance gain will depend on the complexity of the statement and how often the statement has to be executed. Since each physical connection has its own statement cache, the advantage of using statement caching with a pool of physical connections may vary. That is, if you execute a statement in a first connection from a pool of physical connections, it will be cached with that connection. If you later get a different physical connection and want to execute the same statement, then the cache is not useful.

See Also: *Oracle JDBC Developer's Guide and Reference*

7.4.17 Avoid Using More Than One Database Connection Simultaneously in the Same Request

Using more than one database connection simultaneously in a request can cause a deadlock in the database. This is most common in JSPs. First, a JSP will get a database connection to do some data accessing. But then, before the JSP commits the transaction and releases the connection, it invokes a bean which gets its own connection for its database operations. If these operations are in conflict, they can result in a deadlock.

Furthermore, you cannot easily roll back any related operations if they are done by two separate database connections in case of failure.

Unless your transaction spans multiple requests or requires some complex distributed transaction support, you should try to use just one connection at a time to process the request.

7.4.18 Tune the Database and SQL Statements

Current Web applications are still very database-centric. From 60% to 90% of the execution time on a Web application can be spent in accessing the database. No amount of tuning on the mid-tier can give significant performance improvement if the database machine is saturated or the SQL statements are inefficient.

Monitor frequently executed SQL statements. Consider alternative SQL syntax, use PL/SQL or bind variables, pre-fetch rows, and cache rowsets from the database to improve your SQL statements and database operations.

Web applications often access a database at the backend. One must carefully optimize handling of database resources, since a large number of concurrent users and high volumes of data may be involved. Database performance tuning can be divided into two categories:

- Tuning of SQL tables and statements.
- Tuning of JDBC calls to access the SQL database.
- Also refer to the following JDBC tuning topics:
 - [Section 7.4.18.1, "JDBC Tuning"](#)
 - [Section 7.4.18.2, "JDBC Connection Caching"](#)
 - [Section 7.4.18.3, "JDBC Statement Caching"](#)
 - [Section 7.4.18.4, "JDBC Cached Rowsets"](#)

7.4.18.1 JDBC Tuning

JDBC objects such `Connections`, `Statements`, and `Result Sets` are quite often used for database access in Web applications. Frequent creation & destruction of these objects can be quite detrimental to the performance and scalability of the application

as these objects are quite heavy-weight. So it is always desirable to cache these JDBC resources

7.4.18.2 JDBC Connection Caching

- Reuse database connections thus avoiding frequent session creations and tear-downs.
- EJBs, servlets, JSPs can use or share the connection cache within a JVM.
- Create at startup as a single object so that they can be shared across multiple requests.

7.4.18.3 JDBC Statement Caching

- Avoids cursor creation and teardown
- Avoid cursor parsing
- Two types of statement caching:
 - Implicit: Saves Metadata of cursor but clears the State and Data content of the cursor across calls
 - Explicit: Saves Metadata, Data, and State of the cursor across calls
- Can be used with Pooled Connection and Connection Cache
For example: `conn.setStmtCacheSize(<cache-size>)`

7.4.18.4 JDBC Cached Rowsets

- Result set implementation that is disconnected, serializable, and scrollable
- Free up connections and cursors faster
- Local scrolling on cached data
- Specially useful for:
 - small read-only data set
 - scrolling for long time

7.5 J2EE Class Loading Best Practices

This section describes best practices for J2EE class loading. It includes the following topics:

- [Section 7.5.1, "Avoid Duplicating Libraries"](#)
- [Section 7.5.2, "Load Resources Appropriately"](#)
- [Section 7.5.3, "Setting Class Loading Search Order within Web Modules"](#)
- [Section 7.5.4, "Declare and Group Dependencies"](#)
- [Section 7.5.5, "Minimize Visibility"](#)
- [Section 7.5.6, "Keep Configurations Portable"](#)
- [Section 7.5.7, "Do not Use the lib Directory for Container Wide Shared Libraries"](#)

7.5.1 Avoid Duplicating Libraries

Avoid duplicating copies of the same library at different location in your Oracle Application Server installation. Duplication of class libraries can lead to several class loading problems and may consume additional memory and disk space. If your class library is used by multiple applications, then you can put it at the application server level by using the `<library>` tag in the `application.xml` file. Or, use the `<parent>` attribute in the `server.xml` file to share libraries in two applications.

For a class library that is to be shared by all applications running on your OC4J instance, you can use the default global shared library directory. To do this, place the class library in the `$ORACLE_HOME/j2ee/home/applib` directory. This directory is configured as a default in the `$ORACLE_HOME/j2ee/home/config/application.xml` configuration file.

If you have a library that is shared between multiple modules in the same application, i.e. two Web modules in the same EAR file, then use the WAR file manifest `Class-Path` element to share the class libraries between the modules instead of duplicating the libraries in the `WEB-INF/lib` for every module. In order to enable the use of a `Class-Path` entry in a WAR file manifest, the following has to be defined in the `orion-web.xml` for your Web application:

```
<web-app-class-loader include-war-manifest-class-path="true" />
```

If you have a library that is shared between multiple modules in the same EAR file, (for example, two EJB modules in the same EAR file), then you can place the shared class library and an `orion-application.xml` file within your EAR file and define the relative path to the shared library using a `<library>` tag within the `orion-application.xml` file.

7.5.2 Load Resources Appropriately

If you are using dynamic class loading or are loading a resource (for example, the properties file in your application), use the correct loader.

If you call `Class.forName()`, always explicitly pass the loader returned by:

```
Thread.currentThread().getContextClassLoader();
```

If you are loading a properties file, use:

```
Thread.currentThread().getContextClassLoader().getResourceAsStream();
```

7.5.3 Setting Class Loading Search Order within Web Modules

The servlet 2.3 specification requires that the search order within a Web module is `WEB-INF/classes` first, then `WEB-INF/lib`. The specification recommends another class loading related implementation detail:

"the application classloader [could] be implemented so that classes and resources packaged within the WAR are loaded in preference to classes and resources residing in container wide library JARs".

To enable this local classes class loading behavior for a Web application, you configure it on a per Web-application basis by having the following tag in the `orion-web.xml` file:

```
<web-app-class-loader search-local-classes-first="true"/>
```

This tag is commented out by default and is not the default behavior.

7.5.4 Declare and Group Dependencies

Make dependencies between your class libraries explicit. Hidden or unknown dependencies will be left behind when you move your application to another environment. Use available mechanisms such as `Class-Path` entries in manifest files to declare dependencies amongst class libraries and applications.

Group dependencies between your class libraries and ensure that all dependencies are visible at the same level or above. If you must move a library, make sure all the dependencies are still visible.

7.5.5 Minimize Visibility

Dependency libraries should be placed at the lowest visibility level that satisfies all dependencies. For example, if a library is only used by a single Web application, it should only be included in the `WEB-INF/lib` directory of the WAR file.

7.5.6 Keep Configurations Portable

Create configurations that are as portable as possible. Specify configuration options in the following order:

- Standard J2EE options
- Options that can be expressed within your EAR file
- Server-level options
- J2SE extension options

7.5.7 Do not Use the lib Directory for Container Wide Shared Libraries

Do not place container-wide shared application libraries in the `$ORACLE_HOME/j2ee/home/lib` directory. With Oracle Application Server 10g, this directory is no longer used for loading custom container-wide shared libraries. The container will not load additional libraries placed in this directory.

If you wish to use a container-wide shared library, place your class library in the `$ORACLE_HOME/j2ee/home/applib` directory or define it using a `<library>` tag.

7.6 Oracle Application Server TopLink Best Practices

This section describes best practices for Oracle Application Server TopLink (OracleAS TopLink). It includes the following topics:

- [Section 7.6.1, "OracleAS TopLink Mapping"](#)
- [Section 7.6.2, "Team Development"](#)
- [Section 7.6.3, "Caching"](#)
- [Section 7.6.4, "Sequencing"](#)
- [Section 7.6.5, "Performance Options"](#)

OracleAS TopLink is a stand alone component within Oracle Application Server; recommendations in this section are specific to the use of OracleAS TopLink and may not apply to the rest of Oracle Application Server. OracleAS TopLink is compatible with Oracle Application Server or other application servers. OracleAS TopLink fully supports persistence for any Java application with any JDBC2.0 compliant database.

OracleAS TopLink provides flexibility to interact with any application design, not only at initial construction but also as the application evolves. It also interacts with the complexities of the underlying relational database. This flexibility enables the two domains to interact to form a high performance system, and also evolve separately, while minimizing complexity in the application or database domains. Ultimately, supplying a general best practise is difficult as each situation will be different. Therefore, the reader must understand that these guidelines will not apply in all situations.

See Also: *Oracle Application Server TopLink Application Developer's Guide*

7.6.1 OracleAS TopLink Mapping

These are some general guidelines for use in mapping object models and/or designing relational models to persist application data.

- If in doubt, always use indirection for any relationship. Indirection does more than just provide deferred reading. It also provides deferred cloning when an object is made transactional.
- For Inheritance, it is usually advantageous to flatten out the inheritance hierarchy in the relational model. Just because there are several levels of inheritance in the object model, this does not mean that you necessarily need that many levels of joined tables in the relational model.
- Use version numbers, instead of timestamps, for Optimistic Locking.
- For very simple aggregation (for example, involving only one database field), consider `TypeConversionMappings` with an extended `ConversionManager`.

7.6.2 Team Development

This section describes Team Development best practices. It includes the following topics:

- [Section 7.6.2.1, "Team Working with Metadata"](#)
- [Section 7.6.2.2, "Large and/or Geographically Diverse Project Development"](#)

7.6.2.1 Team Working with Metadata

Part of the OracleAS TopLink development effort involves the creation of metadata: the Project, Descriptor and Mappings.

Most of this metadata can be defined using the Mapping Workbench (MW) tool.

It is good practice to create an MW administrator role to be responsible for any necessary updates to metadata, whether occasioned by changes to the object model, schema, or business logic.

An administrator will need to deal with change requests from both the Java development and DBA team, and be able to call on expert resource in both.

However, modelling the metadata requires significant knowledge of the business logic. Answering questions such as which attributes should be indirected or not by default, which descriptors are read-only, are reference data, and others should be reflected in the skill set of the role.

The MW project is represented as a series of files under a directory structure. The number and names of these files changes frequently, it is not recommended to enact source control if all the files are contained individually in source management.

The most effective technique for most teams is to zip up the MW directory and place the zip in source control. This mandates that only one person can be updating the workbench at any one time. In practice, after the first few days of a project this is rarely a restriction, and much preferable than attempting to merge changes made simultaneously by more than one person on the highly interdependent mappings.

Not all the metadata can be defined in the MW: other metadata has to be defined in initialization code, usually through descriptor amendment (after load) methods and in a Session `preLogin` event. (Several examples of metadata configured this way, such as mappings that use `ProxyIndirection`, are included in the [Example 7-1](#)).

Thus it is necessary to consider, for versioning and baselining purposes, the MW project, the `sessions.xml` file, and any code that further refines the metadata during session initialization as a single unit that together fully defines the static and runtime O-R mapping properties. It is recommended to factor out code that alters the metadata into separate clearly defined classes and version/baseline. Named queries, which effectively represent metadata describing how most efficiently to retrieve data required for some business process, should also be considered part of this unit.

O-R mapping metadata has dependencies both to the Object Model and business logic, and on the database schema. Versioning/baselining should reflect a connection between a specific code version and a specific schema version.

7.6.2.2 Large and/or Geographically Diverse Project Development

It is common practice in large or geographically diverse projects to divide the metadata into several projects. For example, one project for all common objects, and other projects for particular subsystems.

Projects will have interdependencies. For instance, many subsystems will have mappings that reference objects from the common project. To be able to map in the subsystem it is necessary only to create a **stub** descriptor with the correct name and package (importing the relevant class into the workbench is the easiest way to achieve this).

It is not necessary to map any part of the stub descriptor, it is just there to be mapped to by the descriptors which are really part of the project. Stub descriptors should be "deactivated" which means that they will not be deployed with the active classes in the deployment XML Project source file. This option is on the right-button of the context menu. Use the following steps:

1. Instantiate all the separate projects.
2. Choose a base project (the project which represents the common domain objects is almost always the appropriate one).
3. Iterate through the descriptors of all the other projects, and add each descriptor to the base project.
4. Only configure and login a session on this base project.

A Session pre-login event can be used to combine the Descriptors from the two separately developed projects together into one super-Project. It is this Project which is actually used to build the runtime OracleAS TopLink Session. However, in this scenario, the different Projects must share exactly the same DatabaseLogin characteristics (for example, target Database, use of ConnectionPools, use of external TransactionController). As long as this is valid, then OracleAS TopLink Projects can be developed separately and later merged into a single run-time Session.

The code sample shown in [Example 7-1](#) represents a `preLogin` event which combines two Projects (from two separate MW Projects) into one single run-time Session.

Example 7-1 preLogin Event Script

```

public void preLogin(SessionEvent event)
{
    // get the Project from the current Session
    // (we are going to add metadata to this project)
    Project projectA = event.getSession().getProject();

    //this session should use this conversion manager which will
    // use the current ClassLoader to load classes

    ConversionManager conversionManager = new ConversionManager();
    ConversionManager.setDefaultLoader(this.getClass().getClassLoader());
    projectA.getLogin().getPlatform().setConversionManager(conversionManager);

    // somehow, we need to look up "other" projects in order to
    // merge their metadata into the current project. There
    // are many potential ways of managing this (SessionManager,
    // .properties filed, etc.). For now, we will hard code
    // a reference to the "other" Project .class

    Project projectB = new ProjectB();

    // iterate over all of the Descriptors in ProjectB
    // and add them to ProjectA
    {
        for (Enumeration enum = projectB.getDescriptors().elements());
        enum.hasMoreElements();
        )
        {
            Descriptor descriptor = (Descriptor)enum.nextElement();
            projectA.addDescriptor(descriptor);
        }
    }
}

```

If you are using a `sessions.xml` deployment descriptor to define runtime sessions the technique above is still used. Create a uniquely named session for each project in the `sessions.xml` file. The only runtime information that is relevant is that in the session describing the base project

There are overloaded versions of the `SessionManager` methods that read in a session from `sessions.xml` without logging it in. Use these to instantiate each Session object, obtain their Project object and iterate through the descriptors as above. Then login to the base Project session.

7.6.3 Caching

OracleAS TopLink has its own specialized object caching mechanism, which is separate from other caching solutions in Oracle Application Server. It is tightly integrated with the rest of the OracleAS TopLink runtime and, provides additional performance benefits.

This section describes caching best practices. It includes the following topics:

- [Section 7.6.3.1, "OracleAS TopLink Cache Refreshing Policies"](#)
- [Section 7.6.3.2, "Avoiding Stale Cache Content"](#)
- [Section 7.6.3.3, "Cache Synchronization"](#)

7.6.3.1 OracleAS TopLink Cache Refreshing Policies

This section describe OracleAS TopLink Refreshing Policies. It includes the following topics:

- [Section 7.6.3.1.1, "When the Cache is Refreshed"](#)
- [Section 7.6.3.1.2, "Choosing A Cache Refresh Policy"](#)

7.6.3.1.1 When the Cache is Refreshed OracleAS TopLink does not perform cache invalidation. In OracleAS TopLink, objects are refreshed in one of the following ways:

- If the cache holds a weak reference (for example, using WeakIdentityMaps) then objects are simply garbage collected on a regular basis
- If a query is set to `refreshIdentityMapResult()` then all objects returned from the query are refreshed with the most recent data from the database
- Objects can be explicitly refreshed using the **refreshObject** API on the OracleAS TopLink Session.
- Objects are sometimes implicitly refreshed as a result of OracleAS TopLink merging in a remote `ChangeSet`. This technique is used by the OracleAS TopLink `CacheSynchronizationManager` whenever a OracleAS TopLink Session is configured to use cache sync.

7.6.3.1.2 Choosing A Cache Refresh Policy OracleAS TopLink cache usage, for a single `ServerSession`, is straight forward in two different situations

- the cache can always be trusted (for example, no non middle-tier processes are changing the database). The default policy of **checkCache** is preferred here.
- the cache can never be trusted. A policy of "alwaysRefreshCache/disableCacheHits" is appropriate in this situation. OracleAS TopLink queries can be configured to handle either of these two extremes.

However, if you the situation lies somewhere in between, there is some criteria that can be used to determine whether or not the a query should be refreshed. The cache usage settings can be adjusted dynamically to create the proper behavior.

In the code example, `QueryRedirector` is shown which can adjust whether or not a query refreshes itself based on some external criteria. Note that this is still implemented as a named query. It differs only in that the decision to refresh or to check the cache is not made until run time.

Example 7–2 QueryRedirector Example

```
public void amend(Descriptor descriptor)
{
    ReadObjectQuery query = new ReadObjectQuery();
    // configure query normally and then add the Redirector
    query.setRedirector
    (
        new QueryRedirector()
    )
    public Object invokeQuery(DatabaseQuery query, DatabaseRow row, Session
session)
    {
        if ( checkCustomRefreshContext() )
        }
        {
            System.out.println("need to refresh");
            ((ObjectLevelReadQuery)query).setShouldRefreshIdentityMapResult(true);
        }
    }
}
```

```

    }
    else
    {
        System.out.println("execute normally");
    }
    return
    ((oracle.toplink.publicinterface.Session)session).executeQuery(query, row);
}
);
descriptor.getQueryManager().addQuery("test", query);
}

```

A pattern such as this can be used to implement queries which refresh themselves only if some external criteria is met. An example of this is a query which changes its cache usage settings on a periodic basis. If the query has not been executed in a set amount of time, then the result would be automatically refreshed.

Note that this discussion addresses problems that occur as a result of non-OracleAS TopLink processes altering the database. For the case of multiple OracleAS TopLink sessions running in a clustered environment, the technique of handling OptimisticLock Exceptions, and possibly even using cache synchronization are still the recommended approaches.

7.6.3.2 Avoiding Stale Cache Content

J2EE Applications often share data with legacy applications and/or are running in a clustered environment. When using caching technology in environments such as this, your applications need a well thought out refresh strategy.

This section summarizes some options a developer has when wanting to explicitly refresh or clear out possible stale caches and how to do this as efficiently as possible. There is also a short discussion on cache synchronization.

If your application requires that a particular object needs to be refreshed, you can make use of the **refreshObject** API. For example:

```
clientSession.refreshObject(myObj);
```

If you are going to execute a query and you know the data is going to become stale quickly, you can tell OracleAS TopLink not to cache the results in the first place:

Here is an example:

```

ExpressionBuilder stockTick = new ExpressionBuilder();
Expression exp = stockTick.get("symbol").equal("ORCL");
ReadAllQuery raq = new ReadAllQuery(StockTick.class, exp);
//This is going to be stale immediately, so don't cache it
raq.dontMaintainCache();
results = session.executeQuery(raq);

```

Recommended Approach: On a per Query basis, tell the query to refresh the results of the cache to ensure you have the most up to date data. This may seem inefficient but it is faster to refresh an object than it would be to create a brand new one. Any type of query can be told to refresh with results.

```

ReadAllQuery raq = new ReadAllQuery(Employee.class);
raq.setSelectionCriteria( ... your expression ...);
raq.refreshIdentityMapResult();

```

You can make the refresh more efficient if you are using optimistic locking. As OracleAS TopLink is refreshing each result from a query, you can query it to check the

optimistic lock version first to see if the refresh is actually necessary. This option is set at the descriptor level. Here is an example:

```
public static void amendCustDescriptor(Descriptor d)
{
    d.onlyRefreshCacheIfNewerVersion();
}
```

See Also: *Oracle Application Server TopLink Mapping Workbench User's Guide*

7.6.3.3 Cache Synchronization

OracleAS TopLink allows developers to use cache synchronization when running in a clustered environment. Cache synchronization works particularly well in read intensive applications. This feature requires experimentation to see if it is appropriate for your applications use cases and can be affected by a number of issues such as the volume and frequency of updates, network, JVM, communication protocol, operating system, number of nodes in cluster, and many other factors. Where cache synchronization is not feasible, employ the previously mentioned refresh strategies as they work well in a clustered environment.

7.6.4 Sequencing

At the Project level, the **Sequencing** tab applies two project wide properties that are applied to all descriptors that use sequencing:

- whether OracleAS TopLink uses database native sequencing objects or a table to manage sequences. The preference is to use Oracle Native Sequencing.
- the Sequence Pre-Allocation size determines how many sequences OracleAS TopLink grabs and caches in one call. If you use Oracle Native Sequencing then the increment property of the sequence object must match the OracleAS TopLink Sequence Pre-Allocation size.

At the Descriptor level, the Use Sequencing section allows you to specify:

- Name: For native sequencing this will be the name of the database sequence object to be used.
- Table: The table which contains the field that the sequence will be applied to (Chosen from drop-down of all the tables a descriptor is mapped to).
- Field: The table field that the sequence will be applied to (Chosen from a drop down of all the fields from the chosen table that are mapped by the descriptor).

7.6.5 Performance Options

This section describes performance options best practices. It includes the following topics:

- [Section 7.6.5.1, "Performance Diagnostics"](#)
- [Section 7.6.5.2, "Tuning"](#)

7.6.5.1 Performance Diagnostics

When the OracleAS TopLink UnitOfWork commits, every object registering into this UnitOfWork is automatically inspected for changes. This is obviously a time-consuming process and one of the easiest ways to improve performance is to

minimize the number of objects that require inspection. One technique to analyze this is to check the size of the UnitOfWork transactional cache just before beginning the commit cycle. The appropriate location for this code depends on whether or not the OracleAS TopLink Session is using an External Transaction Controller.

```
// "session" refers to a TopLink DatabaseSession
// "uow" refers to the current UnitOfWork
for (Enumeration enum = session.getDescriptors().keys(); enum.hasMoreElements();)
{
    Class javaClass = (Class)enum.nextElement();
    IdentityMap map = uow.getIdentityMapManager().getIdentityMap(javaClass);
    System.out.println(javaClass.getName()+" : "+map.getSize());
}
```

This code shows the user how many of each type of object are registered into the current UnitOfWork. Since every object in this transactional cache must be checked for changes, a large cache implies a longer commit cycle.

Developers should have an idea of what the size of this cache should be. If this transaction involves the editing of 5 or 10 objects then try to insure that there are only 5 or 10 objects registered. A large cache size for a comparatively small transaction means that the UnitOfWork will be performing a lot of needless work during its commit cycle.

7.6.5.2 Tuning

This section describes tuning best practices. It includes the following topics:

- [Section 7.6.5.2.1, "Analyzing the UnitOfWork Commit Cycle"](#)
- [Section 7.6.5.2.2, "Reducing The Size of the Transactional Cache"](#)
- [Section 7.6.5.2.3, "Analyzing the Object-Building phase"](#)
- [Section 7.6.5.2.4, "Use of Named Queries"](#)

Tuning affects three important aspects of OracleAS TopLink performance:

- minimizing the number of objects in the UnitOfWork transactional cache
- minimizing the number of objects read in from the Database
- taking advantage of named queries

7.6.5.2.1 Analyzing the UnitOfWork Commit Cycle When the OracleAS TopLink UnitOfWork commits, every object registered into this UnitOfWork is automatically inspected for changes. This is obviously a time-consuming process and one of the easiest ways to improve performance is to minimize the number of objects that require inspection. One technique to analyze this is to check the size of the UnitOfWork "transactional" cache just before beginning the commit cycle. The appropriate location for this code depends on whether or not the OracleAS TopLink Session is using an External Transaction Controller.

```
// "session" refers to a TopLink DatabaseSession
// "uow" refers to the current UnitOfWork
for (Enumeration enum = session.getDescriptors().keys(); enum.hasMoreElements();)
{
    Class javaClass = (Class)enum.nextElement();
    IdentityMap map = uow.getIdentityMapManager().getIdentityMap(javaClass);
    System.out.println(javaClass.getName()+" : "+map.getSize());
}
```

This code shows the user how many of each type of object are registered into the current `UnitOfWork`. Since every object in this transactional cache must be checked for changes, a large cache implies a longer commit cycle.

Developers should have an idea of what the size of this cache should be. If this transaction involves the editing of 5 or 10 objects then try to insure that there are only 5 or 10 objects registered. A large cache size for a comparatively small transaction means that the `UnitOfWork` will be performing a lot of needless work during its commit cycle.

7.6.5.2.2 Reducing The Size of the Transactional Cache After determining that the `UnitOfWork` is checking too many objects, one must look for ways of reducing the size of this transactional cache. There are several techniques to use:

- Try not to register objects that are not going to be changed
- Use Indirection. The importance of this can not be overstated. Indirection allows OracleAS TopLink to only register related objects into a transaction when they are accessed in the context of that transaction. Without indirection, OracleAS TopLink will have to check all related objects for updates.
- Avoid querying against a `UnitOfWork`. If only a few objects from a query will actually be changed then registered the whole result set into the `UnitOfWork` will be a lot of overhead (both at query time and at commit time). Querying against a `UnitOfWork` is very convenient but it can be very dangerous if not used properly.
- Make use of the `UnitOfWork.unregisterObject` API. It is worthwhile un-registering objects from a `UnitOfWork` if you know that there have been no changes to the objects.

7.6.5.2.3 Analyzing the Object-Building phase After executing a OracleAS TopLink `ObjectLevelQuery`, OracleAS TopLink has to build all objects that have not been cached in a previous query. This can be an expensive operation if it is allowed to go unchecked. After building an object, OracleAS TopLink will throw a `postBuild Descriptor Event`. This can be useful to diagnose situations where slow performance is caused by building too many objects. [Example 7-3](#) is an example of a `postBuild Descriptor Event`:

Example 7-3 *postBuild Descriptor Event*

```
public class CaminusSessionListener extends SessionEventAdapter
{
    /**
     * catch the postBuild event for EVERY class in the current system
     */
    public void preLogin(SessionEvent arg0)
    {
        Session session = arg0.getSession(); for (Iterator it =
        session.getDescriptors().values().iterator(); it.hasNext();)
        {
            Descriptor desc = (Descriptor)it.next(); desc.getEventManager().addListener
            {
                new DescriptorEventAdapter()
            }
            {
                public void postBuild(DescriptorEvent event)
            }
        }
        // do we want to keep a running tally on the number of objects that are built
```

```
// during a single transaction in any case, we have access to the object which
// has just been built. Here it is:
{
    object object = event.getObject();
}
}
```

In this example, the postBuild event for every Class in the current OracleAS TopLink Project is noted. Developers should be looking for situations where they are building more objects than they feel are actually required to handle the current request. After determining that too many objects are being built, developers should consider some of the following:

- Further qualify the selection criteria of the DatabaseQuery if possible.
- Use Indirection. Indirection allows OracleAS TopLink to avoid building related objects that are not accessed during request processing.
- Increase the sub-cache sizes if using "soft" identity maps (for example, SoftCacheWeakIdentityMap, HardCacheWeakIdentityMap).
- Are there some "ToMany" mapping which have become overly large? For example, are you mapping collections of objects involving thousands of objects when typical Use Cases involve accessing only a small subset of these objects? In these situations, it is advisable to unmap the "ToMany" relationship and have the parent query for the subset of children directly.
- Use ReportQueries to build summary reports rather than actual objects.

7.6.5.2.4 Use of Named Queries The importance of named queries is typically underestimated. Defining queries in one place and then referencing them by name allows OracleAS TopLink to optimize several key steps in the execution of a query. In addition, centralizing query definitions can save weeks of development effort during the performance tuning phase of application development.

Without named queries, users end up with the following types of usage patterns scattered throughout the system:

```
ReadObjectQuery query = new ReadObjectQuery();
query.setReferenceClass(Person.class);
Expression exp = query.getExpressionBuilder().get("id").equal(argument));
query.setSelectionCriteria(exp);
getTopLinkSession().executeQuery(query);
```

There are several problems with this approach. The expression used above must be built and parsed every time this query is executed. In addition, the argument is statically bound into the expression, and into the generated select statement. This means that this statement will have to be prepared over and over again.

In an alternative implementation, the query is defined outside of the system. The following is an example of the Descriptor Amendment Method).

```
public static void amend(Descriptor desc)
{
    ReadObjectQuery query = new ReadObjectQuery();
    ExpressionBuilder builder = query.getExpressionBuilder();
    query.addArgument("ARG");

    Expression exp = builder.get("id").equal(builder.getParameter("ARG"));
    query.setSelectionCriteria(exp);
    desc.getQueryManager().addQuery("findByArg", query);
}
```

Using this query definition, application code can execute this query over and over again without having to re-build, re-parse, and re-prepare the any of the underlying implementation details.

```
getTopLinkSession().executeQuery("findByArg".Person.class,argument);
```

Using named queries, along with enabling `bindAllParameters` in the OracleAS TopLink, the DatabaseLogin can significantly improve the performance of all DatabaseQueries.

The use of named queries is conceptually very simple; however, it can be logistically very difficult if the system has already been developed with DatabaseQuery definitions spread haphazardly through the application code. This is an aspect of application design that needs to be addressed early in the development cycle.

7.7 Oracle Application Server XML Developer's Kit Best Practices

This section describes Oracle Application Server XML Developer's Kit best practices. It includes the following topics:

- [Section 7.7.1, "Choosing XML Parsers"](#)
- [Section 7.7.2, "High-Performance XSLT Transformations"](#)
- [Section 7.7.3, "Streaming XML Schema Validations"](#)

7.7.1 Choosing XML Parsers

Efficient XML parsing is critical to XML applications because it determines how XML data can be accessed. Oracle XDK in Oracle Application Server 10g supports a DOM, SAX and JAXP of XML Parsers:

- Document Object Model (DOM): a W3C standard, which represents XML data as an object tree in memory and provides object-oriented interface to access the data.
- Simple API for XML (SAX): an event-based XML parsing standard in push mode, which represents XML data as a set of events. SAX parsers push out all the events to the registered content handlers through function callbacks.
- Java API for XML Parsing (JAXP): the JSR-63 standard, which provides standard interfaces in Java for both SAX and DOM XML parsing

However, your particular deployment or use may not need all of them. You need to choose the XML parser according to your application requirements so that it is easy-to-use and delivers high performance.

Oracle Application Server DOM implementations employ object-based XML parsing that creates memory objects to store XML data. SAX use event-based or stream-based XML parsing, which fire off events when transversing the XML data. JAXP simply provides a standard Java interface to call either SAX or DOM.

Representing an XML document in-memory as an object tree, using DOM allows dynamic access and updates to the XML content and structure. Therefore, it is good for XML document editing and transformations, which require extensive and random document transversals.

However, object XML processing may lead to high memory costs, especially when processing large XML documents. To overcome this limitation, you can leverage the customized DOM building features using `DOMParser.setNodeFactory()` in XDK to minimize its performance impact. The `DOMParser.setAttribute()` interface

also allow you to avoid including DTD object in DOM by setting the `DOMParser.USE_DTD_ONLY_FOR_VALIDATION` to be `TRUE`.

Stream-based XML parsing, such as SAX, is the performance choice when processing large XML document because of its limited memory use. Therefore, it is good for retrieving, filtering, and searching large XML documents. However, the streaming XML processing is not easy to use. Because the stream-based XML processing does not maintain the hierarchical structure of XML documents, it is not good for XML transformations or XPath content navigations. Because it does not allow XML document updates in place, it is not recommended for implementing document-editing tools.

JAXP is a standard XML parsing interface. However, it introduces extra overheads because the implement wraps around the existing DOM and SAX implementations. Additionally, JAXP applications are not portable in many cases because of the incompatibility of DOM objects across different XML parser implementations. For example, if you use JAXP based on Apache Xerces to parse XML and create DOM, you cannot use the JAXP XSLT interface implemented based on Oracle XDK to transform the DOM.

7.7.2 High-Performance XSLT Transformations

To boost the performance when transforming XML using XSLT, you can use SAX to build `XSLStylesheet` objects and reuse the XSLT objects for subsequent transformations.

In your XSL stylesheets, avoid unconstrained axis like `"/"` because the Oracle XDK XSL processor takes full DOM traversal for this kind of XPath evaluation.

Since the size of a DOM object significantly affects the memory use of XSLT transformations, you need to set `<xsl:strip-space elements="*" />` to reduce. Since whitespaces do not affect the transformation result and they dramatically reduces the size of a DOM, the ultimately result is better performance.

7.7.3 Streaming XML Schema Validations

Oracle XDK provides stream-based XML Schema validation if no key or keyref is defined in the XML schemas. The following is an example of the SAX-based XML schema validation:

```
// Build the XML Schema Object
XSDBuilder builder = new XSDBuilder();
byte [] docbytes = xsd.getBytes();
ByteArrayInputStream in = new ByteArrayInputStream(docbytes);
XMLSchema schemadoc = (XMLSchema)builder.build(in,null);
//Parse the input XML document with Schema Validation
SAXParser parser = new SAXParser();
parser.setXMLSchema(schemadoc);
parser.setValidationMode(XMLParser.SCHEMA_VALIDATION);
parser.parse(xml.getCharacterStream());
```

Since no DOM object is built during XML schema validation, the process is more scalable.

7.8 Java Message Service Best Practices

This section describes JMS best practices. It includes the following topics:

- [Section 7.8.1, "Set the Correct time_to_live Value"](#)

- [Section 7.8.2, "Do Not Grant Execute Privilege of the AQ PL/SQL Package to a User or Role"](#)
- [Section 7.8.3, "Close JMS Resources No Longer Needed"](#)
- [Section 7.8.4, "Reuse JMS Resources Whenever Possible"](#)
- [Section 7.8.5, "Use Debug Tracing to Track Down Problems"](#)
- [Section 7.8.6, "Understand Handle/Interpret JMS Thrown Exceptions"](#)
- [Section 7.8.7, "Ensure You Can Connect to the Server and Database From the Client Computer"](#)
- [Section 7.8.8, "Tune Your Database Based on Load"](#)
- [Section 7.8.9, "OJMS"](#)
- [Section 7.8.10, "OracleAS JMS Best Practices"](#)

7.8.1 Set the Correct `time_to_live` Value

JMS message expiration is set in the `JMSEXPIRATION` header field. If this value is set to zero (the default), then the message will never expire. If the amount of used table space (memory for OC4J) is a concern, then optimally setting the `time_to_live` parameter will keep messages from accumulating. This is especially true in the publish-subscribe domain where messages may sit forever waiting for the final durable subscriber to return to retrieve the message.

7.8.2 Do Not Grant Execute Privilege of the AQ PL/SQL Package to a User or Role

While there are outstanding OJMS session blocking on a dequeue operation this might cause the granting operation to be blocked and even time-out. Granting calls should be executed before other OJMS operations.

Another way to avoid the blocking or time out is to grant roles instead of granting specific privileges to the user directly. AQ has an `AQ_ADMINISTRATOR_ROLE` that can be used, or users may create their own tailored role. You can then grant the execute privilege of a PL/SQL package to this role. Provided the role was created before hand, the granting of the role to the user does not require a lock on the package. This will allow the granting of the role to be executed concurrently with any other OJMS operation.

7.8.3 Close JMS Resources No Longer Needed

When JMS objects like JMS connections, JMS sessions, and JMS consumers are created, they acquire and hold on to server-side database and client-side resources.

If JMS programs do not close JMS objects when they are done using them either during the normal course of operation or at shutdown, then database and client-side resources held by JMS objects are not available for other programs to use.

The JVM implementation does not guarantee that finalizers will kick in and clean-up JMS object held resources in a timely fashion when the JMS program terminates.

7.8.4 Reuse JMS Resources Whenever Possible

JMS objects like JMS connections are heavy weight and acquire database resources not unlike JDBC connection pools. Instead of creating separate JMS connections based on coding convenience, it is recommended that a given JMS client program create only one JMS connection against a given database instance for a given connect string and

reuse this JMS connection by creating multiple JMS sessions against it to perform concurrent JMS operations.

JMS administrable objects like queues, queue tables, durable subscribers are costly to create and lookup. This is because of the database round trips and in some cases, JDBC connection creation and teardown overhead. It is recommended that JMS clients cache JMS administrable objects once they are created or looked up and reuse them rather than create or look them up each time the JMS client wants to enqueue or dequeue a message. The Oracle Application Server Java Object Cache could be used to facilitate this caching.

7.8.5 Use Debug Tracing to Track Down Problems

OJMS allows users to turn debug tracing by setting `oracle.jms.traceLevel` to values between 1 and 5 (1 captures fatal errors only and 5 captures all possible trace information including stack traces and method entries and exits). Debug tracing allows one to track down silent or less understood error conditions.

7.8.6 Understand Handle/Interpret JMS Thrown Exceptions

OJMS is required by the JMS specification to throw particular JMS defined exceptions when certain error/exception conditions occur. In some cases the JMS specification allows or expects OJMS to throw runtime exceptions when certain conditions occur. The JMS client program should be coded to handle these conditions gracefully.

The catch all JMS exception, `JMSEException`, that OJMS is allowed to throw in certain error/exception cases provides information as to why the error/exception occurred as a linked exception in the `JMSEException`. JMS programs should be coded to obtain and interpret the linked exception in some cases.

For instance, when resources like processes, cursors, or tablespaces run out or when database timeouts/deadlocks occur, SQL exceptions are thrown by the backend database, which are presented to the JMS client program as linked SQL exceptions to the catch all `JMSEException` that is thrown. It would be useful for JMS programs to log or interpret the ORA error numbers and strings so that the administrator of the database can take corrective action.

The code segment below illustrates a way to print both the `JMSEException` and its linked Exception:

```
try
{...}
catch (JMSEException jms_ex)
{

    jms_ex.printStackTrace();
    if (jms_ex.getLinkedException() != null)
        jms_ex.getLinkedException().printStackTrace();
}
```

7.8.7 Ensure You Can Connect to the Server and Database From the Client Computer

When debugging JMS connection creation problems or problems with receiving asynchronous messages/notifications make sure that you can:

- Ping the database using `tnsping`
- Connect to the database with its connect string using `sqlplus`

- Resolve the name or the IP address of the server computer from the client (by using a simple program that accesses a socket) and vice versa.

7.8.8 Tune Your Database Based on Load

OJMS performance is greatly improved by proper database tuning. OJMS performance is dependent on AQ enqueue/dequeue performance. AQ performance will not scale even if you run the database on a computer with better physical resources unless the database is tuned to make use of those physical resources.

7.8.9 OJMS

Make sure that the parameters that control the datasources underlying OJMS connections are set appropriately (see datasources best practices). Runtime exceptions that are caused when transaction/inactivity/cache-connection-availability timeouts occur or connection creation attempts fail can lead to MDBs and OJMS objects becoming unusable until the underlying cause is resolved. In some cases the change cannot be made dynamically and in others container redeployment maybe needed for the changes to take effect.

Make sure that the EJB-location is used to look up the emulated datasources underlying JMS Connections and MDB instances in case these MDBs and JMS Connections need to participate in Container Managed Transactions. If datasource location is used instead MDBs will not receive messages and JMS Connections will not participate in CMTs.

Since datasource support is not available in OC4J for the application client deployment mode in this release, OJMS requires that the JMS code use a URL definition to access OJMS resources within application clients. When the application is a standalone client (that is, when it runs outside of OC4J), configure the <resource-provider> element with a URL property that has the URL of the database where OJMS is installed and, if necessary, provides the username and password for that database. The following demonstrates a URL configuration:

```
<resource-provider class="oracle.jms.OjmsContext" name="ojmsdemo">
  <description> OJMS/AQ </description>
  <property name="url" value="jdbc:oracle:thin:@hostname:port number:SID">
  </property>
  <property name="username" value="user">
  </property>
  <property name="password" value="passwd">
  </property>
</resource-provider>
```

Since OC4J does not support distributed transactions in the application client deployment mode in this release, OJMS only supports local transactions through it's transacted sessions. If the JMS application requires transaction co-ordination then make sure that it is deployed inside a container and not as an application client.

In this release OC4J optimizes transacted Oracle database operations (including OJMS operations) so that a 2-PC is not required for them in a distributed transaction that they are involved in when the operations take place against the same Oracle database instance and the same database schema. So if possible make sure that the AQ queues against which OJMS operations take place are located in the same schema if the applications using them will perform distributed operations against them. when the underlying database version is 10g or later, if the database operations (including OJMS operations) take place against different schemas then for the first time the transacted operations take place a complete 2-PC with prepare and commit phases is performed

and in subsequent operations a 1-PC optimization kicks in. So if it is required that the AQ queues and database tables being used in a distributed transaction be on different schemas, make sure you upgrade to database 10g.

7.8.10 OracleAS JMS Best Practices

OracleAS JMS does not validate invalid configuration information (like host and port) at OC4J start up time and these misconfigurations manifest themselves at runtime as java JMS exceptions. So make sure that the configuration information that you are providing is correct before deploying your JMS application.

OracleAS JMS throws a java.lang.instantiation exception during OC4J startup when the port specified for the JMS Server is already in use. So make sure that the port specified is not already in use when starting up the OC4J instance with a JMS Server enabled.

Make sure that run-time exceptions do not occur in the onMessage call of an MDB instance that uses OracleAS JMS by catching the exceptions in a try-catch block (if feasible). This is because in this release runtime exceptions in the onMessage call can cause the MDB to enter into an endless redelivery loop.

Oracle Application Server Portal

This chapter describes best practices for Oracle Application Server Portal (OracleAS Portal). It includes the following topics:

- [Section 8.1, "Installation, Configuration, Administration, and Troubleshooting Best Practices"](#)
- [Section 8.2, "OracleAS Portal Performance"](#)
- [Section 8.3, "Performance Features for OracleAS Portal"](#)
- [Section 8.4, "Content Management and Publishing"](#)
- [Section 8.5, "Export and Import Best Practices"](#)

8.1 Installation, Configuration, Administration, and Troubleshooting Best Practices

This section describes installation, configuration, administration, and troubleshooting best practices. It includes the following topics:

- [Section 8.1.1, "Use OracleAS RepCA"](#)
- [Section 8.1.2, "Use the Dependency Settings File and Tool"](#)
- [Section 8.1.3, "Configure the Diagnostic Log File for Improved Diagnostics"](#)
- [Section 8.1.4, "Review the Oracle Application Server Portal Configuration Guide for Installation and Configuration Troubleshooting Advice"](#)

8.1.1 Use OracleAS RepCA

To separate the database containing OracleAS Portal metadata from the database that the Identity Management infrastructure resides in, use the following steps:

1. Use Oracle Universal Installer to install and configure the Identity Management infrastructure.
2. Use the Oracle Application Server Repository Creation Assistant (OracleAS RepCA) to install the OracleAS Metadata Repository into an existing (customer) database.
3. Use Oracle Universal Installer or the Oracle Enterprise Manager Application Server Control to configure your OracleAS Portal middle-tier to use the Identity Management infrastructure and the OracleAS Metadata Repository installed in step one and two.

Oracle recommends against using `ptlasst` to create OracleAS Portal schemas in Oracle Application Server.

See Also: ■ *Oracle Application Server Repository Creation Assistant
Installing the Oracle Application Server Metadata Repository into an
Existing Database*

- *Oracle Application Server Portal Configuration Guide*
- *Oracle Application Server 10g Installation Guide*

8.1.2 Use the Dependency Settings File and Tool

If possible, use the OracleAS Portal Dependency Settings file, `iasconfig.xml`, and tool, `ptlconfig`, to perform middle-tier configuration. If you use `ptlasst`, the OracleAS Portal Dependency Settings file does not get updated. Using the `iasconfig.xml` file for subsequent configurations may cause your site to be mis-configured.

For more information refer to the *Oracle Application Server Portal Configuration Guide*.

8.1.3 Configure the Diagnostic Log File for Improved Diagnostics

The OracleAS Metadata Repository contains an OracleAS Portal database schema in which all the OracleAS Portal metadata, content, and PL/SQL code are stored. The PL/SQL code that executes in the OracleAS Portal schema generates diagnostic output that can be correlated with diagnostic output generated from other components of Oracle Application Server. Be sure to configure the database containing the OracleAS Metadata Repository to allow the output of the diagnostic log file. In addition, you must also register this log file with the Application Server Control to achieve the most holistic view of OracleAS Portal Diagnostics.

For more information about the OracleAS Metadata Repository and using the Application Server Control Console Log Viewer refer to the *Oracle Application Server Portal Configuration Guide*.

8.1.4 Review the *Oracle Application Server Portal Configuration Guide* for Installation and Configuration Troubleshooting Advice

The *Oracle Application Server Portal Configuration Guide* discusses various approaches for resolving and diagnosing various configuration issues with OracleAS Portal.

8.2 OracleAS Portal Performance

This section describes OracleAS Portal performance. It features the following topics:

- [Section 8.2.1, "Use Appropriate Caching Strategy"](#)
- [Section 8.2.2, "Use Web and Database Providers Judiciously"](#)
- [Section 8.2.3, "Improve Availability and Scalability"](#)
- [Section 8.2.4, "Scale OracleAS Portal by Tuning"](#)
- [Section 8.2.5, "mod_plsql Tuning Impacts Performance"](#)
- [Section 8.2.6, "Leverage Web Provider Session Caching"](#)
- [Section 8.2.7, "Increase Execution Speed of Slow Portlet"](#)
- [Section 8.2.8, "Reduce Page Complexity to Improve Cachability"](#)

- [Section 8.2.9, "Measure Tuning Effectiveness to Improve Performance"](#)

8.2.1 Use Appropriate Caching Strategy

OracleAS Portal provides two different caching mechanisms to improve performance:

- Out-of-the-box integration with OracleAS Web Cache using in-memory cache solution
- Persistent file-based cache

By default, OracleAS Portal issues dynamic caching instructions to OracleAS Web Cache so that default content (such as pages) can be cached. The page designer can use the different cache options (whole page, page definition, or none) to ensure that the correct balance is maintained between speedy delivery of cached content and avoiding the delivery of stale content. It is important that the page or portlet designer understand that the degree of dynamism of a Web page is inversely proportional to its cachability. Page designers should fully understand validation, expiration, and invalidation-based caching so that they can select the most appropriate cache method for their pages.

The OracleAS Portal file-based cache is always maintained alongside OracleAS Web Cache, providing redundant cover should the in-memory solution fail. The file-based cache is also useful for things like persistent sessions. When you need to maintain a record of either an authenticated session or perhaps a session context as part of a larger atomic transaction. You can improve file-cache performance by locating the cache on a RAM disk, especially the session cache for cached session cookies. The session cache benefits the most from this approach, as this area of the cache suffers a high input-output load in a heavily subscribed secure portal.

OracleAS Portal has enhanced its page caching options in Oracle Application Server, introducing the ability to cache page and portlets at the system level, thus storing a single copy of each object for all users. Here are the page caching options available in OracleAS Portal:

- **Cache Page Definition Only**

With this option, you can create a cached copy of the page definition in both OracleAS Web Cache and the OracleAS Portal cache for each user. The page definition includes:

- Metadata describing the page structure
- Identification of the portlets that the page contains
- Style information for the page

Choose this option for the following types of pages:

- Pages with highly dynamic content
- Pages that contain portlets with short expiry periods
- Pages that contain portlets using validation-based caching or invalidation-based caching

- **Cache Page Definition And Content For [] Minutes**

With this option, you can create a cached copy of the page definition and page content, including the rendered content of all portlets, for a specified period. The fully assembled page is cached by the Web browser and in the OracleAS Portal cache. Choose this option for pages with more static content and long expiry

periods. If you select this option, you may want to include a **Refresh** link on the page to regenerate the page content from the database.

- **Cache Page Definition Only at System Level**

By using this option, you can create a single cached copy of the page definition in the system cache for all users. Because the page definition is the same for all users, page customization options are disabled. This caching option greatly reduces storage requirements and improves performance. Select this option for pages with highly dynamic content as long as they do not require customization.

- **Cache Page Definition And Content at System Level for [] Minutes**

With this option, you can create a single cached copy of the page definition and page content, including the rendered content of all portlets. The cached information remains the same for all users in the system cache for a specified period. Page customizations are not possible because the page definition and content is the same for all users. Select this option for pages that are more static and unlikely to change within the specified period. Note that with this option, portlets display public content only. If you select this option, you may want to include a Refresh link on the page to regenerate the page content from the database.

- **Do not Cache**

Use this option to disable page caching. If you select this option, it may adversely affect your OracleAS Portal performance, as dynamic page generation places a heavy load on both the database and the middle tier.

By default, OracleAS Web Cache is installed, configured, and co-located with the OracleAS Portal middle-tier. For optimal performance, deploy OracleAS Web Cache on a dedicated computer.

See Also: [Chapter 5, "Performance and Scalability"](#)

- **Portlet Caching**

A performance-related enhancement in OracleAS Portal is the ability to cache portlets at the system level, which places a single copy of the portlet in the system cache for all users.

Before caching portlets at the system level, consider the following:

- Caching a portlet at the system level disables all customization options for the portlet.
- Caching a portlet at the system level does not enforce access privileges for the portlet.
- Caching a portlet at the system level means that only public data is displayed. Therefore, portlets such as the Recent Objects portlet or the External Applications portlet, which do not contain public data, are not displayed.
- If a page is cached at system level, and that page happens to contain a portlet, then that portlet is also cached at the system level.
- If the Web provider specifies system-level caching for a portlet, then the portlet is cached at the system level. However, you have the option to change the cache setting for the portlet manually.
- If a portlet is not cached at the system level, but the page on which it appears caches both the page definition and the content, then this portlet is cached at

the user level. With user-level caching, the portlet can be customized and access privileges can be enforced.

In summary, do not cache a portlet at the system level if that portlet includes sensitive data or other information that you do not want to display to all users. Examples of content that may be suitable for system-level caching include page banners and news portlets.

8.2.2 Use Web and Database Providers Judiciously

There are two different types of providers: Web providers and database providers. Using the right type of provider for your portlets can help improve your portal performance.

Database providers are implemented in Java or PL/SQL and are executed as stored procedures within the Oracle database. The OracleAS Portal middle-tier communicates with these providers in two ways:

- through `mod_plsql`, if the provider resides in the local OracleAS Portal database
- by SOAP over HTTP, if the provider resides in a remote database

Note that a portlet that executes in the database is not restricted in terms of functionality, as database facilities allow for external communication in many ways, including HTTP connections to external content. Database providers are particularly appropriate for portlets that require significant interaction with the database, and in situations where the development team has extensive Oracle PL/SQL development experience.

Web providers are implemented in any Web deployment environment (for example: Java, ASP, or Perl) and are executed as applications external to OracleAS Portal. OracleAS Portal communicates with these providers using SOAP over HTTP. Web providers are most appropriate for external information sources (for example: Internet news, business information) and in environments where developers have experience using Java and other Web development languages.

8.2.3 Improve Availability and Scalability

OracleAS Portal provides a parallel page engine (PPE) stateless servlet that fetches page metadata, assembles the page, and manages the cache. Because it is stateless, PPE is a key worker component that can be deployed across multiple OC4J instances.

By default, Oracle Application Server is installed with a single `oc4j_portal` instance in which the PPE servlet is deployed. From a scalability perspective, it is highly recommended that you have at least one other (if not more) OC4J instance that also has the PPE servlet deployed. Alternatively, you can increase the number of OC4J processes dedicated to the single instance. Oracle HTTP Server load balance routing distributes requests across the multiple instances or processes, providing better scalability for the entire system.

8.2.4 Scale OracleAS Portal by Tuning

OracleAS Infrastructure 10g, including the database, provides important functionality to OracleAS Portal, as all metadata, database providers, and infrastructure entities reside there. Because of this heavy dependency on the database, conventional database tuning, such as putting OracleAS Portal indexes on a separate disk, becomes extremely important to optimize OracleAS Portal performance. However, it is not recommended that you analyze the schema for additional tuning opportunities, as this analysis has already been performed for you by the CBO (Cost Based Optimizer) and has been

used where appropriate. Moreover, there are also standard ongoing jobs that re-tune the schema based on collected statistics on a regular basis.

Another aspect of tuning is the SQL*NET tuning between the `mod_plsql` (for example, Oracle HTTP Server) computer and the database itself. You should also consider RAC as an option for the availability and scalability of the OracleAS Portal database.

8.2.5 `mod_plsql` Tuning Impacts Performance

`mod_plsql` maintains its own connection pool, thus removing the need for the use of MTS in most cases. There are, however, some tuning parameters that you can adjust to optimize `mod_plsql` performance and ensure that processes are not shut down heavily:

- Set `MaxClients=MaxSpareServers=average system load`
- Set `MaxRequestPerChild=10000`
- Set `MinSpareServer=1`
- Set `KeepAlive` to off for heavily loaded sites

Tuning these parameters affects the performance of Oracle HTTP Server with respect to non `mod_plsql` requests.

Therefore, if you wish to service other types of requests with the same installation, you can employ a dual listener strategy: Start two Apache listeners, one tuned for standard content requests, and the other tuned for `mod_plsql` content requests. You can find more information about this approach and other factors in the `mod_plsql` section of the *Oracle Application Server 10g Performance Guide*.

8.2.6 Leverage Web Provider Session Caching

When you register a Web provider, you can use a checkbox to cache session-specific information, such as session id and user login time, for each request. Although this is a mandatory requirement for Web providers that rely on session information to ensure the validity of atomic transactions, providers that do not rely on this information should deactivate this option, as doing so improves the portlet cache hit rate.

8.2.7 Increase Execution Speed of Slow Portlet

End user perception of the performance of a page is related to several factors. One of the most obvious factors is the performance of individual portlets. A single slow portlet can slow down the end-user perception of the performance of a whole page. This is because portlets are executed in parallel and the page will not be returned to the user until the slowest portlet either returns content or times out.

Page execution speed, therefore, is equal to the speed of the slowest portlet, plus page assembly overhead.

If you are noting performance issues with a page try using the `_DEBUG` utility. Refer to the *Oracle Application Server Portal Configuration Guide* for more information.

8.2.8 Reduce Page Complexity to Improve Cachability

Page complexity, which is a function of page security and the number of tabs, items, and portlets on a page, affects throughput by increasing the amount of metadata that needs to be generated as well as the number of security and validity checks. Page

complexity does not affect page assembly time in the middle-tier, but may affect the time it takes to validate and refresh portlet content.

8.2.9 Measure Tuning Effectiveness to Improve Performance

One way to evaluate whether your attempts to improve performance are effective is to measure the performance, then use those measurements to further fine tune the system.

To get granular results from system internals, append `&_DEBUG=1` to the end of the portal page URL for which you wish to measure performance. The output is a report from the parallel page engine that provides details of performance of each component on the page, whether a cache miss or hit occurred, and how long page loading took.

Repeating this practice periodically will help keep your system fine-tuned for better performance. You can read more about the information provided by `_DEBUG` in the *Oracle Application Server Portal Configuration Guide*.

8.3 Performance Features for OracleAS Portal

This section describes performance features for OracleAS Portal. It includes the following topics:

- [Section 8.3.1, "Managed Portlet Execution per Page"](#)
- [Section 8.3.2, "Content Pruning"](#)
- [Section 8.3.3, "Search Key Invalidation"](#)

8.3.1 Managed Portlet Execution per Page

The PPE uses the concept of fetcher threads. These are threads within the PPE servlet which are used to service requests for portlet content. By default there are 25 fetcher threads within the PPE waiting to service requests. If a page has 26 portlets and that page is requested then 25 of the portlets will be requested in parallel and the 26th request will wait for the next available fetcher thread.

This serialization effect will ultimately slow down the portal performance as more requests are received. This degradation will affect the whole portal site, to combat this at a more granular page level OracleAS Portal introduces a feature called Managed Portlet Execution (MPE).

This feature provides a throttle effect similar to fetcher threads but on a per-page basis. If a page has 25 portlets 20 will run the other 5 will wait for free slots then they will run, in effect they'll be throttled.

MPE is set to 20 by default, but this is a configurable parameter allowing administrators to ensure that the performance of the whole site is not degraded by over-zealous page designers putting excessive quantities of portlets on one page.

8.3.2 Content Pruning

The file cache under `$ORACLE_HOME/Apache/modplsql/cache` in the Oracle Application Server middle-tier installation has been enhanced so that it now has a sub-directory for explicit storage of the page metadata (PMD). The separate storage of the PMD allows the cleanup processes to be more explicit in their selection of content to be pruned.

Cleanup is now controlled by new explicit parameters:

- PlsqlCacheMaxAge 30
- PlsqlCacheCleanupTime Saturday 23:00

PMD content will be given preferential retention treatment ensuring the greatest cache hit ratios occur for PMD objects. These objects are the most expensive to generate and the least likely to change in a running production site.

8.3.3 Search Key Invalidation

When a document is cached in OracleAS Web Cache it is cached using an invalidation basis. This means the content is stored in OracleAS Web Cache until it is explicitly invalidated by a SOAP message issued by the portal repository. Prior to OracleAS Portal, the portal repository needed to issue a single invalidation message for each piece of content that had changed for each possible cached element.

If a page based on a template is secured and cached on a per user basis for 10 users and the template is changed then the portal must issue 10 invalidation messages.

OracleAS Portal introduces a concept of search key invalidation where the content is cached with a common identifier (search key) that allows the portal repository to issue one invalidation message for the content with the common identifier thus invalidating all the content that matches that key with a single message.

8.4 Content Management and Publishing

The section describes content management and publishing best practices. It includes the following topics:

- [Section 8.4.1, "Use a Single Page Group for Delegating Administration"](#)
- [Section 8.4.2, "Research Your Taxonomy Before Building Up a Page Hierarchy"](#)
- [Section 8.4.3, "Use Page Templates for Consistency"](#)
- [Section 8.4.4, "Use Navigation Pages to Manage Template Content"](#)
- [Section 8.4.5, "Categories, Perspectives, and Custom Attributes"](#)
- [Section 8.4.6, "Understand how Multilingual Content is Managed"](#)
- [Section 8.4.7, "Use Unstructured User Interface Templates"](#)
- [Section 8.4.8, "Use Content Management APIs to Migrate Existing Content"](#)
- [Section 8.4.9, "Use WebDAV Capabilities to Support Desktop Application Centric Users"](#)

8.4.1 Use a Single Page Group for Delegating Administration

OracleAS Portal allows you to organize portal pages within page groups. The easiest way to get started is with a single page group. Within a page group, you can easily copy or move content elements across pages.

However, in a larger environment, you may want different people to administer different areas of the site. Delegating administration is much easier if you separate your site into multiple page groups.

Currently, it is not possible to copy or move content elements between page groups, or to use templates, styles, and metadata elements (categories, perspectives, attributes, and item types) owned by one page group in another page group. However, if you want to use templates, styles, and metadata across page groups, you can place them in

the supplied Shared Object Page Group. In addition, any page can be published as a portlet, which allows the content on that page to be viewed on any other page in any page group.

For more information about configuration and administration refer to the *Oracle Application Server Portal User's Guide*.

8.4.2 Research Your Taxonomy Before Building Up a Page Hierarchy

There are several different ways to organize the content that your portal needs to provide to its end users. This organization is referred to as a taxonomy. You can create a physical taxonomy consisting of pages and sub-pages. You can also create virtual taxonomies consisting of categories and perspectives. Users can browse a taxonomy, which appears as a hierarchy of pages. Each category and perspective page is built dynamically by searching for content belonging to the selected category or perspective when the page is rendered.

Reorganizing the physical taxonomy is easy and can be done by moving pages around within a page group and by moving items between pages (keep in mind that pages and items cannot be moved between page groups). However, reorganizing category and perspective values is not currently supported. Although you can reassign content to a different category or perspective, this must be done manually for each piece of content or programmatically by using the content management APIs. Therefore, carefully plan your category and perspective hierarchies before you start to add content to your pages.

For more information refer to the *Oracle Application Server Portal User's Guide*.

8.4.3 Use Page Templates for Consistency

Creating pages with OracleAS Portal is easy, and it may be tempting to start adding pages quickly at the onset of your portal development project without first defining one or more page templates. To ensure a consistent look and feel to your site and to minimize maintenance effort, it is recommended that you always base your pages on page templates. You can keep this association for the lifecycle of the page in order to enforce a specific look and feel, control the page creation process and minimize maintenance efforts. Alternatively, you can use a page template as a convenient starting point for creating a custom page layout. In this use case, you would create the page using a template (thus inheriting the page layout and region property settings), disassociate the page from the template and make page unique changes to region properties and layout. Keep in mind that you can always re-apply the template to the page or apply a new template if required.

It is also a good practice to manage your templates that could be used in multiple page groups in the Shared Objects page group, as templates cannot be promoted from another page group to Shared Objects.

For more information about OracleAS Portal page templates refer to the *Oracle Application Server Portal User's Guide*.

8.4.4 Use Navigation Pages to Manage Template Content

A page template defines the layout (the placement of item and portlet regions) for pages. A template can also contain content, in the form of portlets and items that you want to appear on all its pages. However, changes made to a template are seen immediately in its dependent pages. Making changes to the content on a template can take minutes or even hours, depending on the extent of the changes. During this time, the pages themselves will be in a state of flux as the template is modified. This can

have an undesirable impact on your portal users and may require that the affected pages be unavailable while performing template maintenance.

You can avoid this by managing template content on navigation pages. For example, use a navigation page to contain the banner for your template, which may include such elements as the Page Name Smart Text, company logo, and various Smart Links such as the Customize icon and the Home Page link. When you want to modify the banner, copy the navigation page and make changes to the copy. When you are satisfied with the changes, replace the original banner navigation page portlet on the template with the modified copy. This can be done very quickly and will have minimal impact on portal users. The same recommendation applies to navigation bars, page footers, and other content that you want to include on the template.

For more information navigation pages refer to the *Oracle Application Server Portal User's Guide*.

8.4.5 Categories, Perspectives, and Custom Attributes

One of the big advantages of OracleAS Portal is the ability to define and associate metadata with any content. OracleAS Portal provides three types of metadata:

- **Categories:** Used for mutually exclusive properties like "What is it?"
- **Perspectives:** Used for content that may require multiple properties values like "Who is the audience?"
- **Custom Attributes:** Used for other mutually exclusive properties

It is important to understand the characteristics of these metadata elements in terms of their impact on content organization, maintenance, presentation, and search. For example, while they all aid in searching for content, each has a different style for search submission and for presentation in search results. There are also important differences in terms of how content contributors assign metadata values to content and in how these elements are presented on pages.

[Table 8–1](#) summarizes the characteristics of the metadata elements.

Table 8–1 Metadata Element Characteristics

Characteristic	Custom Attributes	Categories	Perspectives
Can be mandatory on Add/Edit item	Yes	Yes	Yes
Can be selected for Group By in Region display	No	Yes	No
Can be arranged in a navigable hierarchy	No	Yes	Yes
Allows multiple values for a single item	No	No	Yes
Select values from Static List of Values	Yes	Yes	Yes
Select values from Dynamic List of Values (based on SQL Query)	Yes	No	No
Can be associated with an icon	No	Yes	Yes
Searchable	Yes	Yes	Yes
Can be shown in Item display	Yes	Yes	Yes
Can be shown in Search Results	Yes	Yes	Yes
Can be used to order a custom query (using SQL against the WWSBR_ALL_ITEMS repository view)	No	Yes	No

Table 8–1 (Cont.) Metadata Element Characteristics

Characteristic	Custom Attributes	Categories	Perspectives
Translatable	Yes	Yes	Yes
Data types	Boolean, Date, File, Number, PL/SQL, Text (Single or Multi- Line), URL	Text Only	Text Only

8.4.6 Understand how Multilingual Content is Managed

OracleAS Portal allows you to store, manage, and publish translations of your portal content. Many of the objects that are managed in your portal can be associated with one or more languages in addition to the default language defined for a page group. Translated content is automatically published to viewers of your portal in their selected language.

Although the translation feature is very useful and powerful, it is important to understand how translations are created, managed, published, and queried. In particular, the impact of creating or editing an object in a non-default language should be made clear to your content contributors and portal users. Listed below are several characteristics of the translation feature that are important to understand.

- When edits are unintentionally made to an object in a language other than the default language, the edits may appear to be lost when the object is viewed in the original, language. Content contributors may not realize that multiple language records can exist for a single object.
- When a non-translatable attribute is edited, the value of the attribute is automatically copied to all language records. The content contributor may wonder why the attribute value has suddenly changed after switching to a different language.
- Editing a translatable attribute does not copy the change to all language records. In this case, the content contributor may be concerned that all changes were lost when viewing the object in a different language.
- Translations may exist for one version of an item, but not for another version. If the current version changes, it may seem as if the translation has been lost.

8.4.7 Use Unstructured User Interface Templates

The OracleAS Portal page editing capabilities allow you to create complex page structures and a compelling user interface for a wide range of collaborative and Intranet-based portal configurations.

However, if your portal page calls for a sophisticated graphic design, you can use Unstructured User Interface Templates (also called UI Templates) to help you achieve that design. A UI Templates is HTML code, created in any HTML authoring tool, for the purpose of surrounding an OracleAS Portal page. By placing special OracleAS Portal substitution tags in your HTML, you can place portal elements such as Sub-Page links, an Edit Page link, a Navigator link and other elements common to a portal page directly in your HTML code. The only requirement for an HTML page to

be used as an Unstructured UI Template is the inclusion of the Oracle Application Server substitution tag, #BODY#. The location of the #BODY# tag in the HTML determines where the OracleAS Portal page itself will be rendered. At runtime, this UI Template is combined with the dynamically generated content of the portal page to produce the final result.

By using this technique, you can apply virtually any corporate look to your portal pages, even if this design would not be achievable using the regular portal page design capabilities.

Unstructured UI Templates are created and maintained under **Portlet Providers > Locally Built Providers > Shared Components > User Interface Templates** in the Navigator.

Before using an Unstructured UI Template on a page, the page Page Group properties must be configured to enable UI Templates. Once this is done, use an option on the **Page Properties > Other** tab to select a UI Template for the page. Note that the option to select a UI Template is available only for pages that do not have a Template specified on the **Page Properties > Template** tab. If you would like to use a UI template and a page based on a template together, then the UI Template must be applied to the page template which can then be applied to the page.

For more information about unstructured UI Templates refer to the *Oracle Application Server Portal User's Guide*.

8.4.8 Use Content Management APIs to Migrate Existing Content

When setting up your portal, you may find that you want to load and attribute directory/file structures that exist on a file system into the portal repository. While it is very time consuming to upload and set the attribution for every file using the Web browser interface, there is an alternative. You can use Content Management APIs.

Functions include:

- `add_folder`: creates a new page within a given page group
- `add_item`: uploads a file from the file system and adds a new item to an item region
- `modify_item`: allows you to set the attribution for an existing item and helps you perform content management task programmatically without any Web browser-based interaction

All of the files you plan to upload using the APIs must be accessible from the same file system as the installed Portal Metadata Repository.

8.4.9 Use WebDAV Capabilities to Support Desktop Application Centric Users

OracleAS Portal supports the use of WebDAV clients, such as Microsoft Web Folders, to access the Portal Metadata Repository. This allows users to directly edit a document (like an MS Word document) on a portal page and save it back to the repository without ever having to download or upload the file. It also allows users to publish files located on the local file system to a portal page directly from the Windows Explorer, as well as to delete, copy, or move those files. The same security settings found in the Web browser-based interface are also enforced in the WebDAV environment, using OracleAS Single Sign-On to authenticate to Portal.

For more information refer to the *Oracle Application Server Portal User's Guide*.

8.5 Export and Import Best Practices

This section describes export and import best practices. It includes the following topics:

- [Section 8.5.1, "Review Supported Use Cases Before Performing an Export or Import"](#)
- [Section 8.5.2, "Follow the Guidelines for Export and Import of Portal Objects"](#)

8.5.1 Review Supported Use Cases Before Performing an Export or Import

OracleAS Portal provides a set of export and import utilities to enable you to copy portal content between portal installations. For example, you might use these utilities to copy or update portal page groups and application components between a development instance and a production instance of OracleAS Portal.

It is critical to understand that the provided OracleAS Portal export and import utilities support a specific set of use cases and usage scenarios. Oracle recommends reviewing *Oracle Application Server Portal Configuration Guide* before beginning the page and content design process for your portal if regular export and import of content between portal instances is a requirement. *Oracle Application Server Portal Configuration Guide* provides an overview of the export and import process and key concepts. It also describes the two most common export and import use cases:

- Importing and exporting between development to production instances
- Deploying identical content across multiple portal instances

8.5.2 Follow the Guidelines for Export and Import of Portal Objects

For best practices and recommendations for export and import of the objects defined within OracleAS Portal, refer to the *Oracle Application Server Portal Configuration Guide*. It describes best practices for:

- Migrating Your Users and Groups
- Migrating Your Page Groups and Components
- Migrating Your Web Providers
- Migrating Your Portal DB Providers and Components
- Migrating Your Search Components
- Migrating Your External Applications
- Migrating Your Portal Across Databases

For additional support updates and support information go to Oracle *MetaLink* (<http://metalink.oracle.com>).

Oracle Application Server Wireless

This chapter describes best practices for Oracle Application Server Wireless (OracleAS Wireless). It includes the following topics:

- [Section 9.1, "Deploying Multiple Tiers for High-Volume Environments"](#)
- [Section 9.2, "Firewall Settings"](#)
- [Section 9.3, "Deploying Content Sources"](#)
- [Section 9.4, "Choice of Voice Gateway"](#)
- [Section 9.5, "Deploying Messaging Applications"](#)

9.1 Deploying Multiple Tiers for High-Volume Environments

It is often necessary to deploy Wireless and Voice applications in a high-volume environment where the number of transactions may exceed the capacity of a single Oracle Application Server 10g middle-tier that is associated with an OracleAS Infrastructure 10g. To determine if you need additional middle-tiers for your enterprise, consult sizing information that is available under the **Mobile** section of OTN (<http://otn.oracle.com>).

Since the Device Portal requests usually by far exceed the Customization and Webtool requests, you may want to shut down some OC4J_Portal instances on some of your Oracle Application Server middle tiers. Shut down the instances in proportion to the request ratio. This will free up resources for OC4J_Wireless instances on the middle tiers. However, consider if you will need any Oracle Application Server applications deployed in OC4J_Portal to be available on all middle tiers.

Refer to the *Oracle Application Server 10g Administrator's Guide* for more information on deploying multiple tiers.

9.2 Firewall Settings

A typical OracleAS Wireless request starts from a device to a WAP gateway. The gateway issues an HTTP request for the content to Oracle HTTP Server, which in turn issues an AJP request to the OC4J container. The Wireless application in the container then issues a corresponding HTTP request to a content source. Since these entities may be deployed on separate computers, it is necessary to ensure that the Firewall settings in a DMZ permit these protocols to pass through.

9.3 Deploying Content Sources

Content sources, that is, applications or pages that output XHTML or mobile XML, should be deployed in a JVM other than OC4J_Portal or OC4J_Wireless. You may also consider dedicating a separate instance of the application server if your content source is implemented using Oracle Application Server.

9.4 Choice of Voice Gateway

Applications written in Oracle Application Server presentation independent XML can be delivered:

- to any telephone, either local or wireless
- by audio playback of information
- by a voice-enabled user interface

In the same way that SMS or WAP applications running on OracleAS Wireless can utilize gateways from multiple vendors, Oracle Application Server voice applications can also run on any Oracle-accepted VoiceXML gateway.

The voice gateways include:

- a VoiceXML interpreter
- speech recognition (ASR) or text-to-speech (TTS or synthetic speech) software
- telephony interface cards such as Dialogic, NMS, or AudioCodes

For a list of Oracle-accepted Voice Gateways, refer to the **Partner Solutions** section under the **Mobile** section of OTN (<http://otn.oracle.com>).

9.5 Deploying Messaging Applications

Messaging applications require a gateway; this is most often an SMPP or UCP gateway for sending or receiving SMS messages. It is possible to configure the same short code to multiple SMSCs. This may be necessary if redundancy at the SMSC level is a requirement.

Business Intelligence

This chapter describes best practices for business intelligence. It includes the following topics:

- [Section 10.1, "Oracle Reports"](#)
- [Section 10.2, "Oracle Application Server Discoverer Best Practices"](#)

10.1 Oracle Reports

This section describes best practices for Oracle Reports. It includes the following topics:

- [Section 10.1.1, "Differences Between Paper and Web Reporting"](#)
- [Section 10.1.2, "Dynamic Environment Switching to Consolidate Reports Servers"](#)

10.1.1 Differences Between Paper and Web Reporting

In order to be able to serve the requirements of Oracle Reports customers that use J2EE architecture for their enterprise applications, Oracle Reports introduced Web layout in addition to the paper layout. The Web layout is completely code-based as opposed to the paper layout, which is based on graphical frames, repeating frames, boilerplate objects and so on. When Oracle Reports users need to deploy their reports on the Web, they have 2 options:

- design the reports in paper layout and access the report in HTML/HTMLCSS format
- design the Web layout and access the JSP

The paper layout offers you minute control over pagination. In case you would like to generate HTML output, it also provides several properties that you can use to affect the HTML code in the output. However, in spite of all its capabilities, this format does not offer you full control over the look-and-feel of the HTML output. For example, if you want to alter the width and other attributes of the table that shows your data, it can only be done within the constraints of the graphical capabilities of the report layout designer, and you may not be able to make full use of the HTML or CSS capabilities. The paper layout is quite useful in situations when you do not want to hand-code your HTML report, and you would like to present exactly the same report in HTML as in paginated formats like PDF.

Because Web layout is code-based, it offers minute control over the HTML that appears in the output. You can design your reports to look exactly like the rest of the pages in your application. Using Java code inside the JSP report is more direct and easier than in the paper layout because you have to use PL/SQL logic to call the Java

business logic. Once you have designed the report, you can package the JSP with the rest of the application and deploy it on a J2EE application server. The Web layout is useful when you plan to have a J2EE-based Web application and have expertise in writing Java and HTML code. You can use Web application wizards such as Oracle Reports Wizard or Oracle Graph Wizard to generate JSP code for you however, you will not experience the full potential of a JSP report unless you have a JSP-based Web application team.

10.1.2 Dynamic Environment Switching to Consolidate Reports Servers

Oracle Reports contains a feature called Dynamic Environment Switching. Previously, the Oracle Reports server could only serve reports that were compatible with the operating environment in place when the Oracle Reports server was started. For example, the reports had to be compatible with the value of the NLS_LANG parameter at the time the Oracle Reports server was started. This restriction meant that you needed to have one Oracle Reports server running for each processing language. The new environment switching feature available in Oracle Reports eliminates this restriction by enabling one instance of Oracle Reports server to serve reports with any number of environment settings, including language.

To be able to use this feature, just add as many <environment> tags in the Oracle Reports server configuration file as needed. Each of these tags can have values such as the NLS_LANG setting, a currency symbol, or a calendar. When processing a job, use the EnvID parameter in the command line to specify which <environment> setting you want to use. The Oracle Reports server reads the relevant environment settings from the configuration file, and if an engine is not already running with these settings, a new engine is started. The new engine, started with appropriate environment settings, will be used to process the job.

For more information about the <environment> section of the server configuration file, refer to *Oracle Application Server Reports Services Publishing Reports to the Web* guide.

10.2 Oracle Application Server Discoverer Best Practices

This section describes best practices for Oracle Application Server Discoverer (OracleAS Discoverer).

The performance of a OracleAS Discoverer system refers to the time OracleAS Discoverer takes to complete a specific task. OracleAS Discoverer performance is largely determined by how well the database has been designed and tuned for queries. You can achieve additional performance benefits by designing your Business Areas and Worksheets with performance in mind.

The scalability of a OracleAS Discoverer system refers to the ability of OracleAS Discoverer to handle increasing numbers of users or tasks without compromising performance. To take advantage of inherently scalable architecture of OracleAS Discoverer, install it on multiple computers and share the workload between the computers.

Additional performance and scalability can be achieved by tuning Oracle Application Server and OracleAS Discoverer services on your middle tier installation.

Refer to the *Oracle Application Server Discoverer Configuration Guide* for additional information regarding optimizing performance and scalability of OracleAS Discoverer.

Index

A

- active failover cluster, 4-1
- add_folder, 8-12
- add_item, 8-12
- advanced invalidation
 - OracleAS Web Cache and, 5-14
- AllowOverride, 6-3
- application availability, 2-3
- application server control, 2-2
- Application Server Control home page, 2-4
- asession validation
 - ESI environment tag and, 5-11
- audit log, 3-12
- authentication, 3-5
- authentication callbacks
 - ESI environment tag and, 5-11
- authentication options, 3-7
- authorization, 3-5
- authorization callbacks
 - ESI environment tag and, 5-11
- availability, application, 2-3

B

- basic invalidation
 - OracleAS Web Cache and, 5-13
- bean managed persistence, 7-13
- bulkload.sh, 3-10

C

- cache clusters, 5-2
 - invalidating cache objects, 5-16
 - propagating changes, 5-6
- cache connections, 5-18
- cache hierarchies
 - invalidating cache objects and, 5-16
- cache hits, 5-10
 - increasing, 5-8
- cache page definition, 8-3
- caching rules
 - OracleAS Web Cache and, 5-9
- caching, OracleAS TopLink, 7-29
- category metadata, 8-10
- CDNs

- OracleAS Web Cache and, 5-4
- certificates, 3-3
- cluster, EJB, 7-11
- clusters
 - cache, 5-2
- coarse objects, 7-8
- cold failover cluster, 4-1
- compression
 - OracleAS Web Cache and, 5-19
- confidentiality mode, 3-11
- connection caching, 7-22
- connection retry interval, 7-21
- connections
 - OracleAS Web Cache and, 5-7
- connections, datasource, 7-15
- container managed persistence, 7-12
- container managed relationships, 7-13
- content delivery networks, 5-4
- content sources, 9-2
- control, application server, 2-2
- cookie, 7-8
- cookies, 3-2
 - OracleAS Web Cache and, 5-8
- country EJB, 7-12
- CPU's
 - OracleAS Web Cache and, 5-2
- custom attributes metadata, 8-10

D

- database-based repository, 2-10
- datasource connections, 7-15
- DCM archive, 2-9
- DCM clusters, 2-9
- denial-of-service attacks
 - OracleAS Web Cache and, 5-5
- deployment
 - OracleAS Web Cache and, 5-2, 5-4
- directory integration platform, 3-12
- DMS metrics, 2-12
- document object model, 7-36
- dynamic environment switching, 10-2
- dynamic includes, 7-4
- dynamic scheme, 7-20

E

- Edge Side Includes (ESI), 5-10
- Edge Side Includes for Java (JESI), 5-13
- EJB, 7-10
- EJB cluster, 7-11
- EJB-location, 7-19
- emulated data source, 7-19
- encryption, 3-1, 3-5
- End-User Performance Monitoring, 2-3
- error pages
 - OracleAS Web Cache and, 5-7
- escape processing, 7-16
- ESI environment, 5-11
- ESI environment tag, 5-11
- ESI include tag, 5-11, 5-12
- ESI inline tag, 5-11, 5-12
- ESI variables, 5-10
- event logging
 - OracleAS Web Cache and, 5-19
- event script, 2-7
- expiration, 5-15
 - of cache objects, 5-13
- expiring cache objects, 5-15

F

- fault containment, 3-2
- fetchable fragments, 5-12
- firewall, 3-4
- fixed return null scheme, 7-20
- fixed wait scheme, 7-20
- FollowSymLinks, 6-2
- fragments
 - ESI and, 5-12
 - fetchable, 5-12
 - non-fetchable, 5-12

G

- granular objects, 7-8

H

- hierarchical caching, 5-4
- HostNameLookups, 6-2
- HTTPS protocol
 - OracleAS Web Cache and, 5-5

I

- iHAT, 4-3
- inactivity timeout, 7-21
- infrastructure repository, 2-10
- inline fragments
 - ESI and, 5-11, 5-12
- inline tag
 - ESI, 5-12
- invalidating cache objects, 5-13, 5-15
 - indexes and, 5-16
 - programmatically, 5-15

- invalidation, 5-15
 - advanced, 5-14
 - basic, 5-13
 - programmatic, 5-15
- invalidation propagation
 - in cache clusters, 5-16
- island, 7-8

J

- Java API for XML parsing, 7-36
- Java Server Pages, 7-1
- Java Virtual Machine, 7-5
- JESI tags, 5-13

K

- KeepAlive, 6-2
- KeepAlive httpd.conf directive
 - Web Cache and, 5-17
- Keep-Alive timeout
 - Web Cache setting, 5-18
- KeepAliveTimeout, 6-2
- KeepAliveTimeout httpd.conf directive
 - Web Cache and, 5-17

L

- lazy loading, 7-14
- load balancer
 - OracleAS Web Cache and, 5-3
- load balancing
 - OracleAS Web Cache and, 5-3
- local EJB, 7-10
- locking strategies, 7-12

M

- managed portlet execution, 8-7
- mapping workbench, 7-27
- max connect attempts, 7-22
- MaxClients, 6-2
- MaxClients httpd.conf directive
 - Web Cache and, 5-17
- max-connections, 7-20
- MaxKeepAliveRequests, 6-2
- MaxKeepAliveRequests httpd.conf directive
 - Web Cache and, 5-17
- memory
 - OracleAS Web Cache and, 5-7
- message driven EJB, 7-10
- message facade, 7-12
- method authentication, 3-2
- min-connections, 7-20
- mod_oc4j, 4-4, 6-3
- mod_rewrite, 6-3
- modify_item, 8-12

N

- network bandwidth

- OracleAS Web Cache and, 5-7
- network connections
 - OracleAS Web Cache and, 5-7, 5-18
- network interface cards
 - OracleAS Web Cache and, 5-7
- network load balancer
 - OracleAS Web Cache and, 5-3
- NIC
 - OracleAS Web Cache and, 5-7
- non-emulated data sources, 7-19
- non-fetchable fragments, 5-12

O

- ojspc tool, 7-2
- OPMN, 2-6
- OPMN standard error, 2-6
- OPMN standard output, 2-6
- optimistic locking, 7-12
- Oracle Enterprise Manager 10g Grid Control, 2-5
- Oracle Enterprise Manager deployment wizard, 2-5
- Oracle Enterprise Manager Grid Control, 2-2, 2-4
- Oracle Enterprise Manager job system, 2-5
- Oracle Tuning Pack, 2-4
- OracleAS TopLink caching, 7-29
- origin server capacity
 - OracleAS Web Cache and, 5-18
- origin server timeout
 - Web Cache setting, 5-18

P

- page metadata, 8-7
- paging
 - OracleAS Web Cache and, 5-7
- parallel page engine, 8-5
- partial page caching, 5-11, 5-12
 - Edge Side Includes (ESI), 5-10
- performance metrics, 2-11
- perspective metadata, 8-10
- pessimistic locking, 7-12
- ping, 2-7
- portlet caching, 8-4
- post-crash, 2-8
- prefetching, 7-17
- pre-start, 2-7
- pre-stop, 2-8
- programmatic invalidation, 5-15

R

- redirection
 - OracleAS Web Cache and, 5-9
- remote cache, 1-6
- remote caching, 1-6
- remote EJB, 7-10
- replication, 7-7
- replication overhead, 7-8
- response time
 - OracleAS Web Cache and, 5-17
- response times

- OracleAS Web Cache and, 5-19

S

- secure socket layer, 3-5
- security
 - OracleAS Web Cache and, 5-5
- server and client authentication mode, 3-11
- server authentication, 3-11
- session facade, 7-12
- session state, 7-7
- session timeout, 7-8
- sessions
 - OracleAS Web Cache and, 5-9
- simple API for XML, 7-36
- SQL Access Advisor, 2-4
- SQL Tuning Advisor, 2-4
- SSL, 3-11
- SSL encryption, 3-5
- start element, 2-7
- startproc, 2-7
- stateful inspection, 3-4
- static includes, 7-3
- stop element, 2-7
- substring matching
 - for invalidation, 5-14
- surrogate-control headers
 - OracleAS Web Cache and, 5-9
- swapping
 - OracleAS Web Cache and, 5-7
- SymLinksIfOwnerMatch, 6-2

T

- timeout, 2-7
- time-wait settings
 - Web Cache and, 5-19
- topology viewer, 4-3

U

- UI template, 8-11
- update batching, 7-17
- URL parameters
 - OracleAS Web Cache and, 5-8

V

- value object pattern, 7-12

W

- wait timeout, 7-21
- Web Application Performance page, 2-3
- Web Cache Manager
 - OracleAS Web Cache and, 5-6
- Web layout, 10-1

