

Oracle® Application Server

Performance Guide

10g Release 3 (10.1.3)

B25213-01

January 2006

Primary Author: Thomas Van Raalte

Contributors: Eric Belden, Alice Chan, Greg Cook, Bill Danell, Marcelo Goncalves, Helen Grembowicz, Bruce Irvin, Pushkar Kapasi, Paul Lane, Sharon Malek, Valarie Moore, Carol Orange, Julia Pond, Leela Rao, Ed Rybak, Joan Silverman, Cheryl Smith, Zhunquin Wang, Brian Wright

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, JD Edwards, and PeopleSoft are registered trademarks 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

Preface	vii
Intended Audience.....	vii
Documentation Accessibility	vii
Related Documentation.....	viii
Conventions	viii
 1 Performance Overview	
Introduction to Oracle Application Server Performance	1-2
Performance Terms.....	1-2
What Is Performance Tuning?	1-2
Response Time.....	1-3
System Throughput	1-4
Wait Time	1-4
Critical Resources.....	1-4
Effects of Excessive Demand	1-5
Adjustments to Relieve Problems.....	1-5
Performance Targets	1-6
User Expectations.....	1-6
Performance Evaluation.....	1-6
Performance Methodology	1-6
Factors in Improving Performance.....	1-7
 2 Monitoring Oracle Application Server	
Oracle Enterprise Manager 10g Application Server Control Console	2-2
Oracle Application Server Built-in Performance Metrics	2-2
Centralized Management of Oracle Application Server Instances	2-3
Native Operating System Performance Commands	2-3
Network Performance Monitoring Tools	2-3
 3 Top Performance Areas	
Top Performance Areas	3-2
Ensure Sufficient Hardware Resources	3-3
Ensure Sufficient Java Heap for OC4J.....	3-3
Tune the JVM Garbage Collection Options.....	3-4
Reuse Database Connections.....	3-6

Specify Sufficient Oracle HTTP Server Connections	3-7
Enable Statement Caching for Data Sources	3-8
Verify Database Tuning	3-8
Verify Logging Levels	3-10
Reuse EJB Instances	3-11
Advanced Performance Areas	3-11
Managing Concurrency and Limiting Connections.....	3-11
Load Balancing	3-16
Using the -XX:AppendRatio Option (Sun JVM argument).....	3-19

4 Optimizing PL/SQL Performance

5 Optimizing Oracle HTTP Server

Configuring Oracle HTTP Server Directives	5-1
How Persistent Connections Can Reduce httpd Process Availability	5-3
Oracle HTTP Server Logging Options	5-3
Access Logging.....	5-3
Configuring the HostNameLookups Directive	5-3
Error logging.....	5-4
Oracle HTTP Server Security Performance Considerations	5-4
Oracle HTTP Server Secure Sockets Layer (SSL) Performance Issues	5-4
Oracle HTTP Server Port Tunneling Performance Issues	5-6
Oracle HTTP Server Performance Tips	5-6
Analyze Static Versus Dynamic Requests	5-7
Analyze Time Differences Between Oracle HTTP Server and OC4J Servers.....	5-7
Beware of a Single Data Point Yielding Misleading Results	5-7

A Monitoring Using Built-in Performance Tools

Summary of Oracle Application Server Built-in Performance Metrics	A-2
Viewing Performance Metrics Using AggreSpy	A-2
Using the AggreSpy Display	A-2
AggreSpy URL With a Proxy Server.....	A-4
AggreSpy URL and Access Control	A-4
AggreSpy Limitation When Using Load Balancing With Multiple Instances	A-6
Viewing Performance Metrics Using dmstool	A-6
Access Control for dmstool.....	A-6
Using dmstool to List the Names of All Metrics	A-8
Using dmstool to Report Values for Specific Performance Metrics.....	A-8
Using dmstool With the Interval and Count Options	A-9
Using dmstool to Report All Metrics with Metric Values.....	A-9
Using dmstool to Report All Metrics with Metric Values in XML Format	A-9
Using dmstool to Reset Metric Values	A-10
Using dmstool to View Metrics on a Remote Oracle Application Server System	A-10
Viewing Performance Metrics Using AggreSpy (for Standalone OC4J)	A-10
Using Built-in Performance Metrics on Windows Systems	A-11

B Instrumenting Applications With DMS

Introducing DMS Performance Metrics	B-2
Instrumenting Applications With DMS Metrics.....	B-2
Monitoring DMS Metrics	B-2
Understanding DMS Terminology (Nouns and Sensors)	B-3
DMS Naming Conventions.....	B-6
Adding DMS Instrumentation To Java Applications.....	B-8
Including DMS Imports	B-9
Organizing Performance Data.....	B-9
Defining and Using Metrics for Timing.....	B-10
Defining and Using Metrics for Counting.....	B-11
Defining and Using Metrics for Recording Status Information (State Sensors)	B-12
Validating and Testing Applications Using DMS Metrics.....	B-13
Validating DMS Metrics	B-13
Testing DMS Metrics For Efficiency	B-14
Understanding DMS Security Considerations.....	B-15
Conditional Instrumentation Using DMS Sensor Weight	B-15
Dumping DMS Metrics To Files	B-15
Resetting and Destroying Sensors	B-16
DMS Coding Recommendations.....	B-16
Isolating Expensive Intervals Using PhaseEvent Metrics	B-17
Using A High Resolution Clock To Increase DMS Precision.....	B-17
Configuring DMS Clocks for Reporting Time for OC4J (Java)	B-18
Configuring DMS Clocks for Reporting Time for Oracle HTTP Server	B-20
Rolling Up DMS Data for Descendent Nouns	B-21

C Performance Metrics

Oracle HTTP Server Metrics	C-2
Oracle HTTP Server Child Server Metrics.....	C-2
Oracle HTTP Server Responses Metrics	C-3
Oracle HTTP Server Virtual Host Metrics.....	C-3
Aggregate Module Metrics	C-3
HTTP Server Module Metrics.....	C-3
Oracle HTTP Server mod_oc4j Metrics.....	C-4
Oracle HTTP Server SSL Metrics	C-5
JVM Metrics	C-6
JVM Properties Metrics	C-6
JDBC Metrics.....	C-7
JDBC Driver Metrics	C-7
JDBC Data Source Metrics	C-7
JDBC Driver Specific Connection Metrics	C-8
JDBC Data Source Specific Connection Metrics	C-8
JDBC ConnectionSource Metrics	C-9
JDBC Driver Statement Metrics	C-9
JDBC Data Source Statement Metrics.....	C-10
OC4J Metrics	C-10

Web Module Metrics	C-11
Web Context Metrics	C-11
OC4J Servlet Metrics.....	C-12
OC4J JSP Metrics	C-12
OC4J EJB Metrics	C-13
OC4J OPMN Info Metrics	C-15
OC4J Work Management Pool Metrics	C-15
OC4J JMS Metrics	C-16
JMS Metric Tables	C-16
JMS Stats Metric Table.....	C-17
JMS Request Handler Stats	C-19
JMS Connection Stats.....	C-19
JMS Session Stats	C-20
JMS Message Producer Stats.....	C-20
JMS Message Browser Stats	C-21
JMS Message Consumer Stats	C-21
JMS Durable Subscription Stats	C-21
JMS Destination Stats.....	C-22
JMS Temporary Destination Stats.....	C-22
JMS Store Stats	C-22
JMS Persistence Stats	C-23
OC4J Task Manager Metrics	C-23
mod_plsql Metrics	C-24
Oracle Process Manager and Notification Server - OPMN Metrics	C-28
OPMN_PM Metric Table.....	C-28
OPMN_OC4J_PROC Table	C-29
OPMN_HOST_STATISTICS Metric Table	C-29
OPMN_IAS_INSTANCE Metric Table	C-29
OPMN_IAS_COMPONENT Table.....	C-30
OPMN ONS Metrics	C-31
OPMN_APPCTX Table	C-33
DMS Internal Metrics	C-33

Index

Preface

This guide describes how to monitor and optimize performance, use multiple components for optimal performance, and write highly performant applications in the Oracle Application Server environment.

This preface contains these topics:

- [Intended Audience](#)
- [Documentation Accessibility](#)
- [Related Documentation](#)
- [Conventions](#)

Intended Audience

Oracle Application Server Performance Guide is intended for Internet application developers, Oracle Application Server administrators, database administrators, and Web masters.

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. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

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

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documentation

For more information, see these Oracle resources:

- *Oracle HTTP Server Administrator's Guide*
- *Oracle Application Server Containers for J2EE User's Guide*
- *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide*
- *Oracle Containers for J2EE Servlet Developer's Guide*
- *Oracle Containers for J2EE JSP Tag Libraries and Utilities Reference*
- *Oracle Database Performance Tuning Guide, 10g*
- *Oracle Application Server PL/SQL Web Toolkit Reference*

Conventions

The following text conventions are used in this document:

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

Performance Overview

This chapter discusses Oracle Application Server performance and tuning concepts.

This chapter contains the following sections:

- [Introduction to Oracle Application Server Performance](#)
- [What Is Performance Tuning?](#)
- [Performance Targets](#)
- [Performance Methodology](#)

1.1 Introduction to Oracle Application Server Performance

To maximize Oracle Application Server performance, all components need to be monitored, analyzed, and tuned. The chapters of this guide describe the tools used to monitor performance and the techniques for optimizing the performance of Oracle Application Server components, such as Oracle HTTP Server and Oracle Containers for J2EE (OC4J).

1.1.1 Performance Terms

Following are performance terms used in this book:

concurrency The ability to handle multiple requests simultaneously. Threads and processes are examples of concurrency mechanisms.

contention Competition for resources.

hash A number generated from a string of text with an algorithm. The hash value is substantially smaller than the text itself. Hash numbers are used for security and for faster access to data.

latency The time that one system component spends waiting for another component in order to complete the entire task. Latency can be defined as wasted time. In networking contexts, latency is defined as the travel time of a packet from source to destination.

response time The time between the submission of a request and the receipt of the response.

scalability The ability of a system to provide **throughput** in proportion to, and limited only by, available hardware resources. A scalable system is one that can handle increasing numbers of requests without adversely affecting response time and **throughput**.

service time The time between the receipt of a request and the completion of the response to the request.

think time The time the user is not engaged in actual use of the processor.

throughput The number of requests processed per unit of time.

wait time The time between the submission of the request and initiation of the request.

1.2 What Is Performance Tuning?

Performance must be built in. You must anticipate performance requirements during application analysis and design, and balance the costs and benefits of optimal performance. This section introduces some fundamental concepts:

- [Response Time](#)
- [System Throughput](#)
- [Wait Time](#)
- [Critical Resources](#)

- [Effects of Excessive Demand](#)
- [Adjustments to Relieve Problems](#)

See Also: ["Performance Targets"](#) on page 1-6 for a discussion on performance requirements and determining what parts of the system to tune.

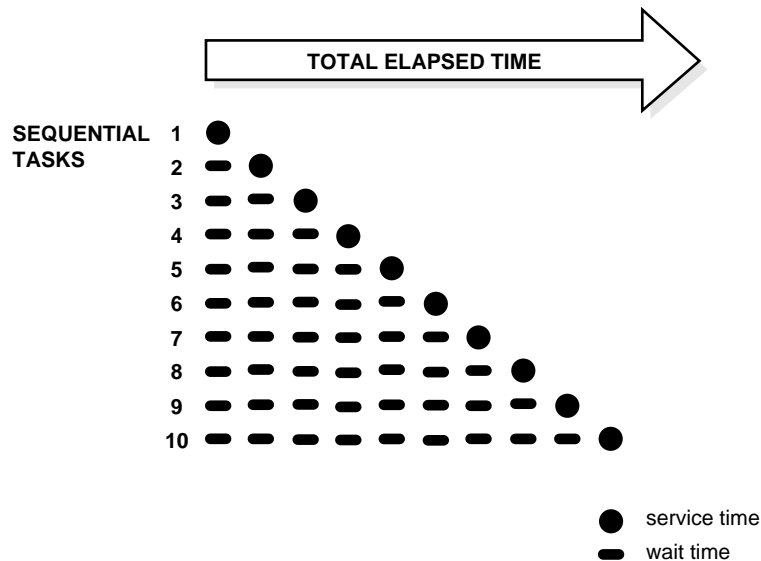
1.2.1 Response Time

Because **response time** equals **service time** plus **wait time**, you can increase performance in this area by:

- Reducing **wait time**
- Reducing **service time**

[Figure 1-1](#) illustrates ten independent sequential tasks competing for a single resource as time elapses.

Figure 1-1 *Sequential Processing of Independent Tasks*



In the example shown in [Figure 1-1](#), only task 1 runs without waiting. Task 2 must wait until task 1 has completed; task 3 must wait until tasks 1 and 2 have completed, and so on. Although the figure shows the independent tasks as the same size, the size of the tasks will vary.

In parallel processing with multiple resources, more resources are available to the tasks. Each independent task executes immediately using its own resource and no **wait time** is involved.

The Oracle HTTP Server processes requests in this fashion, allocating client requests to available httpd processes (or threads). The `MaxClients` directive specifies the maximum number of httpd processes simultaneously available to handle client requests. When the number of processes in use reaches the `MaxClients` value, the server refuses connections until requests are completed and processes are freed.

See Also: [Chapter 5, "Optimizing Oracle HTTP Server"](#)

1.2.2 System Throughput

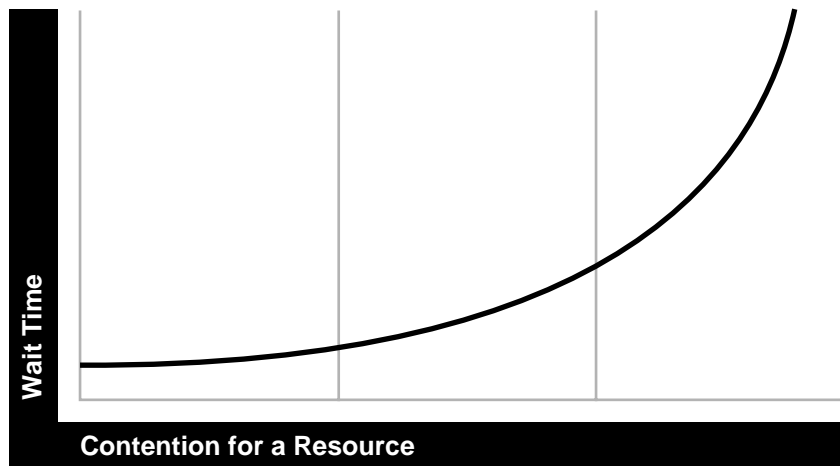
System **throughput** is the amount of work accomplished in a given amount of time. You can increase **throughput** by:

- Reducing **service time**
- Reducing overall **response time** by increasing the amount of scarce resources available. For example, if the system is CPU bound, then adding CPU resources should improve performance.

1.2.3 Wait Time

While the **service time** for a task may stay the same, **wait time** will lengthen with increased **contention**. If many users are waiting for a service that takes one second, the tenth user must wait 9 seconds. [Figure 1–2](#) shows the relationship between **wait time** and resource **contention**. In the figure, the graph illustrates that wait time increases exponentially as contention for a resource increases.

Figure 1–2 *Wait Time Rising With Increased Contention for a Resource*

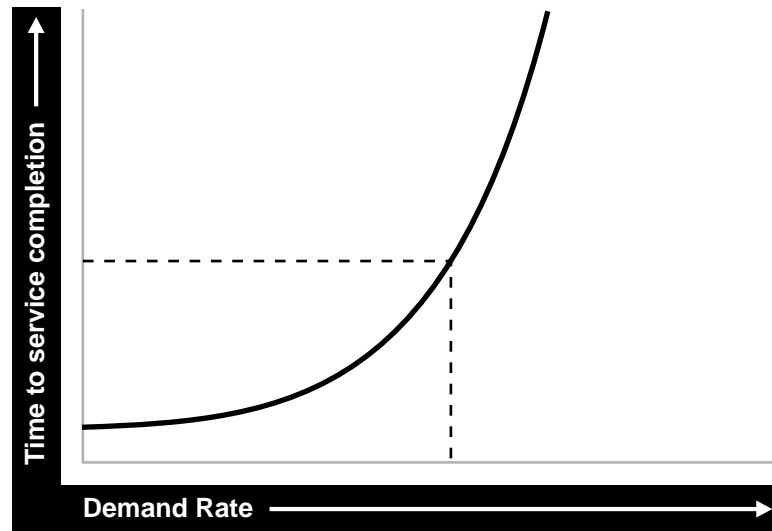


1.2.4 Critical Resources

Resources such as CPU, memory, I/O capacity, and network bandwidth are key to reducing **service time**. Adding resources increases **throughput** and reduces **response time**. Performance depends on these factors:

- How many resources are available?
- How many clients need the resource?
- How long must they wait for the resource?
- How long do they hold the resource?

[Figure 1–3](#) shows the relationship between time to service completion and demand rate. The graph in the figure illustrates that as the number of units requested rises, the time to service completion increases.

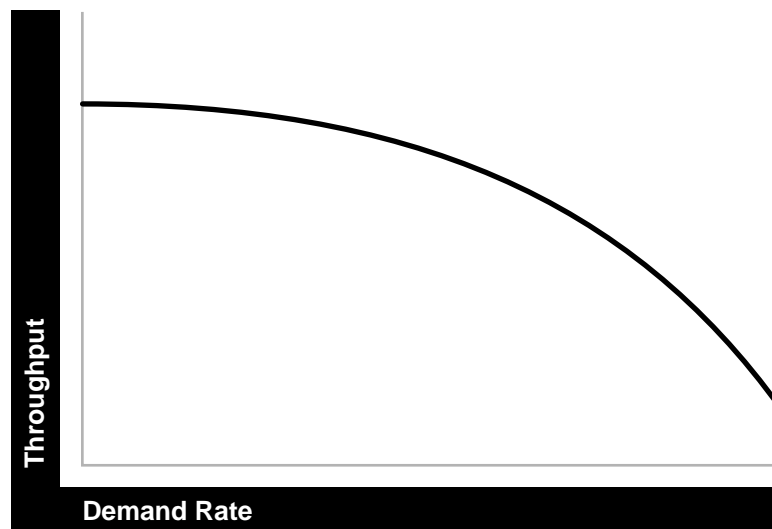
Figure 1–3 Time to Service Completion Versus Demand Rate

To manage this situation, you have two options:

- Limit demand rate to maintain acceptable response times
- Add resources

1.2.5 Effects of Excessive Demand

Excessive demand increases **response time** and reduces **throughput**, as illustrated by the graph in [Figure 1–4](#).

Figure 1–4 Increased Demand/Reduced Throughput

If the demand rate exceeds the achievable **throughput**, then determine through monitoring which resource is exhausted and increase the resource, if possible.

1.2.6 Adjustments to Relieve Problems

Performance problems can be relieved by making adjustments in the following:

- unit consumption
Reducing the resource (CPU, memory) consumption of each request can improve performance. This might be achieved by pooling and caching.
- functional demand
Rescheduling or redistributing the work will relieve some problems.
- capacity
Increasing or reallocating resources (such as CPUs) relieves some problems.

1.3 Performance Targets

Whether you are designing or maintaining a system, you should set specific performance goals so that you know how and what to optimize. If you alter parameters without a specific goal in mind, you can waste time tuning your system without significant gain.

An example of a specific performance goal is an order entry **response time** under three seconds. If the application does not meet that goal, identify the cause (for example, I/O **contention**), and take corrective action. During development, test the application to determine if it meets the designed performance goals.

Tuning usually involves a series of trade-offs. After you have determined the bottlenecks, you may have to modify performance in some other areas to achieve the desired results. For example, if I/O is a problem, you may need to purchase more memory or more disks. If a purchase is not possible, you may have to limit the **concurrency** of the system to achieve the desired performance. However, if you have clearly defined goals for performance, the decision on what to trade for higher performance is easier because you have identified the most important areas.

1.3.1 User Expectations

Application developers, database administrators, and system administrators must be careful to set appropriate performance expectations for users. When the system carries out a particularly complicated operation, **response time** may be slower than when it is performing a simple operation. Users should be made aware of which operations might take longer.

1.3.2 Performance Evaluation

With clearly defined performance goals, you can readily determine when performance tuning has been successful. Success depends on the functional objectives you have established with the user community, your ability to measure whether or not the criteria are being met, and your ability to take corrective action to overcome any exceptions.

Ongoing performance monitoring enables you to maintain a well tuned system. Keeping a history of the application's performance over time enables you to make useful comparisons. With data about actual resource consumption for a range of loads, you can conduct objective **scalability** studies and from these predict the resource requirements for anticipated load volumes.

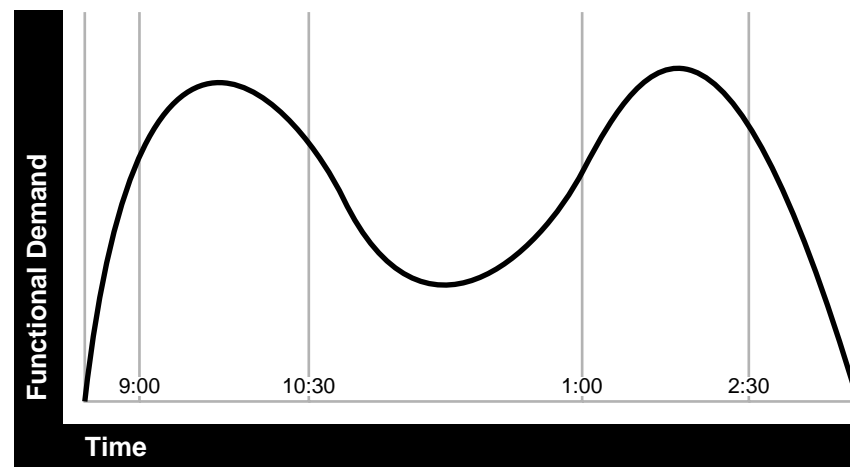
1.4 Performance Methodology

Achieving optimal effectiveness in your system requires planning, monitoring, and periodic adjustment. The first step in performance tuning is to determine the goals you

need to achieve and to design effective usage of available technology into your applications. After implementing your system, it is necessary to periodically monitor and adjust your system. For example, you might want to ensure that 90% of the users experience **response times** no greater than 5 seconds and the maximum **response time** for all users is 20 seconds. Usually, it's not that simple. Your application may include a variety of operations with differing characteristics and acceptable response times. You need to set measurable goals for each of these.

You also need to determine variances in the load. For example, users might access the system heavily between 9:00am and 10:00am and then again between 1:00pm and 2:00pm, as illustrated by the graph in [Figure 1-5](#). If your peak load occurs on a regular basis, for example, daily or weekly, the conventional wisdom is to configure and tune systems to meet your peak load requirements. The lucky users who access the application in off-time will experience better **response times** than your peak-time users. If your peak load is infrequent, you may be willing to tolerate higher **response times** at peak loads for the cost savings of smaller hardware configurations.

Figure 1-5 *Adjusting Capacity and Functional Demand*



1.4.1 Factors in Improving Performance

Performance spans several areas:

- Sizing and configuration: Determining the type of hardware needed to support your performance goals.
- Parameter tuning: Setting configurable parameters to achieve the best performance for your application.
- Performance monitoring: Determining what hardware resources are being used by your application and what **response time** your users are experiencing.
- Troubleshooting: Diagnosing why an application is using excessive hardware resources, or why the **response time** exceeds the desired limit.

Monitoring Oracle Application Server

This chapter provides an overview and presents information on monitoring Oracle Application Server and its components. Monitoring Oracle Application Server and obtaining performance data can assist you in tuning the system and debugging applications with performance problems.

This section describes the available Oracle Application Server tools for performance monitoring. You can monitor the server and its components using one or more of the following:

This chapter contains the following topics:

- [Oracle Enterprise Manager 10g Application Server Control Console](#)
- [Oracle Application Server Built-in Performance Metrics](#)
- [Centralized Management of Oracle Application Server Instances](#)
- [Native Operating System Performance Commands](#)
- [Network Performance Monitoring Tools](#)

2.1 Oracle Enterprise Manager 10g Application Server Control Console

Oracle Enterprise Manager 10g Application Server Control Console (Application Server Control Console) allows you to monitor Oracle Application Server and its components. Application Server Control Console shows performance metrics for Oracle Application Server components, including:

- Oracle Containers for J2EE (OC4J) and Applications running under OC4J

Using Application Server Control Console, you can also view performance metrics and other status information using Application Server Control Console.

See Also: *Oracle Application Server Administrator's Guide*

2.2 Oracle Application Server Built-in Performance Metrics

Oracle Application Server automatically measures runtime performance and collects metrics for Oracle HTTP Server and Oracle Containers for J2EE (OC4J) servers and components. The server performance metrics are measured automatically and continuously using performance instrumentation inserted into the implementations of Oracle Application Server components. The performance metrics are automatically enabled; you do not need to set options or perform any extra configuration to collect them (for performance reasons the JDBC metrics are enabled by setting options).

The Oracle HTTP Server performance metrics enable you to do the following:

- Monitor the duration of important phases of Oracle HTTP Server request processing.
- Collect status information on Oracle HTTP Server requests. For example, you can monitor the number of requests being handled at any given moment.

The OC4J performance metrics allow you to monitor the performance of J2EE containers and components and enable you to do the following:

- Monitor the number of active servlets, JSPs, EJBs, and EJB methods.
- Monitor the time spent processing an individual servlet, JSP, EJB, or EJB method.
- Monitor the sessions and JDBC connections associated with servlets, JSPs, EJBs, or EJB methods.
- Monitor OC4J JMS events and status.

You can use the performance metrics while troubleshooting Oracle Application Server components to help locate bottlenecks, identify resource availability issues, or help tune your components to improve throughput and response times.

Note: You can use the commands that access the built-in metrics in scripts or in combination with other monitoring tools to gather performance data or to check application performance.

See Also:

- [Appendix A, "Monitoring Using Built-in Performance Tools"](#)
- [Appendix C, "Performance Metrics"](#)

2.3 Centralized Management of Oracle Application Server Instances

While Application Server Control Console provides standalone management for an Application Server and its components, you can centrally manage all your Application Servers through one tool rather than through several Application Server Control Consoles by using the Oracle Enterprise Manager 10g Grid Control Console. For example, say you have 10 Application Servers deployed on five hosts. By deploying a Management Agent on each host, Enterprise Manager automatically discovers the Application Server on those hosts and automatically begins monitoring them using default monitoring levels, notification rules, and so on.

The Oracle Enterprise Manager 10g Grid Control Console contains an Application Server Home page which provides easy access to key information required by application server administrators, including the following:

- Links to Oracle Application Server component home pages
- Application server status, responsiveness, and performance data
- Alerts and diagnostic drill-downs so you can identify and resolve problems quickly
- Resource usage for the application server and its components
- A single view of all Java 2 Platform Enterprise Edition (J2EE) applications and web services
- Links to the Application Server Control Console for administration operations such as starting and stopping components, modifying configurations, and deploying applications.

See Also:

Oracle Enterprise Manager Concepts for more information on Oracle Enterprise Manager 10g Grid Control Console

Oracle Application Server Administrator's Guide

Oracle Enterprise Manager Grid Control Installation and Basic Configuration

2.4 Native Operating System Performance Commands

In order to solve performance problems or to monitor your system's activity, you can use the available native operating system commands. Native operating system commands allow you to gather and monitor CPU utilization, paging activity, swapping, and other system activity information.

See Also: Refer to the system level documentation for information on native operating system monitoring commands.

2.5 Network Performance Monitoring Tools

You can use network monitoring tools to verify the status of requests that access your Oracle Application Server components. Tools are available that allow you to examine and save network traffic information. These tools can be helpful in analyzing and solving performance problems.

Top Performance Areas

This chapter provides a description of top tuning areas for Oracle Application Server and includes the following sections:

- [Top Performance Areas](#)
- [Advanced Performance Areas](#)

3.1 Top Performance Areas

This section covers critical Oracle Application Server performance issues and provides a quickstart for tuning J2EE applications that run on OC4J. [Table 3–1](#) lists a quick guide for performance considerations for Oracle Application Server.

Table 3–1 Top Performance Areas for Oracle Application Server Applications

Performance Area	Description and Reference
Ensure Sufficient Hardware Resources	<p>Oracle Application Server has minimum hardware requirements that enable you to use the product. You also need to run on hardware that supports your application's needs, including the database tier. For details on Oracle Application Server installation requirements, see the Requirements chapter in the <i>Installation Guide</i> your platform.</p> <p>See "Ensure Sufficient Hardware Resources" on page 3-3 for more information on platform specific tools that can help you determine if your hardware resources are sufficient.</p>
Ensure Sufficient Java Heap Size	<p>To improve the performance of a J2EE application, you need to provide an adequate heap size for the JVM where the application runs. If the OC4J running your J2EE application does not have enough memory, performance can suffer due to the overhead required to manage limited memory.</p> <p>See "Ensure Sufficient Java Heap for OC4J" on page 3-3 for details on setting JVM heap size options.</p>
Tune the JVM for Garbage Collection	<p>To improve the performance of a J2EE application and ensure that your application performance is not being negatively impacted by JVM garbage collection, you need to manage the heap size and sometimes you also need to set JVM options that impact garbage collection frequency.</p> <p>See "Tune the JVM Garbage Collection Options" on page 3-4 for more information on garbage collection.</p>
Reuse Database Connections	<p>By default, OC4J data sources enable database connection pooling. Thus, OC4J manages database connections to avoid frequently reestablishing new connections. The Oracle Application Server data source facility provides options you can use to control the number of database connections maintained, and how long they are maintained.</p> <p>See "Reuse Database Connections" on page 3-6.</p>
Specify Sufficient HTTP Connections	<p>Tune the Oracle HTTP Server directives to set the level of concurrency by specifying the number of HTTP connections.</p> <p>See "Specify Sufficient Oracle HTTP Server Connections" on page 3-7.</p>
Enable JDBC Statement Caching Option	<p>By enabling statement caching to lower the overhead of repeated cursor creation and repeated statement parsing and creation, you can improve performance for applications using the database tier.</p> <p>See "Enable Statement Caching for Data Sources" on page 3-8.</p>
Ensure the Database is Properly Tuned	<p>For applications that access a database, ensure that your database is properly configured to support your application's requirements.</p> <p>See "Verify Database Tuning" on page 3-8.</p>
Verify Logging Levels	<p>You need to make sure that the logging level is not set higher than the default INFO level logging. If the logging setting does not match the default level, reset the logging level to the default for best performance.</p> <p>See "Verify Logging Levels" on page 3-10.</p>
Reuse EJB Instances	<p>Set the EJB options for creating and reusing instances to improve EJB performance.</p> <p>See "Reuse EJB Instances" on page 3-11.</p>

3.1.1 Ensure Sufficient Hardware Resources

A most crucial performance area is ensuring that there are sufficient CPU, memory and network resources to support the user population and application requirements for your Oracle Application Server installation. You need to monitor resource utilization over an extended period to determine if you have occasional peaks of usage or whether a resource is consistently saturated. You also need to define the acceptable response times and throughputs for applications running at your site, for both peak and extended periods. Also, check the system while running your application under normal load and monitor operating system statistics, including, CPU, memory, disk, and network performance to determine if any hardware resource is saturated.

To check the CPU, memory, and disk performance you can use the following commands:

On Linux systems use the `sar` or `mpstat` command.

On Windows systems use the `perfmon` command.

To check network performance, you can use the following commands:

On Linux and Windows systems:

```
% netstat
```

On Windows systems, you can also use the Windows Task Manager to check network performance.

If any of the hardware resources are saturated, this could be due to one or more of the following:

- The hardware resources are insufficient to run the application.
- The system is not properly configured.
- The application or database needs to be tuned.

For a consistently saturated resource, the solutions are to reduce load or increase resources. For peak traffic periods, if the increased response time is not acceptable the alternatives are to again increase resources or to determine if there is traffic that can be rescheduled to reduce the peak load, such as scheduling batch or background operations during slower periods. Oracle Application Server provides a variety of mechanisms to help you control resource concurrency; this can limit the impact of bursts of traffic. However, for a consistently saturated system, these mechanisms should be viewed as temporary solutions.

See Also:

- ["Specify Sufficient Oracle HTTP Server Connections"](#) on page 3-7
- ["Managing Concurrency and Limiting Connections"](#) on page 3-11

3.1.2 Ensure Sufficient Java Heap for OC4J

If you have sufficient memory available on your system and your application is memory intensive, increase the JVM heap size from the default value. While the amount of heap required varies based on the application and on the available memory, for most OC4J server applications, if you have sufficient memory, then Oracle recommends using a heap size of 512 Megabytes or larger.

You can improve performance by setting the minimum heap size equal to the maximum heap size. To do this set the option `-Xms size` to be the same as the `-Xmx`

size. To change the size of the heap allocated to the OC4J process in an OC4J instance, specify the following JVM options:

```
-Xmssizem -Xmxsizem
```

If your Oracle Application Server topology includes more than one JVM on the same system, then to maximize performance, set the maximum heap size to accommodate application requirements and make sure that the total memory consumed by all of the JVMs running on the system does not exceed the memory capacity of your system.

When you are running an OC4J instance in an Oracle Application Server managed environment, set the Java heap size by including the `-Xms` and `-Xmx` arguments in the `start-parameters java-options` element in the `opmn.xml` file.

For example, the following `opmn.xml` file includes these arguments:

```
<ias-component id="OC4J">
  <process-type id="home" module-id="OC4J" status="enabled">
    <module-data>
      <category id="start-parameters">
        <data id="java-options" value="-Xms512m -Xmx512m . . ."/>
        ...
      </category>
      ...
    </module-data>
    ...
  </process-type>
  ...
</ias-component>
```

See Also:

- The JVM Metrics page in Application Server Control Console. This page is available from the OC4J home page, by clicking the **Performance** Secondary tab, and then, in the Related Links area, clicking **JVM Metrics**.
- You can find detailed information about JVM options and their impact on performance on the JVM vendor's Web sites, such as http://java.sun.com/performance/reference/whitepapers/5.0_performance.html

3.1.3 Tune the JVM Garbage Collection Options

JVM garbage collection is an expensive operation that can have an impact on application performance; inefficient garbage collection can severely degrade application performance. Therefore, it is important to understand how applications create and destroy objects.

To tune the JVM garbage collection options you need to analyze garbage collections data and check for the frequency and type of garbage collections, the size of the memory pools, and the time spent on garbage collection.

In order to determine application memory requirements you can monitor JVM garbage collection and memory pool sizes using the following:

- The JVM options:

```
-verbose:gc
-XX:+PrintGCDetails
```

Look for "Full GC" to identify major collections.

- jstat tool
- visualgc tool
- The Application Server Control Console JVM Metrics page shows JVM memory pool and garbage collector information. This page is available from the OC4J home page by clicking the **Performance** Secondary tab, and then, in the Related Links area, clicking **JVM Metrics**.

Set the `-XX:+AggressiveHeap` JVM option to tune internal VM parameters, and increase the total heap size, as described in "[Ensure Sufficient Java Heap for OC4J](#)" on page 3-3 to reduce the overhead associated with Full CG garbage collections. The `-XX:+AggressiveHeap` option tunes internal VM parameters to be optimal for long-running, memory-intensive workloads. This option should follow, on the command line, the heap sizing options `-Xms` and `-Xmx`. See "[Ensure Sufficient Java Heap for OC4J](#)" on page 3-3 for details on setting Java command line options in an Oracle Application Server managed environment (please disregard older versions of Sun documentation which advise against using `-XX:+AggressiveHeap`).

Note: The JVM provides a variety of parameters to allow you to more finely tune heap management and garbage collection behavior. See the following links for a detailed description of these topics.

To determine if an application uses explicit garbage collection, which can have a negative impact on performance, set the `-XX:+DisableExplicitGC` option. This debugging option disables explicit garbage collection. Applications should avoid the use of `system.gc()` calls. If you suspect an application may be explicitly triggering garbage collection, set this parameter and observe the differences in your garbage collection behavior. If you determine that the application is making explicit `system.gc()` calls, discuss with the application developer why this was done and the impact of disabling the calls. Application developers sometimes use `system.gc()` calls to trigger finalizers. This is not a recommended practice and can yield indeterminate behavior.

See Also:

- http://java.sun.com/j2se/1.5/pdf/jdk50_ts_guide.pdf
- <http://java.sun.com/docs/hotspot/gc5.0/ergo5.html>
- http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html

3.1.4 Reuse Database Connections

To obtain better performance in your application, by lowering the overhead of creating and re-creating database connections, specify the connection pool `min-connections` attribute to set the minimum number of connections that the connection pool maintains.

By default, the value of `min-connections` is 0. For best performance, you should specify a value for `min-connections` other than 0. If `min-connections` is set to a value other than zero, the specified number of connections is maintained; OC4J maintains the connections when they are not in use and they do not time out when the specified `inactivity-timeout` is reached. The `min-connections` attribute does not specify that OC4J pre-create connections at startup. Specify the `initial-limit` attribute to set the number of connections in the connection pool when the pool is initially created or reinitialized. Oracle recommends that you set the `initial-limit` attribute to the same value as the `min-connections` attribute.

If the specified value for `min-connections` is less than `max-connections`, then you should set the `inactivity-timeout` to make sure that connections only time out after an appropriately long period of inactivity. The connection pool `inactivity-timeout` specifies the time, in seconds, to cache unused connections before closing the connection.

To improve performance, you can set the `inactivity-timeout` to a value that allows the connection pool 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 frequently accessed applications 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`, or you can increase the `min-connections`.

Notes for reusing database connections:

1. Limiting the total number of open database connections to a number your database can handle is an important tuning consideration. You should check with your database administrator to make sure that the database is configured to support a number of connections that is greater than the following:

At least as large a number of connections as the sum of the values specified for all the connection pool `min-connections` that could be concurrently active, and as large as the maximum desired concurrency across all the datasources for the database.
2. If the `min-connections` is set to a value other than zero, the specified number of connections is maintained; OC4J maintains the connections when they are not in use, and they do not time out when the specified `inactivity-timeout` is reached.

Once the specified connections are opened, you need to either stop OC4J or use the refresh operation to close the connections. Application Server Control shows the refresh operation in Connection Pool area on the JDBC Resources page. Click the icon in the **Refresh Connection Pool** field to initiate a refresh operation.

3.1.5 Specify Sufficient Oracle HTTP Server Connections

The Oracle HTTP Server `MaxClients` directive limits the number of clients that can simultaneously connect to your web server, and thus the number of `httpd` processes.

For Windows, the analogous parameter is `ThreadsPerChild`. The `Oc4jCacheSize` directive specifies the maximum number of idle connections that `mod_oc4j` maintains per OC4J JVM, relevant only on Windows.

You can use the `MaxClients`, `ThreadsPerChild`, and `Oc4jCacheSize` directives to limit incoming connections to the OC4J instances from the Oracle HTTP Server. This section covers the following topics:

- [Configuring the MaxClients Directive \(for UNIX\)](#)
- [Configuring the ThreadsPerChild Directive \(for Windows\)](#)
- [Configuring the Oc4jCacheSize Directive](#)

Note: This discussion in this section only applies for the default Oracle HTTP Server supplied with Oracle Application Server (based on Apache 1.3). This discussion does not apply for the Apache 2.0 based standalone version of Oracle HTTP Server.

3.1.5.1 Configuring the MaxClients Directive (for UNIX)

You can configure the `MaxClients` directive in the `httpd.conf` file up to a maximum of 8K (the default value is 150). If your system is not resource-saturated and you have a user population of more than 150 concurrent `http` connections, you can improve your performance by increasing `MaxClients` to increase server concurrency. Increase `MaxClients` until your system becomes fully utilized (85% is a good threshold).

When system resources are saturated, increasing `MaxClients` does not improve performance. In this case, the `MaxClients` value could be reduced as a throttle on the number of concurrent requests on the server.

If the server handles persistent connections, then it may require sufficient concurrent `httpd` server processes to handle both active and idle connections. When you specify `MaxClients` to act as a throttle for system concurrency, you need to consider that persistent idle `httpd` connections also consume `httpd` processes. Specifically, the number of connections includes the currently active persistent and non-persistent connections and the idle persistent connections. A persistent, `KeepAlive`, `http` connection consumes an `httpd` child process, or thread, for the duration of the connection, even if no requests are currently being processed for the connection.

If you have sufficient capacity, `KeepAlive` should be enabled; using persistent connections improves performance and prevents wasting CPU resources reestablishing HTTP connections. Normally, you should not need to change `KeepAlive` parameters.

Note: The default maximum requests for a persistent connection is 100, as specified with the `MaxKeepAliveRequests` directive in `httpd.conf`. By default, the server waits for 15 seconds between requests from a client before closing a connection, as specified with the `KeepAliveTimeout` directive in `httpd.conf`.

When there are no httpd processes available, connection requests are queued in the TCP/IP system until a process becomes available, and eventually clients terminate connections.

3.1.5.2 Configuring the `ThreadsPerChild` Directive (for Windows)

You can configure the `ThreadsPerChild` directive in the `httpd.conf` file up to a maximum of 8K (the default value is 50). The `ThreadsPerChild` parameter on Windows systems works like the `MaxClients` parameter on UNIX systems.

See Also: ["Configuring the MaxClients Directive \(for UNIX\)"](#) on page 3-7

3.1.5.3 Configuring the `Oc4jCacheSize` Directive

The `Oc4jCacheSize` directive specifies the maximum number of idle connections that `mod_oc4j` maintains per OC4J JVM. On Windows only, it is sometimes useful to change the default value of this directive.

On UNIX systems where each Oracle HTTP Server process is single threaded, the only meaningful values are 1 which is the default value, and zero (0). A value of zero (0) specifies that Oracle HTTP Server should not maintain any connections and should open a new connection for every request. Since each process is single threaded, a process never needs more than one connection and hence a value of 1 or greater has the same effect on UNIX systems. For best performance, on UNIX systems, do not change the default value for `Oc4jCacheSize`.

On Windows systems, the default `Oc4jCacheSize` value is 75% of the value of `ThreadsPerChild`; the connection cache is shared among threads in the child process. If Oracle HTTP Server is serving a mixed load of static content along with OC4J requests, then the default should be adequate. If the user's load is all OC4J requests, that is, Oracle HTTP Server serves up little or no content and serves just as a front end for OC4J, then it is a good idea to set `Oc4jCacheSize` equal to `ThreadsPerChild`. This setting provides a dedicated connection per thread, if needed, and should give the best performance.

3.1.6 Enable Statement Caching for Data Sources

Enable statement caching to lower the overhead of repeated cursor creation and repeated statement parsing and creation by setting the `num-cached-statements` attribute to a value greater than 0 (the default value is 0, disabled). The number you set for `num-cached-statements` should be the number of SQL statements that you use in your application.

3.1.7 Verify Database Tuning

To achieve optimal performance in Oracle Application Server, for applications that use the database, the database tables you access need to be designed with performance in mind and you need to monitor and tune the database server to assure that the system is performant.

This section covers the following:

- [Tuning `init.ora` Database Parameters](#)
- [Tuning Redo Logs Location and Sizing](#)
- [Automatic Segment-Space Management \(ASSM\)](#)

See Also: *Oracle Database Performance Tuning Guide*

3.1.7.1 Tuning init.ora Database Parameters

Table 3–2 shows tuning information for several of the init.ora database initialization parameters.

Table 3–2 Important init.ora Database Tuning Parameters

init.ora Parameter	Description
DB_BLOCK_SIZE	<p>The default block size of 8K is optimal for most systems. However, OLTP systems occasionally benefit from smaller block sizes, and DSS systems occasionally benefit from larger block sizes.</p> <p>See Also: table 8-3, "Block Size Advantages and Disadvantages" in the <i>Oracle Database Performance Tuning Guide</i>.</p>
PGA_AGGREGATE_TARGET	<p>Specifies the target aggregate PGA memory available to all server processes attached to the instance.</p> <p>See Also: "Memory Configuration and Use" in the <i>Oracle Database Performance Tuning Guide</i> for information on PGA memory management.</p>
PROCESSES	<p>Sets the maximum number of operating system processes that can be connected to Oracle concurrently. The value of this parameter must be 6 or greater (5 for the background processes plus 1 for each user process). For example, if you plan to have 50 concurrent users, set this parameter to at least 55. Many other initialization parameter values are deduced from this value.</p>
SGA_MAX_SIZE	<p>This parameter is the maximum size of the SGA for a running instance. Set this parameter to the amount of memory that you want dedicated for the SGA, which includes the following memory pools:</p> <ul style="list-style-type: none"> ■ Database buffer cache ■ Shared pool ■ Large pool ■ Java pool <p>It is a good practice to regularly monitor the buffer cache hit ratio and size the SGA so that the buffer cache has an adequate number of frames for the workload. The buffer cache hit ratio may be calculated from data in the view V\$SYSSTAT. Also the view V\$DB_CACHE_ADVICE provides data that can be used to tune the buffer cache.</p> <p>See Also: the chapter, "Memory Configuration and Use" in the <i>Oracle Database Performance Tuning Guide</i> for detailed information on how to set the SGA_MAX_SIZE parameter, including on how to use the V\$SYSSTAT and V\$DB_CACHE_ADVICE views to optimize the buffer cache hit ratio.</p>
SGA_TARGET	<p>Setting this parameter to a nonzero value enables Automatic Shared Memory Management. Oracle strongly recommends the use of automatic memory management, both to simplify configuration and to improve performance. Automatic Shared Memory Management was introduced with the Oracle Database 10g (10.1). For prior versions, you must manually configure individual SGA memory pools.</p> <p>See Also: the section, "Automatic Shared Memory Management" in the Chapter, "Memory Configuration and Use" in the <i>Oracle Database Performance Tuning Guide</i> for details on choosing a value for the SGA_TARGET parameter.</p>
UNDO_TABLESPACE UNDO_MANAGEMENT	<p>Oracle strongly recommends that you use automatic undo management (UNDO_MANAGEMENT = AUTO) and manage undo space using an UNDO_TABLESPACE. For backward compatibility reasons, the default value of UNDO_MANAGEMENT is MANUAL.</p> <p>See Also: <i>Oracle Database Performance Tuning Guide</i> for additional information on undo space management.</p>

3.1.7.2 Tuning Redo Logs Location and Sizing

Managing the database I/O load balancing is a non-trivial task. However, tuning the redo log options can provide performance improvement for applications running in an Oracle Application Server environment, and in some cases, you can significantly improve I/O throughput by moving the redo logs to a separate disk.

The size of the redo log files can also influence performance, because the behavior of the database writer and archiver processes depend on the redo log sizes. Generally, larger redo log files provide better performance by reducing checkpoint activity. It is not possible to provide a specific size recommendation for redo log files, but redo log files in the range of a hundred megabytes to a few gigabytes are considered reasonable. Size your online redo log files according to the amount of redo your system generates. A rough guide is to switch logs at most once every twenty minutes. Set the initialization parameter `LOG_CHECKPOINTS_TO_ALERT = true` to have checkpoint times written to the alert file.

The complete set of required redo log files can be created during database creation. After they are created, the size of a redo log size cannot be changed. However, new, larger files can be added later, and the original (smaller) ones can subsequently be dropped.

See Also: The chapters, "Configuring a Database for Performance" and "I/O Configuration and Design" in the *Oracle Database Performance Tuning Guide*

3.1.7.3 Automatic Segment-Space Management (ASM)

For permanent tablespaces, Oracle recommends using automatic segment-space management. Such tablespaces, often referred to as bitmap tablespaces, are locally managed tablespaces with bitmap segment space management.

For backward compatibility, the default local tablespace segment-space management mode is `MANUAL`.

See Also: *Oracle Database Concepts* for a discussion of free space management, and *Oracle Database Administrator's Guide* for more information on creating and using automatic segment-space management for tablespaces.

3.1.8 Verify Logging Levels

You need to assure that application and server logging levels are set appropriately, and that debugging properties or other application level debugging flags are set to appropriate levels or disabled. Set Oracle Application Server OC4J logger log levels to log messages at the `INFO` level (do not set log levels to levels that produce more diagnostic message, including the `FINE` or `TRACE` levels).

To configure OC4J component loggers through Application Server Control Console, do the following from the OC4J home page:

1. Click the Administration link
2. Click Logger Configuration under Properties.
3. Set the root Log Level to the desired value, or expand the tree to select individual Log Levels for specified loggers.
4. Click **Apply** to apply your changes to the OC4J runtime.

3.1.9 Reuse EJB Instances

This section describes EJB options for creating and reusing instances that you can tune to improve EJB performance; these attributes are specific to OC4J and apply for all types of EJBs (except Stateful Session EJBs). You can configure these options by setting attributes in `orion-ejb-jar.xml`.

The `min-instances` attribute specifies the minimum number of bean implementation instances to be kept instantiated or pooled. The default value is 0. For best performance, you should specify a value for `min-instances` other than 0. If `min-instances` is `> 0`, OC4J maintains the `min-instances` number of instances in the pool when they are not in use. For instances above the `min-instances`, the instances are removed from the pool after the `pool-cache-timeout` specified timeout expires.

The default value for the `pool-cache-timeout` is 60 seconds, which is typically too low for frequently accessed EJBs. If the `pool-cache-timeout` is 0 or negative, then the `pool-cache-timeout` is disabled and beans are not removed from the pool.

For performance tuning, try to reduce the frequency of the removal of beans from the pool by setting the `pool-cache-timeout` to a large value. You should set the `pool-cache-timeout` to a large enough value to allow OC4J to avoid destroying and then re-creating instances while your J2EE application is running.

3.2 Advanced Performance Areas

This section describes areas that can provide improved performance for some usage cases and environments.

This section covers the following topics:

- [Managing Concurrency and Limiting Connections](#)
- [Load Balancing](#)
- [Using the -XX:AppendRatio Option \(Sun JVM argument\)](#)

3.2.1 Managing Concurrency and Limiting Connections

Oracle Application Server lets you limit concurrency at multiple layers of the system to match specific usage needs. In addition to controlling HTTP connections, you can control concurrency at additional levels of the product to meet specific usage requirements.

This section covers the following topics:

- [Using OC4J Thread Pools to Control Concurrency](#)
- [Setting the Maximum Number of Connections for Data Sources](#)
- [Controlling the Number of EJB Instances When Using EJBs](#)
- [Limiting Remote EJB Client Connections](#)

See Also: ["Specify Sufficient Oracle HTTP Server Connections"](#) on page 3-7

3.2.1.1 Using OC4J Thread Pools to Control Concurrency

By default, OC4J creates two thread pools, as described in the following list. New threads are created and added to the pools on an as-needed basis. You can limit the number of threads that OC4J creates for the tasks required to handle application requests. The default behavior should be sufficient for most common usage scenarios.

- 2 Thread Pool Configuration

Application Server Thread Pool contains the pool of worker threads plus the threads used for RMI server connection threads. The default behavior allows unbounded threads and threads are created on demand.

Work Management Thread Pool contains the pool of threads that OC4J reserves specifically for use by deployed resource adapters.

Alternatively, you can divide the Application Server Thread Pool into two pools to segregate the RMI connection threads into their own pool. In this case, the Application Server Thread Pool is divided into a worker thread pool and a connection thread pool.

- 3 Thread Pool Configuration

Application Server Thread Pools

- Worker Thread Pool contains the pool of worker threads
- Connection Thread Pool contains the RMI server connection threads

Work Management Thread Pool contains the pool of threads that OC4J reserves specifically for use by deployed resource adapters.

Note: Using the thread pool management options is considered an expert-mode task. When specifying thread pool settings, it is important to remember that the resulting concurrency will be determined by the sum of all the active thread pools in the OC4J process.

Specifying thread pool options can improve performance in certain cases, including the following:

- A hardware resource is nearing maximum capacity on Oracle Application Server 10g. For example, consider the case where a site has a user population of 1000 users with the site often processing 20 concurrent requests; at this rate, the site is nearing full resource utilization. Occasionally, the site needs to handle peaks of 75 to 100 concurrent requests. Specifying a thread pool with a maximum number of threads value in the range of 20 to 25 is likely to yield better overall results (setting the value in the range of 75 to 100 is unlikely to improve peak performance, since at this level there are no additional resources available to service the threads).
- A hardware resource is nearing maximum capacity on the database or the database is a bottleneck due to other limitations, and you want to limit connections to the database using OC4J thread control (in addition, you can limit database connections using the data-source `max-connections` parameter).

Thread pool tuning does not improve performance in the following situations:

- There are sufficient hardware resources available on the system when the system is under maximum load and there are no other known software issues such as database locking problems. This includes both Oracle Application Server 10g tier and database systems. For example, consider the case where a site has a user population of 1000 users but typically only sees a few concurrent requests. In this

case, using application server thread pool tuning to limit threads is not advantageous, since the number of threads is determined by the concurrent request rate, which for this case is very low.

- There already are sufficient concurrency limits specified elsewhere in Oracle Application Server 10g or in the database systems. For example when the Oracle HTTP Server `MaxClients` directive is set to control concurrency, at the HTTP server level, or when the data source `max-connections` attribute is set to control concurrency for connections to the database.

Note: If the number of incoming requests is consistently higher than the request rate that the physical hardware can support, consider increasing the site's physical resources. Likewise, if the response time at peak periods is unacceptable, then you may need to increase the hardware configuration to remedy this situation.

See Also:

- ["Specify Sufficient Oracle HTTP Server Connections"](#) on page 3-7
- ["Setting the Maximum Number of Connections for Data Sources"](#) on page 3-15

3.2.1.1.1 Controlling and Using Application Server Thread Pools This section describes how to manage thread settings for a 2-Pool configuration. See ["Using the Work Manager Thread Pool"](#) on page 3-15 for information on options for specifying limitations for the Work Manager Thread Pool.

You can manage application server thread pools through any of the following:

- Adding the `<global-thread-pool>` element with appropriate attributes set in `server.xml`. For example,

```
<global-thread-pool min="30" max="30" queue="60" \>
```
- Updating the attributes in the `ApplicationServerThreadPool` MBean, which is accessible through the system MBean Browser in Application Server Control Console.
- Using the Thread Pool Configuration page in Application Server Control Console. You access this page by starting at the Administration secondary tab and then clicking the Thread Pool Configuration task (see [Figure 3-1](#)).

Figure 3–1 Application Server Control Console Thread Pool Configuration

ORACLE Enterprise Manager 10g
Application Server Control

Cluster Topology > Application Server: rel3.tvanraal-pc.us.oracle.com > OC4J: home >

Thread Pool Configuration

Page Refreshed Nov 17, 2005 3:58:31 PM PST

Thread pools create and store threads for use and re-use by an OC4J process. Re-using existing threads rather than creating new threads on demand improves performance and reduces the burden on the JVM and underlying operating system.

Name ▲	Current Pool Size	Minimum Pool Size	Maximum Pool Size	Keep Alive		Queue Capacity	Current Queue Size	Debug Mode
				Duration	Unit			
Application Server Thread Pool	20	20	20	10	minutes	80	0	<input type="checkbox"/>
Work Management Thread Pool	2	1	2147483647	10	minutes	0	0	<input type="checkbox"/>

Copyright © 1996, 2005, Oracle. All rights reserved.

Notes for specifying thread pool options:

1. Oracle recommends setting the min value (Minimum Pool Size) equal to the max value (Maximum Pool Size) for the simple case. When min=max, you should set the queue to a value equal to the maximum concurrent number of requests you expect. For example, if you are using Oracle HTTP Server with one OC4J instance and no direct RMI connections, then the value of `MaxClients` would represent your maximum concurrency. Note: if you set `MaxClients` to a very large number, a setting of 300 or less for the queue size is probably sufficient. See ["Validating and Monitoring Thread Pool Performance"](#) on page 3-15 for instructions on monitoring your thread pool and queues.

Oracle recommends starting with a smaller number of threads, for example set min and max to a value in the range of 15 to 40. Monitor your resulting performance with this value. If the site has sufficient CPU and memory resources available, and the concurrent request rate is higher than the current thread count setting, try increasing the number of threads. If you observe that the system is resource saturated, then try reducing the number of threads. Specific thread settings depend on the characteristics of your application.

2. The simplest case is to set min value = max value. However, if you do set a larger max value to accommodate occasional peaks of traffic, OC4J will add a new thread on each request until the minimum number of threads are created. After the minimum number of threads are created, new threads will not be created until the queue is full. OC4J attempts to keep the number of threads at or near the minimum, unless the queue is full. If you set the min value less than the max value, it is generally advisable to keep the queue size small; the queue default value is 80. If you do not see your threads increasing beyond the minimum value, then you should decrease the queue size.

See Also: *Oracle Containers for J2EE Configuration and Administration Guide* for more details on using the application server thread pool

3.2.1.1.2 Using the Connection Thread Pool The application server thread pool supports a second thread pool for RMI connection threads. By default the RMI connection threads are allocated from the application server worker thread pool. To create two pools you configure the min, max, queue, and keepAlive attributes and the cx-min, cx-max, cx-queue, and cx-keepAlive attributes in the `<global-thread-pool>` element.

Defining the second pool using the `cx-` attributes puts a separate limit on the RMI connections; they are not taken from the worker thread pool. In either case, the work from the RMI connections executes from threads in the application server worker thread pool. Specify the connection pool to separate potentially long-lived connection threads from the threads used for application work. This frees the threads in the worker thread pool to do work, instead of being allocated to long lived connections.

Note: Specifying the second thread pool, the connection pool, is only needed if you anticipate many concurrent RMI connections, where the number varies greatly and you are trying to bound the active worker threads.

See Also: Section, "Managing Thread Pool Configurations", in Chapter 11, Task Manager and Thread Pool Configuration, in the *Oracle Containers for J2EE Configuration and Administration Guide*

3.2.1.1.3 Using the Work Manager Thread Pool Starting with 10g Release 3 (10.1.3), EJBs of type MDB use receiver threads with the JMS Resource Adapter. OC4J allocates these threads from the `work-manager-thread-pool` (not from the application server thread pool). However, you need to consider receiver threads to control the overall concurrency on the system. Oracle recommends leaving the `work-manager-thread-pool` at the default setting (by default `work-manager-thread-pool` threads are unlimited). If you want to control concurrency for EJB MDBs, use the JMS ReceiverThreads maximum values. The overall concurrency limit on your system includes the `global-thread-pool` max threads plus the sum of all the MDB receiverThreads configured for your applications deployed to OC4J, or the `max work-manager-thread-pool` setting if smaller.

3.2.1.1.4 Validating and Monitoring Thread Pool Performance You can use the Application Server Control Console **Current Pool Size** and **Current Queue Size** metrics to observe OC4J thread use on your system. You can access these metrics by starting at the Administration secondary tab and then clicking the Thread Pool Configuration task (see [Figure 3-1](#)). The **Current Pool Size** shows the current number of threads in the pool. The **Current Queue Size** shows the current number of requests waiting in the queue for a thread to become available. Oracle recommends starting with the default settings and observing the behavior on your system. You should also observe these metrics after setting any thread pool or queue size limits for your instance.

3.2.1.2 Setting the Maximum Number of Connections for Data Sources

For applications that use a database, performance can improve when the connection pool associated with a data source limits the number of connections. You can use the `max-connections` attribute to limit the database requests from Oracle Application Server so that incoming requests do not saturate the database, or to limit the database requests so that the database access does not overload Oracle Application Server-tier resource.

The connection pool `max-connections` attribute specifies the maximum number of connections that a connection pool allows. By default, the value of `max-connections` is set to -1 (unlimited). For best performance, you should specify a value for `max-connections` that matches the number appropriate to your database performance characteristics.

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 `max-connections` option, as specified in all the applications that access the database.

3.2.1.3 Controlling the Number of EJB Instances When Using EJBs

You may want to limit the number of EJB instances to reduce memory usage or to control concurrency to reduce contention on resources that the EJBs use (for example a data source).

The `max-instances` parameter specifies the number of bean instances allowed in memory – either instantiated or pooled.

- For all types of EJBs except stateful session beans, when the `max-instances` value is reached, and a new EJB is requested, the container waits the number of milliseconds set in the `call-timeout` attribute to see if a bean instance becomes available in the pool. If no bean instance is available in the pool then a `TimeoutExpiredException` is thrown back to the client.
- For stateful session beans, when the `max-instances` value is reached, the container attempts to passivate the oldest bean instance from memory. If unsuccessful, the container waits the number of milliseconds set in the `call-timeout` attribute to see if a bean instance is removed from memory, either through passivation, using the `remove()` method, or by bean expiration before a `TimeoutExpiredException` is thrown back to the client.

To allow an unlimited number of bean instances, use `max-instances = 0` (the default value is 0).

Set `max-instances < 0`, for example to -1, to disable instance pooling. In this case OC4J creates a new bean instance or context when starting the EJB call, and releases the context and throws the instance away to Non-existence state at the end of the call.

The exception, `com.evermind.server.ejb.TimeoutExpiredException: timeout expired waiting for an instance`, occurs when there is no available EJB instance. To avoid this problem set the `max-instances` and `call-timeout` parameters appropriately.

3.2.1.4 Limiting Remote EJB Client Connections

To limit remote EJB client connections, you can use the global thread pool features that control the maximum number of threads that service incoming EJB clients. By configuring the `<global-thread-pool>` in `server.xml` to use two thread pools, you can set the parameter `cx-max` to limit remote EJB client connections.

3.2.2 Load Balancing

Oracle Application Server provides load-balancing features that spread the J2EE application load and incoming requests among multiple application server instances, which generally results in higher throughput and shorter response time. Using multiple application server instances with load-balancing allows you to improve performance by directing requests across the multiple application server instances. In addition, you can use multiple application server instances running on multiple hosts to handle high availability and failover needs.

This section covers the following topics:

- [Configuring Multiple Oracle Application Server Instances](#)
- [Web Application Load Balancing](#)
- [EJB Application Load Balancing](#)

Note: The Oracle Application Server features that provide replication for failover with Web sessions and for stateful session EJBs have a performance overhead; only use these features if you need replication for failover.

3.2.2.1 Configuring Multiple Oracle Application Server Instances

This section covers the following:

- [Determining the Number of OC4J Processes](#)
- [Partitioning Applications into Different OC4J Instances](#)

3.2.2.1.1 Determining the Number of OC4J Processes Determining the optimal ratio of OC4J processes to available CPUs is dependent on the characteristics of the applications you run, the OC4J configuration, the hardware configuration, and the type and number of expected incoming requests. In hardware configurations with a small number of CPUs, you may only need one OC4J instance.

Adding OC4J instances beyond the available resources of the system does not improve performance. For example, if one OC4J instance is sufficient to saturate the CPU resources of a system, adding additional OC4J instances is not likely to improve performance and may, in fact, degrade it. A good starting point is to initially configure one OC4J instance and measure the performance improvement from adding additional OC4J instances.

See Also:

- *Oracle Application Server High Availability Guide*
- *Oracle Containers for J2EE Configuration and Administration Guide*

3.2.2.1.2 Partitioning Applications into Different OC4J Instances Partitioning different applications to be run under different OC4J instances, each of which has different requirements, may help improve the performance of your applications. In this case, you may want to configure different OC4J instances to service the different applications. After deploying the applications to different OC4J instances, you can monitor the performance to see if the overall throughput increases, or the response time decreases.

3.2.2.2 Web Application Load Balancing

In an Oracle Application Server environment, the Oracle HTTP Server uses `mod_oc4j` to load balance requests between the available OC4J instances. In this environment you can select `mod_oc4j` configuration options to choose the appropriate load balancing policies to improve performance. By default, the requests are routed using a roundrobin algorithm.

At many sites Oracle Application Server uses the Oracle HTTP Server module `mod_oc4j` to load balance incoming stateless HTTP requests. By selecting the appropriate load balancing policy for `mod_oc4j` you can improve performance on your site.

The `mod_oc4j` module supports several configurable load balancing policies, including the following:

- Round robin routing (this is the default `mod_oc4j` load balancing policy)
- Random routing
- Round robin or random with local affinity routing, using the `local` option
- Round robin or random with host-level weighted routing, using the `weighted` option

Note: For a session based request `mod_oc4j` always directs the request to the original OC4J which created the session, unless the original OC4J process is not available. In case of failure, `mod_oc4j` sends the request to another OC4J within the same group as the original request (either within same host if available, or on a different host).

Recommendations for Load Balancing with `mod_oc4j`:

1. On a Single Host for both Oracle HTTP Server and OC4J, the default load balancing policy, round robin load balancing is recommended. Random load balancing typically gives comparable performance.
2. With Oracle HTTP Server on a separate host from OC4J:
 - Using a Single OC4J host the default load balancing policy, round robin load balancing is recommended. Random load balancing typically gives comparable performance.
 - Using Multiple OC4J hosts, if all OC4J hosts provide comparable capacity, the default load balancing is recommended. Note that the number of OC4J processes started on a host will implicitly weight the number of requests sent to that host even with random or round robin load balancing. A host with 4 OC4J processes will receive 4 times as many requests as a host with 1 OC4J process.
 - With Multiple OC4J Hosts with varying hardware resources or capacity, you may wish to weight the number of requests sent to each host explicitly to match its capacity. In this case, use either the round robin with weighted option or the random with weighted option. If the hosts have comparable capacity, use simple random or round robin load balancing.

For example, to configure the `mod_oc4j` module in Oracle HTTP Server to specify round robin with a routing weight of 3 for `Host_A` and a routing weight of 1 for `Host_B`, add the following directives to `mod_oc4j.conf`:

```
Oc4jSelectMethod roundrobin:weighted
Oc4jRoutingWeight Host_A 3
```

In this example, you do not need to specify a routing weight for `Host_B`, since the default routing weight is 1.

3. With Multiple hosts with Oracle HTTP Server and OC4J on each host:
 - Multiple hosts with Oracle HTTP Server and OC4J on each host, and a hardware load balancer. Select the local affinity option to direct `mod_oc4j` to only select the local OC4J processes to service incoming requests. This will generally improve performance. When no local OC4J processes are available, `mod_oc4j` selects from the list of available remote OC4J processes.

For example, to select the round robin policy with local affinity, specify the following directive in `mod_oc4j.conf`:

```
Oc4jSelectMethod roundrobin:local
```

3.2.2.3 EJB Application Load Balancing

After an EJB application is deployed to multiple OC4J instances, an EJB client-side application can load balance its requests across the available OC4J instances. To use load balancing, the client-side application configures the JNDI properties to use load balancing. For good performance in some clients, you need to set `oracle.j2ee.rmi.loadBalance=context` to load balance for every initialcontext call, rather than only once for the entire client.

See Also:

- "Configuring ORMI Request Load Balancing" in the *Oracle Containers for J2EE Services Guide*
- "Understanding OC4J EJB Application Clustering Services" in the *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide*

3.2.3 Using the -XX:AppendRatio Option (Sun JVM argument)

With the Sun 5.0 JVM on Linux, under some circumstances under heavy load, synchronization in an application can result in thread starvation. This may cause some requests for your application to appear hung or to timeout after a long time.

If you believe your installation has this problem, you can also try the following JDK parameter: `-XX:AppendRatio=3`. This setting will help to balance the load, but may decrease your overall system throughput. We only recommend this setting for installations experiencing problems due to this bug.

See Also: See the SUN bug database for a description of this issue and the suggested workaround:

http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4985566

Optimizing PL/SQL Performance

This chapter provides references to the information that describes improving PL/SQL performance for web applications. Most of this information is in the Oracle Application Server `mod_plsql` User's Guide.

See Also:

- *Oracle Application Server mod_plsql User's Guide* for information on optimizing PL/SQL performance
- [Appendix C, "Performance Metrics"](#) for information on `mod_plsql` metrics
- *Oracle HTTP Server Administrator's Guide* for details on DAD Parameters
- *Oracle Application Server PL/SQL Web Toolkit Reference* for information on the PL/SQL Web Toolkit that enables you to develop Web applications as PL/SQL procedures stored in an Oracle database server

Optimizing Oracle HTTP Server

This chapter discusses the techniques for optimizing Oracle HTTP Server performance in Oracle Application Server.

This chapter contains:

- [Configuring Oracle HTTP Server Directives](#)
- [Oracle HTTP Server Logging Options](#)
- [Oracle HTTP Server Security Performance Considerations](#)
- [Oracle HTTP Server Performance Tips](#)

5.1 Configuring Oracle HTTP Server Directives

Oracle HTTP Server uses directives in `httpd.conf` to configure the application server. This configuration file specifies the maximum number of HTTP requests that can be processed simultaneously, logging details, and certain limits and timeouts.

Table 5–1 lists directives that may be significant for performance.

Table 5–1 Oracle HTTP Server Configuration Properties

Directive	Description
<code>ListenBackLog</code>	<p>Specifies the maximum length of the queue of pending connections. Generally no tuning is needed or desired. Note that some Operating Systems do not use exactly what is specified as the backlog, but use a number based on, but normally larger than, what is set.</p> <p>Default Value: 511</p>
<code>MaxClients</code>	<p>Specifies a limit on the total number of servers running, that is, a limit on the number of clients who can simultaneously connect. If the number of client connections reaches this limit, then subsequent requests are queued in the TCP/IP system up to the limit specified with the <code>ListenBackLog</code> directive (after the queue of pending connections is full, new requests generate connection errors until a process becomes available).</p> <p>The maximum allowed value for <code>MaxClients</code> is 8192 (8K).</p> <p>Default Value: 150</p>
<code>MaxRequestsPerChild</code>	<p>The number of requests each child process is allowed to process before the child dies. The child will exit so as to avoid problems after prolonged use when Apache (and maybe the libraries it uses) leak memory or other resources. On most systems, this isn't really needed, but some UNIX systems have notable leaks in the libraries. For these platforms, set <code>MaxRequestsPerChild</code> to something like 10000 or so; a setting of 0 means unlimited.</p> <p>This value does not include <code>KeepAlive</code> requests after the initial request per connection. For example, if a child process handles an initial request and 10 subsequent "keep alive" requests, it would only count as 1 request toward this limit.</p> <p>Note: On Windows systems <code>MaxRequestsPerChild</code> should always be set to 0 (unlimited). On Windows there is only one server process, so it is not a good idea to limit this process.</p>
<code>MaxSpareServers</code> <code>MinSpareServers</code>	<p>Server-pool size regulation. Rather than making you guess how many server processes you need, Oracle HTTP Server dynamically adapts to the load it sees, that is, it tries to maintain enough server processes to handle the current load, plus a few spare servers to handle transient load spikes (for example, multiple simultaneous requests from a single Netscape browser).</p> <p>It does this by periodically checking how many servers are waiting for a request. If there are fewer than <code>MinSpareServers</code>, it creates a new spare. If there are more than <code>MaxSpareServers</code>, some of the spares die off.</p> <p>The default values are probably ok for most sites.</p> <p>Default Values:</p> <p><code>MaxSpareServers</code>: 10</p> <p><code>MinSpareServers</code>: 5</p>
<code>StartServers</code>	<p>Number of servers to start initially. If you expect a sudden load after restart, set this value based on the number child servers required.</p> <p>Default Value: 5</p>
<code>Timeout</code>	<p>The number of seconds before incoming receives and outgoing sends time out.</p> <p>Default Value: 300</p>

Table 5–1 (Cont.) Oracle HTTP Server Configuration Properties

Directive	Description
KeepAlive	Whether or not to allow persistent connections (more than one request per connection). Set to <code>Off</code> to deactivate. Default Value: On
MaxKeepAliveRequests	The maximum number of requests to allow during a persistent connection. Set to 0 to allow an unlimited amount. If you have long client sessions, you might want to increase this value. Default Value: 100
KeepAliveTimeout	Number of seconds to wait for the next request from the same client on the same connection. Default Value: 15 seconds

5.1.1 How Persistent Connections Can Reduce httpd Process Availability

The default settings for the `KeepAlive` directives are:

```
KeepAlive on
MaxKeepAliveRequests 100
KeepAliveTimeout 15
```

These settings allow enough requests per connection and time between requests to reap the benefits of the persistent connections, while minimizing the drawbacks. You should consider the size and behavior of your own user population in setting these values on your system. For example, if you have a large user population and the users make small infrequent requests, you may want to reduce the `keepAlive` directive default settings, or even set `KeepAlive` to off. If you have a small population of users that return to your site frequently, you may want to increase the settings.

5.2 Oracle HTTP Server Logging Options

This section discusses types of logging, log levels, and the performance implications for using logging.

5.2.1 Access Logging

For static page requests, access logging of the default fields results in a 2-3% performance cost.

5.2.2 Configuring the `HostNameLookups` Directive

By default, the `HostNameLookups` directive is set to `Off`. The server writes the IP addresses of incoming requests to the log files. When `HostNameLookups` is set to on, the server queries the DNS system on the Internet to find the host name associated with the IP address of each request, then writes the host names to the log.

Performance degraded by about 3% (best case) in Oracle in-house 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 high. Unless you really need to have host names in your logs in real time, it is best to log IP addresses.

On UNIX systems, you can resolve IP addresses to host names off-line, with the `logresolve` utility found in the `$ORACLE_HOME/Apache/Apache/bin/` directory.

5.2.3 Error logging

The server notes unusual activity in an error log. The `ErrorLog` and `LogLevel` directives identify the log file and the level of detail of the messages recorded. The default level is `warn`. There was no difference in static page performance on a loaded system between the `warn`, `info`, and `debug` levels.

For requests that use dynamic resources, for example requests that use `mod_ossso`, `mod_plsql`, or `mod_oc4j`, there is a performance cost associated with setting higher debugging levels, such as the `debug` level.

5.3 Oracle HTTP Server Security Performance Considerations

This section covers the following topics:

- [Oracle HTTP Server Secure Sockets Layer \(SSL\) Performance Issues](#)
- [Oracle HTTP Server Port Tunneling Performance Issues](#)

5.3.1 Oracle HTTP Server Secure Sockets Layer (SSL) Performance Issues

Secure Sockets Layer (SSL) is a protocol developed by Netscape Communications Corporation that provides authentication and encrypted communication over the Internet. Conceptually, SSL resides between the application layer and the transport layer on the protocol stack. While SSL is technically an application-independent protocol, it has become a standard for providing security over HTTP, and all major web browsers support SSL.

SSL can become a bottleneck in both the responsiveness and the scalability of a web-based application. Where SSL is required, the performance challenges of the protocol should be carefully considered. Session management, in particular session creation and initialization, is generally the most costly part of using the SSL protocol, in terms of performance.

This section covers the following SSL Performance related information:

- [Oracle HTTP Server SSL Caching](#)
- [SSL Application Level Data Encryption](#)
- [SSL Performance Recommendations](#)

See Also: *Oracle Application Server Security Guide*

5.3.1.1 Oracle HTTP Server SSL Caching

When an SSL connection is initialized, a session based handshake between client and server occurs that involves the negotiation of a cipher suite, the exchange of a private key for data encryption, and server and, optionally, client authentication through digitally-signed certificates.

After the SSL session state has been initiated between a client and a server, the server can avoid the session creation handshake in subsequent SSL requests by saving and reusing the session state. The Oracle HTTP Server caches a client's Secure Sockets Layer (SSL) session information by default. With session caching, only the first connection to the server incurs high latency.

The `SSLSessionCacheTimeout` directive in `httpd.conf` determines how long the server keeps a saved SSL session (the default is 300 seconds). Session state is discarded if it is not used after the specified time period, and any subsequent SSL request must establish a new SSL session and begin the handshake again. The

`SSLSessionCache` directive specifies the location for saved SSL session information, the default location on UNIX is the `$ORACLE_HOME/Apache/Apache/logs/` directory or on Windows systems, `%ORACLE_HOME%\Apache\Apache\logs\`. Multiple Oracle HTTP Server processes can use a saved session cache file.

Saving SSL session state can significantly improve performance for applications using SSL. For example, in a simple test to connect and disconnect to an SSL-enabled server, the elapsed time for 5 connections was 11.4 seconds without SSL session caching. With SSL session caching enabled, the elapsed time for 5 round trips was 1.9 seconds.

The reuse of saved SSL session state has some performance costs. When SSL session state is stored to disk, reuse of the saved state normally requires locating and retrieving the relevant state from disk. This cost can be reduced when using HTTP persistent connections. Oracle HTTP Server uses persistent HTTP connections by default, assuming they are supported on the client side. In HTTP over SSL as implemented by Oracle HTTP Server, SSL session state is kept in memory while the associated HTTP connection is persisted, a process which essentially eliminates the overhead of SSL session reuse (conceptually, the SSL connection is kept open along with the HTTP connection).

5.3.1.2 SSL Application Level Data Encryption

In most applications using SSL, the data encryption cost is small compared with the cost of SSL session management. Encryption costs can be significant where the volume of encrypted data is large, and in such cases the data encryption algorithm and key size chosen for an SSL session can be significant.

In general there is a trade-off between security level and performance. For example, on a modern processor, RSA estimates its RC4 cipher to take in the vicinity of 8-16 machine operations per output byte. Standard DES encryption will incur roughly 8 times the overhead of RC4, and triple DES will take about 25 times the overhead of DES. However, when using triple DES, the encryption costs will not be noticeable in most applications. Oracle HTTP Server supports these three cipher suites, and other cipher suites as well.

Oracle HTTP Server negotiates a cipher suite with a client based on the `SSLCipherSuite` attribute specified in `httpd.conf`.

See Also: *Oracle HTTP Server Administrator's Guide* for information on using supported cipher suites

5.3.1.3 SSL Performance Recommendations

The following recommendations can assist you with determining performance requirements when working with Oracle HTTP Server and SSL.

1. The SSL handshake is an inherently expensive process in terms of both CPU usage and response time. Thus, use SSL only where needed. Determine the parts of the application that require the security, and the level of security required, and protect only those parts at the requisite security level. Attempt to minimize the need for the SSL handshake by using SSL sparingly, and by reusing session state as much as possible. For example, if a page contains a small amount of sensitive data and a number of non-sensitive graphic images, use SSL to transfer the sensitive data only, use normal HTTP to transfer the images. If the application requires server authentication only, do not use client authentication. If the performance goals of an application cannot be met by this method alone, additional hardware may be required.

2. Design the application to use SSL efficiently. Group secure operations together to take advantage of SSL session reuse and SSL connection reuse.
3. Use persistent connections, if possible, to minimize cost of SSL session reuse.
4. Tune the session cache timeout value (the `SSLSessionCacheTimeout` attribute in `httpd.conf`). A trade-off exists between the cost of maintaining an SSL session cache and the cost of establishing a new SSL session. As a rule, any secured business process, or conceptual grouping of SSL exchanges, should be completed without incurring session creation more than once. The default value for the `SSLSessionCacheTimeout` attribute is 300 seconds. It is a good idea to test an application's usability to help tune this setting.
5. If large volumes of data are being protected through SSL, pay close attention to the cipher suite being used. The `SSLCipherSuite` directive specified in `httpd.conf` controls the cipher suite. If lower levels of security are acceptable, use a less-secure protocol using a smaller key size (this may improve performance significantly). Finally, test the application using each available cipher suite for the desired security level to find the most performant suite.
6. Having taken the preceding considerations into account, if SSL remains a bottleneck to the performance and scalability of your application, consider deploying multiple Oracle HTTP Server instances over a hardware cluster or consider the use of SSL accelerator cards.

5.3.2 Oracle HTTP Server Port Tunneling Performance Issues

When OracleAS Port Tunneling is configured, every request processed passes through the OracleAS Port Tunneling infrastructure. Thus, using OracleAS Port Tunneling can have an impact on the overall Oracle HTTP Server request handling performance and scalability.

With the exception of the number of OracleAS Port Tunneling processes to run, the performance of OracleAS Port Tunneling is self tuning. The only performance control available is to start more OracleAS Port Tunneling processes, this increases the number of available connections and hence the scalability of the system.

The number of OracleAS Port Tunneling processes is based on the degree of availability required, and the number of anticipated connections. This number cannot be automatically determined because for each additional process a new port must be opened through the firewall between the DMZ and the intranet. You cannot start more processes than you have open ports, and you do not want less processes than open ports, since in this case ports would not have any process bound to them.

To measure the OracleAS Port Tunneling performance, determine the request time for servlet requests that pass through the OracleAS Port Tunneling infrastructure. The response time of an Oracle Application Server instance running with OracleAS Port Tunneling should be compared with a system without OracleAS Port Tunneling to determine whether your performance requirements can be met using OracleAS Port Tunneling.

See Also: *Oracle HTTP Server Administrator's Guide* for information on configuring OracleAS Port Tunneling

5.4 Oracle HTTP Server Performance Tips

The following tips can enable you to avoid or debug potential Oracle HTTP Server (OHS) performance problems:

- [Analyze Static Versus Dynamic Requests](#)
- [Analyze Time Differences Between Oracle HTTP Server and OC4J Servers](#)
- [Beware of a Single Data Point Yielding Misleading Results](#)

5.4.1 Analyze Static Versus Dynamic Requests

It is important to understand where your server is spending resources so you can focus your tuning efforts in the areas where the most stands to be gained. In configuring your system, it can be useful to know what percentage of the incoming requests are static and what percentage are dynamic.

Generally, you want to concentrate your tuning effort on dynamic pages because dynamic pages can be costly to generate. Also, by monitoring and tuning your application, you may find that much of the dynamically generated content, such as catalog data, can be cached, sparing significant resource usage.

5.4.2 Analyze Time Differences Between Oracle HTTP Server and OC4J Servers

In some cases, you may notice a high discrepancy between the average time to process a request in Oracle Containers for J2EE (OC4J) and the average response time experienced by the user. If the time is not being spent actually doing the work in OC4J, then it is probably being spent in transport.

If you notice a large discrepancy between the request processing time in OC4J and the average response time, consider tuning the Oracle HTTP Server directives shown in the section, "[Configuring Oracle HTTP Server Directives](#)" on page 5-1.

5.4.3 Beware of a Single Data Point Yielding Misleading Results

You can get unrepresentative results when data outliers appear. This can sometimes occur at start-up. To simulate a simple example, assume that you ran a PL/SQL "Hello, World" application for about 30 seconds. Examining the results, you can see that the work was all done in `mod_plsql.c`:

```
/ohs_server/ohs_module/mod_plsql.c
handle.maxTime:      859330
handle.minTime:      17099
handle.avg:          19531
handle.active:        0
handle.time:         24023499
handle.completed:    1230
```

Note that `handle.maxTime` is much higher than `handle.avg` for this module. This is probably because when the first request is received, a database connection must be opened. Later requests can make use of the established connection. In this case, to obtain a better estimate of the average service time for a PL/SQL module, that does not include the database connection open time which causes the `handle.maxTime` to be very large, recalculate the average as in the following:

```
(time - maxTime)/(completed -1)
```

For example, in this case this would be:

```
(24023499 - 859330)/(1230 -1) = 18847.98
```

Monitoring Using Built-in Performance Tools

This appendix includes the following sections:

- [Summary of Oracle Application Server Built-in Performance Metrics](#)
- [Viewing Performance Metrics Using AggreSpy](#)
- [Viewing Performance Metrics Using dmstool](#)
- [Viewing Performance Metrics Using AggreSpy \(for Standalone OC4J\)](#)
- [Using Built-in Performance Metrics on Windows Systems](#)

A.1 Summary of Oracle Application Server Built-in Performance Metrics

You can monitor performance using the Application Server Control Console **Performance** secondary tab, using the System MBean Browser from the JMX area of the **Administration** secondary tab, or by viewing the Oracle Application Server built-in performance metrics.

This appendix describes how to view the built-in performance metrics using the Oracle Application Server AggreSpy servlet or using the dmstool command.

[Table A-1](#) summarizes the built-in tools that allow you to view performance metrics.

Table A-1 Oracle Application Server Built-in Monitoring Commands

Command	Description
AggreSpy	AggreSpy is a pre-packaged servlet that reports performance metrics for an Oracle Application Server instance. You can only run AggreSpy when the home OC4J instance is running. By default the OC4J instance named home supports AggreSpy. Note: in some cases the home instance needs to be started to use AggreSpy.
dmstool	Allows you to monitor a specific performance metric, a set of performance metrics, or all performance metrics. Options allow you to specify a reporting interval to report the requested metrics. This command also allows you to show a text report listing all the built-in performance metrics available on the site. The dmstool command is located in the directory \$ORACLE_HOME/bin on UNIX systems and in %ORACLE_HOME%\bin on Windows systems.

See Also: [Appendix C, "Performance Metrics"](#)

A.2 Viewing Performance Metrics Using AggreSpy

The AggreSpy Servlet displays metrics for Oracle Application Server processes, including: Oracle HTTP Server, OC4J, Oracle Process Manager and Notification Server, and other Oracle Application Server component processes.

This section covers the following topics:

- [Using the AggreSpy Display](#)
- [AggreSpy URL With a Proxy Server](#)
- [AggreSpy URL and Access Control](#)
- [AggreSpy Limitation When Using Load Balancing With Multiple Instances](#)

A.2.1 Using the AggreSpy Display

AggreSpy organizes metrics into two areas: DMS Spies and Metric Tables.

- DMS Spies show the available metrics by parent process type and parent process number. By selecting individual DMS Spies, you can view, in text form, all metrics collected for the associated process.
- Metric Tables show the available metrics by metric table type and when multiple OC4Js are running include OC4J metrics from multiple OC4J instances. By selecting individual metric tables you can view, in table form, all metrics of a specified type. For example, metric tables allow you to view the metrics associated with OC4J Servlets, Oracle HTTP Server Modules, and Oracle Process Manager and Notification Server processes.

Note: To view DMS metrics using AggreSpy, you may need to configure your browser to disable the use of a proxy for the localhost, if your system is configured to use proxies. By default Oracle Application Server only allows access for the localhost. See ["AggreSpy URL With a Proxy Server"](#) on page A-4 for details.

DMS metric tables are identified by a name, such as `ohs_server` for the Oracle HTTP Server metrics. In AggreSpy, the term *metric table* refers to the built-in performance metric table names.

You can access performance metrics using AggreSpy from the following URL:

`http://host:port/dms0/AggreSpy`

where:

host is the Oracle HTTP Server host, for example, `tv.us.oracle.com`.

port is the Oracle HTTP Server listener port, for example `7777`.

Note: You can only run AggreSpy when the home OC4J instance is running. By default, the OC4J instance named `home` supports AggreSpy. Using an OracleAS Infrastructure, the home instance needs to be started to use AggreSpy, since by default the home instance is installed with OracleAS Infrastructure, but it is not started.

[Figure A-1](#) shows a sample AggreSpy display. The display shows two frames, one containing a list of DMS Spies and DMS Metric Tables, and one showing selected values for the DMS Spies or the Metric Tables.

AggreSpy provides navigation and display options, including:

- Access DMS Spies and Metric Tables using the links in the left frame.
- Sort rows in metric tables by clicking on the column headings.
- Display a table containing the descriptions of a Metric Table's metrics by clicking the Metric Definitions link shown on each metric table.

You need to refresh your browser to display built-in metric data after you start AggreSpy. When you first use AggreSpy many of the fields, and the complete list of DMS Spies may not contain all of the current Metric Tables. If you wait a short time, and then refresh the display, the data is available and AggreSpy shows the complete list of Metric Tables.

Note: The OC4J home instance must be running to use AggreSpy. When the home instance is down, requests to AggreSpy, `http://<host>:<port>:/dms0/AggreSpy`, report an HTTP 500 Internal Server error.

In the J2EE install, the home instance starts up with the command, `opmnctl startall`, or by clicking **Start** using Application Server Control Console.

Figure A–1 AggreSpy Performance Metric Display

DMS Spies

[All DMS Spies](#)

HTTP_Server:OHS:2992:6100 | [Text](#)

home:OC4J:12501:6100 | [Text](#)

opmn:2152:6100 | [Text](#)

Metric Tables

[All Metric Tables](#)

[JDBC_ConnectionSource](#)

[JMSSConnectionStats](#)

[JMSSConsumerStats](#)

[JMSSDestinationStats](#)

[JMSSPersistenceStats](#)

[JMSSSessionStats](#)

[JMSSStats](#)

[JMSSStoreStats](#)

[JTAResource](#)

[JVM](#)

[agg_oc4j_application](#)

[agg_oc4j_application_no_rate](#)

[agg_oc4j_ejb](#)

[agg_oc4j_ejb_module](#)

[agg_oc4j_ejb_module_no_rate](#)

[agg_oc4j_ejb_no_rate](#)

[agg_oc4j_instance](#)

[agg_oc4j_instance_no_rate](#)

[agg_oc4j_jca_factory](#)

[agg_oc4j_jca_factory_no_rate](#)

[agg_oc4j_jca_pool](#)

[agg_oc4j_jca_pool_no_rate](#)

[agg_oc4j_jdbc_pool](#)

[agg_oc4j_jdbc_pool_no_rate](#)

Table of Contents

127.0.0.1:7201/dmsoc4j/AggreSpy: OC4J:12501

- [Spies](#)
- [Tables](#)

Spies

Process	Format	SpvType	Host	Port	Path	uid	instance
opmn:2152:6100	Text	opmnlocal	tvanraal-pc	6100	/connect		
home:OC4J:12501:6100	Text	oc4j	127.0.0.1	7201	/dmsoc4j/Spy	814550108	10gR3.tvanraal-pc.us.oracle.co
HTTP_Server:OHS:2992:6100	Text	ohs	127.0.0.1	7201	/dms0/Spy	814550107	10gR3.tvanraal-pc.us.oracle.co

[Top](#)

Tables

Name	numRows	numColumns
JDBC_ConnectionSource	1	23
JMSSConnectionStats	3	7
JMSSConsumerStats	2	18
JMSSDestinationStats	9	81
JMSSPersistenceStats	5	56
JMSSSessionStats	2	8
JMSSStats	1	126
JMSSStoreStats	8	70
JTAResource	1	77
JVM	1	20
agg_oc4j_application	0	12
agg_oc4j_application_no_rate	0	8
agg_oc4j_ejb	0	10

A.2.2 AggreSpy URL With a Proxy Server

If your browser is configured to use a proxy server, then to access AggreSpy on the localhost, you need to configure the browser to disable the use of proxies for the localhost. The exact steps required to disable the use of a proxy server for the localhost depends on the browser you use.

A.2.3 AggreSpy URL and Access Control

By default, the dms0/AggreSpy URL is redirected and the redirect location is protected, allowing only the localhost (127.0.0.1) to access the AggreSpy Servlet.

To view metrics from a system other than the localhost you need to change the DMS configuration for the system that is running the Oracle Application Server that you want to monitor by modifying the file

\$ORACLE_HOME/Apache/Apache/conf/dms.conf on UNIX, or
%ORACLE_HOME%\Apache\Apache\conf\dms.conf on Windows systems.

Example A–1 shows a sample default configuration from dms.conf. This configuration limits AggreSpy to access metrics on the localhost (127.0.0.1). The port shown, 7200, may differ on your installation.

Example A–1 Sample dms.conf File for localhost Access for DMS Metrics

```
# proxy to DMS AggreSpy
Redirect /dms0/AggreSpy http://localhost:7200/dmsoc4j/AggreSpy
#DMS VirtualHost for access and logging control
Listen 127.0.0.1:7200
OpmnHostPort http://127.0.0.1:7200
<VirtualHost 127.0.0.1:7200>
    ServerName 127.0.0.1
```

By changing the `dms.conf` configuration to specify the host that provides, or serves DMS metrics, you can allow users on systems other than the localhost to access the DMS metrics from the location `http://host:port/dms0/AggreSpy`.

Caution: Modifying `dms.conf` has security implications. Only modify this file if you understand the security implications for your site. By exposing metrics to systems other than the localhost, you allow other sites to potentially view critical Oracle Application Server internal status and runtime information.

To view metrics from a system other than the localhost (127.0.0.1), do the following:

1. Modify `dms.conf` by changing the entries with the value for localhost "127.0.0.1" shown in [Example A–1](#) to the name of the server providing the metrics (obtain the server name from the `ServerName` directive in the `httpd.conf` file, for example `tv.us.oracle.com`).
2. [Example A–2](#) shows a sample updated `dms.conf` that allows access from a system other than the localhost (127.0.0.1).

Example A–2 Sample dms.conf File for Remote Host Access for DMS Metrics

```
# proxy to DMS AggreSpy
Redirect /dms0/AggreSpy http://tv.us.oracle.com:7200/dmsoc4j/AggreSpy
#DMS VirtualHost for access and logging control
Listen tv.us.oracle.com:7200
OpmnHostPort http://tv.us.oracle.com:7200
<VirtualHost tv.us.oracle.com:7200>
    ServerName tv.us.oracle.com
```

3. Restart, or stop and start the Oracle HTTP Server using Application Server Control Console or using the `opmnctl` command. For example,

```
%opmnctl restartproc process-type=HTTP_Server
```

or

```
%opmnctl stopproc process-type=HTTP_Server
```

```
%opmnctl startproc process-type=HTTP_Server
```

See Also: *Oracle Application Server Security Guide* for information on Oracle HTTP Server access control

A.2.4 AggreSpy Limitation When Using Load Balancing With Multiple Instances

AggreSpy does not work as expected when using Oracle Application Server with multiple instances. When the Oracle HTTP Server `mod_oc4j` component load balances OC4J requests across multiple instances, AggreSpy may report results for systems that are not the localhost (127.0.0.1).

Note: In this case it is recommended that you use `dmstool` instead of AggreSpy.

A.3 Viewing Performance Metrics Using dmstool

The `dmstool` command allows you to view a specific performance metric, a set of performance metrics, or all performance metrics for an Oracle Application Server instance. The `dmstool` command also supports an option that allows you to set a reporting interval, specified in seconds, to report updated metrics every *t* seconds.

For example, you can monitor the performance of a specific servlet, JSP, EJB, EJB method, or database connection and you can request periodic snapshots of metrics specific to these components.

The format for using `dmstool` to display built-in performance metrics is:

```
% dmstool [options] metric metric ...
```

or

```
% dmstool [options] -list
```

or

```
% dmstool [options] -dump
```

[Table A-2](#) lists the `dmstool` command-line *options*. Following [Table A-2](#) this section presents examples that show sample usage with specific performance metrics. The `dmstool` command is located in the `$ORACLE_HOME/bin` directory on UNIX or in `%ORACLE_HOME%\bin` directory on Windows.

Note: You can use `dmstool` in scripts or in combination with other monitoring tools to gather performance data, to check application performance, or to build tools that modify your system based on the values of performance metrics.

See Also:

["Using dmstool to List the Names of All Metrics"](#) on page A-8

[Appendix C, "Performance Metrics"](#) for a list and description of the DMS metrics

A.3.1 Access Control for dmstool

By default, `dmstool` shows metrics only when it is run from the localhost (127.0.0.1). If you want to view metrics from an Oracle Application Server running on a remote host, then you need to use `dmstool` with the `-a` option, on the local host, and update the `dms.conf` file of the remote Oracle Application Server instance in the `$ORACLE_`

HOME/Apache/Apache/conf/ directory on UNIX or %ORACLE_HOME%\Apache\Apache\conf\ directory on Windows.

The configuration changes required to control the access to metrics using dmstool are the same as those for accessing dms0/AggreSpy.

See Also: ["AggreSpy URL and Access Control"](#) on page A-4

Table A-2 dmstool Command-line Options

Option	Description
<code>-a[address] opmn://host[:port]</code>	<p>By default, without the <code>-a</code> option, dmstool gets metrics from the Oracle Application Server instance with the same \$ORACLE_HOME as dmstool. When dmstool runs in the same \$ORACLE_HOME as the Oracle Process Manager and Notification Server, OPMN, the <code>-a</code> option is not required.</p> <p>You can specify <code>-a</code> with the <code>opmn://</code> prefix and the arguments shown to monitor the Oracle Application Server processes under OPMN control that produce DMS metrics (some OPMN controlled processes, for example Oracle Application Server Web Cache, do not expose DMS metrics).</p> <p>Where:</p> <p><code>host</code> is the domain name or IP address of the host on which the OPMN process is running.</p> <p><code>port</code> specifies the OPMN request port that supplies metrics. The request port is specified in \$ORACLE_HOME/opmn/conf/opmn.xml.</p> <p>For example, the following shows the specification in <code>opmn.xml</code> for a request port (<code>request="6003"</code>):</p> <pre><notification-server> <port local="6100" remote="6200" request="6003"/> . . </notification-server></pre> <p>Note, if you use dmstool <code>-a</code> to request metrics from a remote system, the system must be configured to provide metrics (by default you can access DMS metrics on the localhost).</p> <p>See Also: "AggreSpy URL and Access Control" on page A-4</p>
<code>-c[ount] num</code>	<p>Specifies the number of times to retrieve values when monitoring metrics. If not specified, dmstool continues retrieving metric values until the process is stopped.</p> <p>The <code>-count</code> option is not used with the <code>-list</code> option.</p>
<code>-dump [format=xml]</code>	<p>Using dmstool with the <code>-dump</code> option reports all the available metrics on the standard output. Oracle recommends that you run with the <code>-dump</code> option periodically, such as every 15 to 20 minutes, to capture and save a record of performance data for your Oracle Application Server server.</p> <p>The <code>-dump</code> option also supports the <code>format=xml</code> query. Using this query at the end of the command line supplies the metric output in XML format.</p>
<code>-help</code>	List the dmstool command-line options.
<code>-i[nterval] secs</code>	<p>Specifies the number of seconds to wait between metric retrievals. The default is 5 seconds. The <code>interval</code> argument is not used with the <code>-list</code> option. The interval specified is approximate.</p> <p>Note: if the system load is high, the actual interval may vary from the interval specified using the <code>-interval</code> option.</p>

Table A–2 (Cont.) dmstool Command-line Options

Option	Description
<code>-l[ist] [-table]</code>	<p>Generates a list of all metrics available. Use the <code>-list</code> option with the <code>-table</code> option to display a list of all the metric table names.</p> <p>Note, including metric names on the command-line is not valid when using the <code>-list</code> option with <code>dmstool</code>.</p>
<code>-reset [-table metric_table]</code>	<p>Resets the specified metrics or with the <code>-table</code> option, all of the metrics contained in the specified metric table.</p> <p>Event and phaseEvent metrics are reset to 0 (as if they were never updated). State metrics are reset to the current value (as if they started with the current value).</p> <p>Note: The <code>reset</code> option may reset information that Application Server Control Console uses to compute and report values.</p>
<code>-table metric_table</code>	<p>Includes all the performance metrics for the specified metric table with the name, <code>metric_table</code>.</p> <p>See Appendix C, "Performance Metrics" or run <code>AggreSpy</code> for a list of metric table names.</p>

A.3.2 Using dmstool to List the Names of All Metrics

Every Oracle Application Server performance metric has a unique name. Using `dmstool` with the `-list` option produces a list of all metric names. The `-list` output contains the metric names that you can use with `dmstool` to request monitoring information for a specific metric or set of metrics.

Using the following command, `dmstool` displays a list of all metrics available on the server:

```
% dmstool -list
```

This command displays a list of the available metrics.

See Also: [Appendix C, "Performance Metrics"](#)

A.3.3 Using dmstool to Report Values for Specific Performance Metrics

To monitor a specific metric or set of metrics, use `dmstool` and include the metric name on the command-line. For example, to monitor the time the JVM has been running, perform the following steps:

1. Use `dmstool` with the `-list` option to find the name of the metric that shows the JVM uptime:

```
% dmstool -list | grep JVM/upTime.value
/system1/OC4J:12502:6100/JVM/upTime.value
```

2. Use `dmstool` and supply the full metric name as an argument to show the metric value:

```
% dmstool /system1/OC4J:12502:6100/JVM/upTime.value
Wed Dec 21 15:37:08 PST 2005
/system1/OC4J:12502:6100/JVM/upTime.value 159312 msecs
```

Using `dmstool`, the default repeat interval is 5 seconds, so this command shows the updated metric every 5 seconds. Use the `-count` option to limit the number of times `dmstool` reports values.

For example:

```
% dmstool /system1/OC4J:12502:6100/JVM/upTime.value -count 2
Wed Dec 21 15:39:38 PST 2005
/system1/OC4J:12502:6100/JVM/upTime.value    310031    msecs
Wed Dec 21 15:39:43 PST 2005
/system1/OC4J:12502:6100/JVM/upTime.value    314516    msecs
```

Note: In some cases, the full path of a metric name may contain a space. If the path contains a space, the space must be quoted on the `dmstool` command line, so that the shell sends the metric name to `dmstool` as a single argument.

A.3.4 Using dmstool With the Interval and Count Options

To monitor the requests completed for an application over an interval of one minute, use the following `dmstool` command, supplying metric names on the command-line:

```
% dmstool -i 60 -c 120 \
/system1/OC4J:3301:6003/oc4j/default/WEBs/processRequest.completed
```

This command reports 120 sets of output for the metric listed on the command line, while collecting data at intervals of 60 seconds:

```
Tue Oct 12 14:43:43 PDT 2004
/system1/OC4J:3301:6003/oc4j/default/WEBs/processRequest.completed    8576 ops

Tue Oct 12 14:44:43 PDT 2004
/system1/OC4J:3301:6003/oc4j/default/WEBs/processRequest.completed    8581 ops

Tue Oct 12 14:45:43 PDT 2004
/system1/OC4J:3301:6003/oc4j/default/WEBs/processRequest.completed    8588 ops
.
.
.
```

A.3.5 Using dmstool to Report All Metrics with Metric Values

Using `dmstool` with the `-dump` option displays all the metrics from an Oracle Application Server instance to the standard output.

The following command displays all available metrics:

```
% dmstool -dump
```

Oracle recommends that you run `dmstool` with the `-dump` option periodically, such as every 15 to 20 minutes, to capture and save a record of performance data. If you save performance data over time, this data can assist you if you need to analyze system behavior to improve performance or when problems occur.

A.3.6 Using dmstool to Report All Metrics with Metric Values in XML Format

When you need to process metric data, use the `format=xml` query on the `dmstool` command line to report all metric values in XML format.

The following command displays all available metrics using XML format:

```
% dmstool -dump format=xml
```

A.3.7 Using dmstool to Reset Metric Values

When you want to reset metric values, use the `reset` option on the `dmstool` command line to reset values for a set of metrics, or for all metrics in a specified metric table.

Using the `reset` option, Event and phaseEvent metrics are reset to 0, as if they were never updated, and State metrics are reset to the current value (as if they started with the current value).

The following command resets the specified metric:

```
% dmstool -reset /system1/OC4J:3000:6004/JVM/upTime.value
```

The following command resets the specified metric table:

```
% dmstool -reset /system1/OC4J:3000:6004/JVM/upTime.value
```

Note: The `reset` option may reset information that Application Server Control Console uses to compute and report values.

A.3.8 Using dmstool to View Metrics on a Remote Oracle Application Server System

Using `dmstool` with the `-a` option reports the metrics from a remote Oracle Application Server instance.

Note: By default the Oracle Application Server only allows `dmstool` to access metrics from the localhost. You need to modify `dms.conf` to support access from systems other than the localhost. See "[AggreSpy URL and Access Control](#)" on page A-4 for information on DMS access control.

The following command displays all available metrics and metric values on the Oracle Application Server Instance, as specified with the `-a` option:

```
% dmstool -a opmn://system1:6003 -list
```

Using the `dmstool -a` option, supply an argument with the prefix `opmn://` and include the host name where you want to obtain metrics, and the OPMN request port number. The port specifies the OPMN request port that supplies metrics for Oracle Application Server which is specified the request attribute under the `<notification-server>` element in `$ORACLE_HOME/opmn/conf/opmn.xml` on UNIX and `%ORACLE_HOME%\opmn\conf\opmn.xml` on Windows.

See Also: "[AggreSpy URL and Access Control](#)" on page A-4

A.4 Viewing Performance Metrics Using AggreSpy (for Standalone OC4J)

When you are using OC4J in standalone mode, without the Oracle Application Server, the AggreSpy Servlet allows you to access OC4J metrics.

When running OC4J standalone, access performance metrics using AggreSpy from the following URL:

```
http://myhost:myport/dms0/AggreSpy
```

Note: You can only run AggreSpy when OC4J is configured to support it, and OC4J is running. By default, OC4J supports AggreSpy.

Table A-3 covers the dmstool option that only applies to OC4J standalone mode. In addition, the options shown in Table A-2 also apply to dmstool (except the -a option with the opmn:// prefix.

Table A-3 *dmstool Command-line Options (for Standalone OC4J only)*

Option	Description
-a[address] host[:port][path],...	For a standalone OC4J system, use the -a option. This specifies the http:// protocol, where: host is the domain name or IP address of the host on which the Oracle HTTP Server is running and port specifies the associated port.

A.5 Using Built-in Performance Metrics on Windows Systems

Using Oracle Application Server on Windows systems, statistics collection needs to be enabled to view certain DMS metrics. If some DMS metrics report the value "0" for values that you expect to be other than 0, then statistics collection may be disabled on your system. To enable statistics collection on Windows systems where statistics collection is disabled, set the value of the following registry entry to 0.

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\PerfProc\Performance\Disable  
Performance Counters
```

Note: Incorrectly editing the registry may severely damage your system. At the very least, you should back up any valued data on the computer before making changes to the registry.

Instrumenting Applications With DMS

The Oracle Dynamic Monitoring Service (DMS) enables application developers, support analysts, system administrators, and others to measure application specific performance information. This chapter describes DMS and shows a sample application that demonstrates how to instrument Oracle Application Server Java applications using DMS.

Note: Oracle Application Server provides a number of built-in metrics. Using DMS to instrument applications adds new metrics to the set of built-in metrics.

This chapter includes the following sections:

- [Introducing DMS Performance Metrics](#)
- [Adding DMS Instrumentation To Java Applications](#)
- [Validating and Testing Applications Using DMS Metrics](#)
- [Understanding DMS Security Considerations](#)
- [Conditional Instrumentation Using DMS Sensor Weight](#)
- [Dumping DMS Metrics To Files](#)
- [Resetting and Destroying Sensors](#)
- [DMS Coding Recommendations](#)
- [Using A High Resolution Clock To Increase DMS Precision](#)
- [Rolling Up DMS Data for Descendent Nouns](#)

See Also: [Appendix C, "Performance Metrics"](#)

B.1 Introducing DMS Performance Metrics

The Dynamic Monitoring Service (DMS) API allows you to add performance instrumentation to Oracle Application Server applications. At runtime OC4J collects performance information, called DMS metrics, that developers, system administrators, and support analysts use to help analyze system performance or monitor system status.

This section includes the following:

- [Instrumenting Applications With DMS Metrics](#)
- [Monitoring DMS Metrics](#)
- [Understanding DMS Terminology \(Nouns and Sensors\)](#)
- [DMS Naming Conventions](#)

Note: Oracle Application Server components, including OC4J, provide a number of predefined metrics. For a listing of the predefined metrics see [Appendix C, "Performance Metrics"](#).

B.1.1 Instrumenting Applications With DMS Metrics

DMS **Instrumentation** refers to the process you use when you insert DMS calls into application code. By using the DMS API you can enable your application to measure, collect, and save performance information.

To create DMS metrics you add DMS API 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 metrics in memory and allows you to save or view the metrics.

Oracle Application Server includes built-in DMS metrics. By adding DMS calls to your applications you can expand the set of built-in metrics. When you instrument your applications with DMS calls, you use the same API that the built-in metrics use. In addition, to save and display your metrics, you use the same monitoring tools that you use with built-in metrics.

Tip: ["Adding DMS Instrumentation To Java Applications"](#) on page B-8

B.1.2 Monitoring DMS Metrics

Monitoring DMS metrics refers to the process of retrieving performance metrics. When an application runs, DMS stores metrics in memory and allows you to show metrics on the console or to view metrics using a web browser.

Oracle Application Server provides several runtime tools for viewing and saving DMS metrics, including `dmstool` and the `AggreSpy` Servlet.

[Example B-1](#) shows a set of metrics output using `dmstool`.

Example B-1 Set of Sample `dmsDemo` Metrics Using `dmstool`

```
/dmsDemo [type=n/a]
/dmsDemo/BasicBinomial [type=MathSeries]
computeSeries.active:      0      threads
computeSeries.avg:    21.1818181818183      msecs
computeSeries.completed:  11      ops
computeSeries.maxActive:  1      threads
```



```

computeSeries.maxTime:      93      msecs
computeSeries.minTime:      0      msecs
computeSeries.time: 233      msecs
lastComputed.value: 184756
loops.count: 604      ops

```

See Also: [Appendix A, "Monitoring Using Built-in Performance Tools"](#)

B.1.3 Understanding DMS Terminology (Nouns and Sensors)

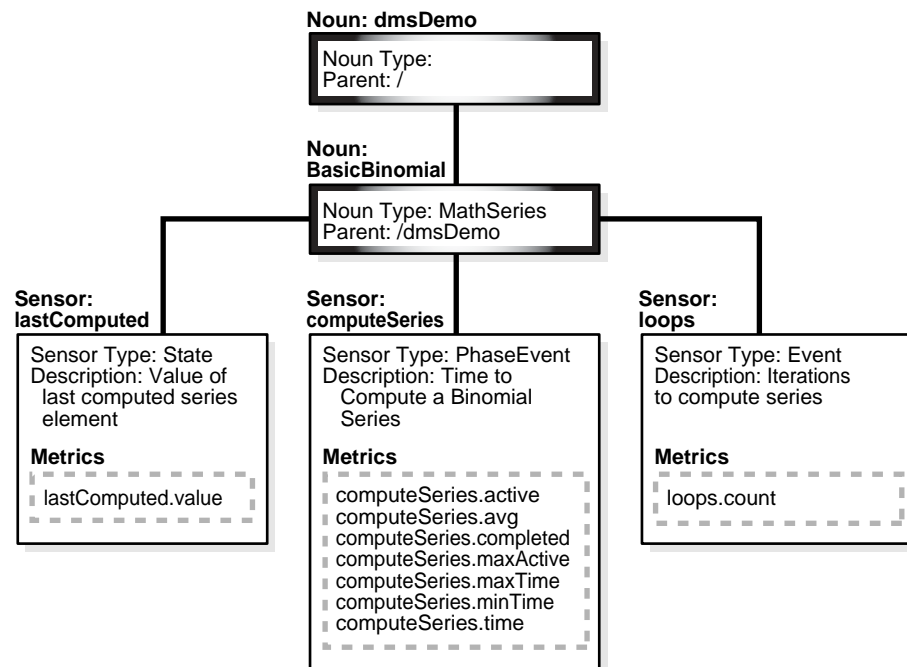
This section introduces the terminology you need to understand to use DMS.

[Figure B-1](#) illustrates the organization of a set of DMS metrics corresponding to the metrics in the demo application described in this chapter and the metrics shown in [Example B-1](#).

This section covers the following topics:

- [DMS Metrics](#)
- [DMS Sensors](#)
- [DMS Nouns](#)
- [DMS Rollup Nouns](#)
- [DMS Object Relationships](#)

Figure B-1 Organization of Sample Metrics From *dmsDemo* Application



B.1.3.1 DMS Metrics

DMS Metrics track performance information that developers, system administrators, and support analysts use to help analyze system performance or monitor system status.

B.1.3.2 DMS Sensors

DMS Sensors measure performance data and allow DMS to define and collect a set of metrics. Certain metrics are always included with a Sensor and other metrics are optionally included with a Sensor.

B.1.3.2.1 DMS PhaseEvent Sensors A **DMS PhaseEvent Sensor** measures the time spent in a specific section of code that has a beginning and an end. Use a PhaseEvent Sensor to track time in a method or in a block of code.

DMS can calculate optional metrics associated with a PhaseEvent, including: the average, maximum, and minimum time that is spent in the PhaseEvent Sensor.

Table B–1 describes metrics available with a PhaseEvent Sensor.

Table B–1 DMS PhaseEvent Sensor Metrics

Metric	Description
<i>sensor_name.time</i>	Specifies the total time spent in the phase <i>sensor_name</i> . Default metric: time is a default PhaseEvent Sensor metric.
<i>sensor_name.completed</i>	Specifies the number of times the phase <i>sensor_name</i> , has completed since the process was started. Optional metric
<i>sensor_name.minTime</i>	Specifies the minimum time spent in the phase <i>sensor_name</i> , for all the times the phase completed. Optional metric
<i>sensor_name.maxTime</i>	Specifies the maximum time spent in the phase <i>sensor_name</i> , over all the times the <i>sensor_name</i> phase completed. Optional metric
<i>sensor_name.avg</i>	Specifies the average time spent in the phase <i>sensor_name</i> , computed as the (time total)/(number of times the phase completed). Optional metric
<i>sensor_name.active</i>	Specifies the number of threads in the phase <i>sensor_name</i> , at the time the DMS statistics are gathered (the value may change over time). Optional metric
<i>sensor_name.maxActive</i>	Specifies the maximum number of concurrent threads in the phase <i>sensor_name</i> , since the process started. Optional metric

B.1.3.2.2 DMS Event Sensors A **DMS Event Sensor** is a Sensor that counts system events. Use a DMS Event Sensor to track system events that have a short duration, or where the duration of the event is not of interest but the occurrence of the event is of interest.

Table B–2 describes the metric that is associated with an Event Sensor.

Table B–2 DMS Event Sensor Metrics

Metric	Description
<i>sensor_name.count</i>	Specifies the number of times the event has occurred since the process started, where <i>sensor_name</i> is the name of the Event Sensor as specified in the DMS instrumentation API. Default: count is the default metric for an Event Sensor. No other metrics are available for an Event Sensor.

B.1.3.2.3 DMS State Sensors A DMS State Sensor is a Sensor to which you assign a precomputed value. State Sensors track the value of Java primitives or the content of a Java Object. The supported types include integer, double, long, and object. Use a State Sensor when you want to track system status information or when you need a performance metric that is not associated with an event. For example, use State Sensors to represent queue lengths, pool sizes, buffer sizes, or host names.

Table B–3 describes the State Sensor metrics. State Sensors support a default metric value, as well as optional metrics. The optional *minValue* and *maxValue* metrics only apply for State Sensors if the State Sensor represents a numeric Java primitive (of type integer, double, or long).

Table B–3 DMS State Sensor Metrics

Metric	Description
<i>sensor_name.value</i>	Specifies the metric value for <i>sensor_name</i> , using the type assigned when <i>sensor_name</i> is created. Default: value is the default State metric.
<i>sensor_name.count</i>	Specifies the number of times <i>sensor_name</i> is updated. Optional metric
<i>sensor_name.minValue</i>	Specifies the minimum value for <i>sensor_name</i> since startup. Optional metric
<i>sensor_name.maxValue</i>	Specifies the maximum value this <i>sensor_name</i> since startup. Optional metric

B.1.3.3 DMS Nouns

DMS Nouns (Nouns) organize performance data. Each Sensor, with its associated metrics is organized in a hierarchy according to Nouns. Nouns allow you to organize DMS metrics in a manner comparable to a directory structure in a file system. For example, Nouns can represent classes, methods, objects, queues, connections, applications, databases, or other objects that you want to measure.

A Noun type is a name that reflects the set of metrics being collected. For example, in the built-in metrics the Noun type *oc4j_servlet* represents the metrics collected for each servlet in each Web module within each J2EE application. And the Noun type *JVM* represents the set of metrics for each Java process (OC4J) currently running in the site.

Note: In [Appendix C, "Performance Metrics"](#), the Noun type is called the metric table name.

The Noun naming scheme uses a '/' as the root of the hierarchy, with each Noun acting as a container under the root, or under its parent Noun.

See Also: [Appendix C, "Performance Metrics"](#)

B.1.3.4 DMS Rollup Nouns

DMS Rollup Nouns are nouns that DMS generates when you include instrumentation to request a set of aggregate nouns. The rollup noun contains metrics from a set of Sensors in the descendent nouns of a specified noun type. A rollup noun also contains summary information.

See Also: ["Rolling Up DMS Data for Descendent Nouns"](#) on page B-21

B.1.3.5 DMS Object Relationships

This section describes the object relationships and attributes for DMS metrics, Sensors, and Nouns.

[Table B-4](#) describes the relationships between DMS objects. [Figure B-1](#) illustrates the relationships shown in [Table B-4](#) using a sample set of metrics.

Table B-4 DMS Object Relationships and Attributes

Object	Contains	Attributes
Noun	Sensors or other Nouns	Name, Noun Type, Parent
Sensor	Metrics	Name, Description, Sensor Type, Parent There are three Sensor Types: PhaseEvent, Event, and State.
Metric	Value	Name, Units designation

B.1.4 DMS Naming Conventions

Certain guidelines apply for defining DMS names. By following these guidelines, people viewing DMS metric reports can easily understand metrics across applications and across Oracle Application Server components.

Note: View the naming conventions as guidelines; for each convention there may be an exception. Try to be as clear as possible, if there is a conflict, you may need to make an exception.

This section covers the following topics:

- [General DMS Naming](#)
- [General DMS Naming Conventions and Character Sets](#)
- [Noun and Noun Type Naming Conventions](#)
- [Sensor Naming Conventions](#)

B.1.4.1 General DMS Naming

DMS metric names consist of a Sensor name plus the "." character plus the metric. For example, the names: `computeSeries.time`, `loops.count`, and `lastComputed.value` are valid DMS metric names.

A Sensor name is a simple string, not including the "." or the derivation. For example `computeSeries`, `loops`, and `lastComputed` are Sensor names. A Sensor full name consists of the Sensor name, preceded by the name of its associated Noun, and a delimiter. For example, `/dmsDemo/BasicBinomial/computeSeries`,

/dmsDemo/BasicBinomial/loops, and
/dmsDemo/BasicBinomial/lastComputed.

A Noun name is a simple string, not including a delimiter. For example BasicBinomial is a Noun name. A Noun full name consists of the Noun name, preceded by the full name of its parent, and a delimiter. For example /dmsDemo/BasicBinomial is a full Noun name.

B.1.4.2 General DMS Naming Conventions and Character Sets

DMS names should be as compact as possible. Whenever possible, when you define Noun and Sensor names, avoid special characters such as white space, slashes, periods, parenthesis, commas, and control characters.

Table B-5 shows DMS replacement for special characters in names.

Table B-5 DMS Naming Special Character Replacement

Character	DMS Replacement Character
Space " " or Period "."	Underscore "_"
Control Character	Underscore "_"
"<"	"("
">"	")"
"&"	"^"
"" (double quote)	"" (backquote) That is, a backquote replaces a double quote.
" (single quote)	" (backquote). That is, a backquote replaces a single quote.

Note: Oracle Application Server includes a number of built-in metrics. The Oracle Application Server built-in metrics do not always follow the DMS naming conventions.

B.1.4.3 Noun and Noun Type Naming Conventions

A Noun name should be a name which identifies a specific entity of interest.

Noun types should have names which clearly reflect the set of metrics being collected. For example, Servlet is the type for a Noun under which the metrics that are specific to a given servlet fall.

Noun type names should start with a capitol letter to distinguish them from other DMS names. All Nouns of a given type should contain the same set of sensors.

B.1.4.4 Sensor Naming Conventions

The following list outlines DMS Sensor naming conventions.

1. Sensor names should be descriptive, but not redundant. Sensor names should not contain any part of the Noun name hierarchy, or type, as this is redundant.
2. Sensor names should avoid containing the specification of the units for the individual metrics.

3. Where multiple words are required to describe a Sensor, the first word should start with a lowercase letter, and the following words should start with uppercase letters. For example `computeSeries`.
4. In general, using a "/" in a Sensor name should be avoided. However, there are cases where it makes sense to use a name that contains "/". If a "/" is used in a Noun or Sensor name, then when you use the Sensor in a string with DMS methods, you need to use an alternative delimiter, such as "," or "_", which does not appear anywhere in the path; this allows the "/" to be properly understood as part of the Noun or Sensor name rather than as a delimiter.

For example, a child Noun can have a name such as:

```
examples/jsp/num/numguess.jsp
```

and you can look this up using the string:

```
,oc4j,default,WEBs,defaultWebApp,JSPs,example/jsp/num/numguess.jsp,service
```

Where the delimiter is the "," character.

5. Event Sensor and PhaseEvent Sensor names should have the form *verbNoun* where *verb* and *Noun* are interpreted as defined by English grammar. For example, `activateInstance` and `runMethod`. When a PhaseEvent monitors a function, method, or code block, it should be named to reflect the task performed as clearly as possible.
6. The name of a State Sensor should be a Noun, possibly preceded by an adjective, which describes the semantics of the value which is tracked with this State. For example, `lastComputed`, `totalMemory`, `port`, `availableThreads`, `activeInstances`.
7. To avoid confusion, do not name Sensors with strings such as: ".time", ".value", or ".avg", that are the same as the default metrics or optional derivations for a Sensor, as shown in [Table B-1](#), [Table B-2](#), and [Table B-3](#).

B.2 Adding DMS Instrumentation To Java Applications

You can collect performance information in Java applications by adding DMS instrumentation to existing applications or by creating new applications that include DMS instrumentation.

The DMS samples shown in this chapter are supplied on the Oracle Technology Network Web site

<http://www.oracle.com/technology/tech/java/oc4j/demos/index.html>

The DMS demo.zip file includes a ready to deploy .ear file and source code with build instructions. The demo includes two servlets, `BasicBinomial.java` and `ImprovedBinomial.java`.

The `BasicBinomial` servlet shows how to use the DMS API to add DMS Sensors.

The `ImprovedBinomial` servlet expands on the `BasicBinomial` and illustrates measuring the improved code, as compared with the `BasicBinomial`. `ImprovedBinomial` servlet also shows how to add more costly metrics that gather more detailed information.

Refer to the sample code for full details on the examples in this chapter.

To use DMS instrumentation, add DMS calls by performing the following steps:

- [Including DMS Imports](#)

- [Organizing Performance Data](#)
- [Defining and Using Metrics for Timing](#)
- [Defining and Using Metrics for Counting](#)
- [Defining and Using Metrics for Recording Status Information \(State Sensors\)](#)

B.2.1 Including DMS Imports

To use DMS you need to add DMS imports. The following example shows the imports that the sample application `BasicBinomial.java` requires.

```
import oracle.dms.instrument.DMSConsole;
import oracle.dms.instrument.Event;
import oracle.dms.instrument.Noun;
import oracle.dms.instrument.PhaseEvent;
import oracle.dms.instrument.State;
import oracle.dms.instrument.Sensor;
```

B.2.2 Organizing Performance Data

Define DMS Nouns to organize Sensors and their associated metrics. DMS Nouns organize Sensors in a tree hierarchy in a manner comparable to a directory structure in a file system, starting with a root at the top of the tree.

[Example B-2](#) shows a section of code using `Noun.create()` from the `BasicBinomial.java`.

In [Example B-2](#), `MathSeries` specifies the Noun type. The Noun type is a name that reflects the set of metrics being collected. For example, `MathSeries` represents the metrics collected for the sample application containing a Binomial series computation. `AggreSpy` displays Sensors using the same Noun type together.

It is good practice to only use Noun types for Nouns that directly contain Sensors. When a Noun contains only Nouns, as in the Noun `dmsDemo`, and does not directly contain Sensors, `AggreSpy` displays the Noun type as a metric table, with no metrics. [Example B-2](#) shows the `dmsDemo` Noun that includes a Noun, `BasicBinomial`, but no Sensors. When the Noun type is not included for such a Noun, `AggreSpy` does not display a metric table associated with the Noun.

Note: Start Noun type names with a capital letter to distinguish them from other DMS names.

Example B-2 Using Noun.create To Organize Sensors

```
private Noun binRoot;           // Container for Binomial series DMS metrics.
Noun base = Noun.create("/dmsDemo");
binRoot = Noun.create(base, "BasicBinomial", "MathSeries");
```

See Also: ["DMS Naming Conventions"](#) on page B-6

B.2.2.1 Choosing Noun Types

In general, nouns should not be of the same noun type as any of their ancestor or descendent nouns. Usually, this is easy to code, and provides a logical hierarchy for nouns of the same type at the same level. For example, in the `dmsDemo` application, there is a second servlet, `ImprovedBinomial`, and there is the `BasicBinomial` servlet. In this case, the instrumentation uses the noun of type `MathSeries` for both.

This noun is created under `/dmsDemo` in the same hierarchy level for both servlets. Adhering to this practice makes the generated metric tables easier to understand. It also prevents some minimal information loss in the reporting process.

B.2.3 Defining and Using Metrics for Timing

To create metrics that measure the duration of a segment of code, define and use a `PhaseEvent` Sensor using the following steps:

- [Defining PhaseEvent Sensors](#)
- [Using PhaseEvent Sensors](#)

B.2.3.1 Defining PhaseEvent Sensors

[Example B-3](#) shows the DMS calls that declare and create the `computeSeries` `PhaseEvent` Sensor. This code defines a DMS metric named `/dmsDemo/BasicBinomial/computeSeries.time`.

`PhaseEvent` Sensors support a set of optional metrics, along with the default metric `.time` (representing the time, as measured between the `PhaseEvent start()` and the `PhaseEvent stop()` calls). You can derive optional metrics with `PhaseEvent` Sensors individually or as a complete set. [Table B-1](#) shows the available metrics for a `PhaseEvent` Sensor. The `binComp.deriveMetric(Sensor.all)` call in [Example B-3](#) causes all the supported optional metrics to be computed and reported.

Note: Using the method `deriveMetric(Sensor.all)` is recommended for adding optional metrics. Using this method with `Sensor.all` adds all metrics; this is good practice since the list of optional metrics could change in a future Oracle Application Server release. In addition, the metrics are efficient to compute and are often useful in evaluating performance.

Example B-3 Defining PhaseEvent Sensors

```
private PhaseEvent binComp; // Time to compute Binomial series.
.
.
.
binComp = PhaseEvent.create(binRoot, "computeSeries",
                           "Time to compute a Binomial series");
binComp.deriveMetric(Sensor.all);
```

B.2.3.2 Using PhaseEvent Sensors

To use a `PhaseEvent` Sensor, an application calls the `start()` method to indicate the beginning of a phase and subsequently calls the `stop()` method to indicate the completion of the phase.

[Example B-4](#) shows a code segment from `BasicBinomial.java` that uses the `start()` and `stop()` methods for the `/dmsDemo/BasicBinomial/computeSeries.time` metric. The long value named `token` that is returned from the `PhaseEvent start()` method must be passed to the corresponding `PhaseEvent stop()` method. This value is a timestamp representing the start time. Passing this value to the `stop()` method allows DMS to compute the `PhaseEvent` duration.

Note: To assure that PhaseEvents are stopped, each PhaseEvent start() method, together with the code to be measured should be in a try block with the PhaseEvent stop() method in a corresponding finally block, as shown in [Example B-4](#).

Example B-4 Using start() and stop() With PhaseEvent Sensors

```
long token = 0; // DMS
try {
    token = binComp.start(); // DMS
    BigInteger bins[] = bin(length);
    out.println("<H2>Binomial series for " + length + "</H2>");
    for (int i = 0; i < length; i++)
        out.println("<br>" + bins[i]);
}
finally {
    binComp.stop(token); // DMS
    out.close();
}
```

[Example B-4](#) shows code instrumented such that each time a phase starts, it is stopped (since the stop method is placed in the finally clause). This prevents runaway Phase Sensors; however, this can result in the time required to throw an exception possibly contributing to phase statistics. To prevent exception handling from impacting a PhaseEvent, use the abort() method, as shown in [Example B-5](#).

[Example B-5](#) shows a code sample where a Phase that is not successfully stopped will be aborted. The abort call removes the statistics corresponding to the corresponding start, and these statistics do not contribute to metric calculations.

Example B-5 Using abort() with PhaseEvent Sensors

```
PhaseEvent pe = heavyPhase(param);
long token1 = 0;
long token2 = 0;
boolean stopped = false;
try {
    token1 = binComp.start();
    if (pe != null) token2 = pe.start();
    BigInteger bins[] = bin(length);
    out.println("<H2>ImprovedBinomial series for " + length + "</H2>");
    for (int i = 0; i < length; i++)
        out.println("<br>" + bins[i]);
    if (pe != null) pe.stop(token2);
    binComp.stop(token1);
    stopped = true;
}
finally {
    if (!stopped) {
        if (pe != null) pe.abort(token2);
        binComp.abort(token1);
    }
}
```

B.2.4 Defining and Using Metrics for Counting

To create metrics that count the occurrences of an event, define and use an Event Sensor as follows:

- [Defining Event Sensors](#)

- [Using Event Sensors](#)

B.2.4.1 Defining Event Sensors

[Example B–6](#) shows the DMS calls that define an Event Sensor. This code allocates a counter and defines a DMS metric named `/dmsDemo/BasicBinomial/loops.count`.

Example B–6 Defining Event Sensors

```
private Event binLoop;    // Loops needed for Binomial series.
.
.
.

binLoop = Event.create(binRoot, "loops", "Iterations to compute series");
```

B.2.4.2 Using Event Sensors

DMS increments a counter when an application calls the `occurred()` method for an Event Sensor. [Example B–7](#) shows the `occurred()` call for an Event Sensor that increments the `/dmsDemo/BasicBinomial/loops.count` metric.

Example B–7 Using `occurred()` With Event Sensors

```
binLoop.occurred();
```

B.2.5 Defining and Using Metrics for Recording Status Information (State Sensors)

DMS captures status information with State Sensors. State Sensors track the value of Java primitives or the content of a Java Object. The supported types include integer, double, long, and object, as specified in the third argument to the `create()` method. When a Java primitive State Sensor is updated with the wrong type, DMS attempts to convert the supplied value to the correct type. For Object type State Sensors, DMS stores a reference to the Object and by default and calls `toString()` on the object when the DMS value is sampled.

To create metrics that record status information, define and use a State Sensor as follows:

- [Defining State Sensors](#)
- [Using State Sensors](#)

B.2.5.1 Defining State Sensors

State Sensors support a default metric value, as well as optional metrics. You can define the `minValue` and `maxValue` optional metrics with State Sensors only if the State Sensor represents a numeric Java primitive (of type integer, double, or long). [Table B–3](#) shows the available metrics for a State Sensor. [Example B–3](#) shows how to enable optional metrics.

[Example B–8](#) shows the DMS calls that declare and create a State Sensor. This code defines a DMS metric named `/dmsDemo/BasicBinomial/lastComputed.value`.

Example B–8 Defining State Sensors

```
private State binLast;    // Value of the last computed element in series.
.
```

```

.
.
binLast = State.create(binRoot, "lastComputed", State.OBJECT, "",
    "Value of last computed series element");

```

When you define a State Sensor, use an empty string in the fourth argument to the `create()` method if no units are associated with the State Sensor, otherwise use a string listing the appropriate units (see [Example B-8](#)). State Sensors are created without an initial value. If you need to check whether a State Sensor has been initialized, use the `isInitialized()` method.

If you want your State Sensor to store the string value of an object, and not store a reference to the object, use the `setCopy()` method with the value `TRUE`. This tells the State Sensor to store the result of calling `toString()` on an object rather than using a reference to the object for the metric value.

B.2.5.2 Using State Sensors

When an application calls a State Sensor's `update()` method, DMS updates the value of the State Sensor. [Example B-9](#) shows the `update()` call for a State Sensor that updates the `/dmsDemo/BasicBinomial/lastComputed.value` metric.

Example B-9 Using update() With State Sensors

```
binLast.update(bins[k-1].toString());
```

B.3 Validating and Testing Applications Using DMS Metrics

You should test and verify the accuracy of the metrics that you add to Java applications.

This section includes the following:

- [Validating DMS Metrics](#)
- [Testing DMS Metrics For Efficiency](#)

B.3.1 Validating DMS Metrics

Use the `dmstool` and the other available DMS monitoring tools to verify and test new metrics.

Try to validate the following for new metrics:

- Do expected metrics appear in the display? Test this by examining the code to make sure that all the metric names added using DMS instrumentation appear in your display or saved set of metrics.
- 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? Usually, upper and lower bounds for metrics can be established. You then test that the reported values for metrics do not exceed the expected bounds.

For example, a "size of pool" metric should never report a negative value.

- Make sure that new metrics are needed. For example, if you add a `PhaseEvent` that always measures an event of very short duration, consider changing the metric to an `Event` metric, or remove the metric.

- Make sure that new metrics are accurate. For most applications using DMS metrics, accuracy is more important than the performance cost of adding the DMS instrumentation. New DMS metrics should provide reliable and useful information.

Testing for accuracy can be difficult; however, if an alternate means of measuring a particular metric is available then use it to verify metric values. For example, if you submit a known number of requests to a server and measure total time for the experiment, then you predict correct values for the relevant metrics and compare them with the actual monitored values. As another example, you can verify an Event Sensor count metric by examining records that you write to a log file or to the console.

Check for timing inaccuracies that may apply for the metrics. Timing inaccuracies may be caused when low-resolution clocks time metrics for an interval of short duration. For example on Windows systems, the default Java clock advances only once every 15 milliseconds. DMS metrics reported for brief events on these systems must be analyzed with care. Consider using the high resolution clock to address this issue.

See Also: ["Using A High Resolution Clock To Increase DMS Precision"](#) on page B-17

B.3.2 Testing DMS Metrics For Efficiency

The use of DMS metrics has some influence on application performance. When adding metrics, note the following:

- The processing required for computing and storing metrics can slow down the execution of an application. DMS is fast, but it does have some required overhead cost. In addition, DMS cannot prevent developers from using the DMS API inefficiently. Therefore, before adding DMS instrumentation, establish reasonable expectations. After completing the implementation, measure the actual costs and compare them to your expectations. Be prepared to make changes to the instrumentation to reduce overhead costs until the measurements agree with expectations.
- DMS provides the `DMSConsole.getSensorWeight()` method to help you control the use of metrics. The central setting is an advisory measurement level that DMS does not enforce. To control which metrics to include, at runtime, the code must test the value for `SensorWeight` to determine whether to make DMS calls.
- When integrating DMS instrumentation with an existing package or when implementing a new feature, you should consider insulating a previously working system. For example, you could include an option to enable and disable new DMS metrics.
- Worrying about performance too soon often leads to costly design and implementation errors. According to Donald Knuth, "Premature optimization is the root of all evil".
- You should run your performance tests with and without DMS enabled. If your tests show unacceptable results with DMS enabled, then you may want to re-design or re-implement metrics.

B.4 Understanding DMS Security Considerations

DMS metrics do not support user based access to DMS reports. When you define and use a DMS metric, the metric is available to any administrator that has access to DMS metrics. This means when you add DMS metrics, it is good practice to avoid placing customer sensitive information in the metrics.

When you add DMS instrumentation, the following users have access to the DMS metrics that you create:

- Applications running in the same OC4J instance can access the DMS metrics.
- All users that have access to the `dmstool` command, or the `AggreSpy` Servlet have access to the metrics (by default this is limited to Administrators).

See Also:

- ["AggreSpy URL and Access Control"](#) on page A-4
- ["Access Control for dmstool"](#) on page A-6

B.5 Conditional Instrumentation Using DMS Sensor Weight

Use the DMS Sensor weight feature to conditionally limit your instrumentation. With Sensor weight, you specify that applications execute expensive instrumentation only when the Sensor weight is set to a particular value. Using this feature enables you to include expensive metrics that you may only need for debugging.

[Example B-10](#) shows how to use `DMSConsole.getSensorWeight()` to test the value of the Sensor weight, and optionally define and use a metric.

The Sensor weight is set globally using the `oracle.dms.sensors` property on the command-line. Set this property using the OC4J startup options. Supported values for this property include: `none`, `normal`, `heavy`, and `all`.

Example B-10 Using SensorWeight for Conditional Instrumentation

```
/* DMS Method
 *
 * If the SensorWeight is high enough, return a phase with the
 * parameter in the name. Otherwise, return null.
 */
PhaseEvent heavyPhase(String param) {
    PhaseEvent pe = null;
    if (DMSConsole.getSensorWeight() > DMSConsole.NORMAL) {
        Noun base = Noun.create(binRoot, param, "MathSeries");
        pe = PhaseEvent.create(base, "computeSeries",
                               "Time to compute a Binomial series");
        pe.deriveMetric(Sensor.all);
    }
    return pe;
}
```

B.6 Dumping DMS Metrics To Files

In a Java application, use the following method to dump DMS metrics to a file.

The following code allows you to append or replace the contents of the specified file with the current metrics:

```
DMSConsole cons2 = new DMSConsole();
```

```
DMSConsole.dump("dmsmathseries.log", true, true);
```

The first argument specifies the file path name, the second argument specifies the output format, and the third argument specifies if the output is appended to the file or replaces the contents of the file.

B.7 Resetting and Destroying Sensors

The Sensor abstract class provides methods to control PhaseEvent, Event, and State Sensors. The `reset()` method resets a Sensor's metrics to initial values. The `getResetTime()` method determines if a Sensor has been reset. The `destroy()` method removes a Sensor from DMS and releases references to its underlying resources.

Note: Do not use these methods to reset or destroy built-in metrics. The `reset()` and `destroy()` methods are intended for use with metrics that you create. Application Server Control Console, and other Oracle Application Server administrative facilities could report unexpected values or have unexpected behavior if you use these methods on internal, built-in metrics.

B.8 DMS Coding Recommendations

The following list includes coding recommendations for working with DMS.

1. There is a global name space for DMS metrics. When you create a new Noun Sensor (PhaseEvent, Event, or State), its full name must not conflict with names in use by Oracle built-in metrics, or by other applications. It is therefore a good idea to have a root Noun for your application that contains the application's full name. This prevents name space collisions.

See Also: ["General DMS Naming"](#) on page B-6

2. Be sure all PhaseEvents are stopped. If the code block to be measured is not in a `try` block, then put it in a `try` block that includes PhaseEvent's `start()`. Put the PhaseEvent's `stop()` in a `finally` block. Alternatively, make use of the `abort()` method in the `finally` block, as shown in [Example B-5](#).

See Also: ["Using PhaseEvent Sensors"](#) on page B-10

3. Use the DMS naming conventions.

See Also: ["DMS Naming Conventions"](#) on page B-6

4. Avoid creating any DMS Sensor or Noun more than once. The DMS API allows this, and avoids creation of multiple objects, but DMS performs lookups for each subsequent creation attempt. Thus, whenever possible, you should define Sensors and Nouns during static initialization, or in the case of a Servlet, in the `init()` method.
5. Assign a type for each Noun that contains Sensors. If no type is assigned, the type is given the value "n/a" (not available). Nouns with the type specified as "n/a" are not shown in the AggreSpy display.

6. Only use PhaseEvents to measure a section of code that is expensive to execute, and takes a significant time to execute under some conditions. In the case where the code never takes significant time to execute, use an Event metric, or remove the PhaseEvent.
7. The DMS API calls are threadsafe; they provide sufficient synchronization to prevent races and access bugs.

B.8.1 Isolating 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.

Try to observe the following guidelines when you add DMS metrics:

1. Add PhaseEvent Sensors 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.
2. When your code calls external code that you do not control, and that you expect could take a significant amount of time, add a PhaseEvent Sensor to track the start and the completion of the external code.

Following these guidelines for adding PhaseEvent metrics provides the following benefits:

- Helps to limit the amount of information that DMS collects.
- Allows those analyzing the system to prove that a module gives the expected runtime performance.
- Ensures that people viewing DMS metrics can validate runtime performance without seeing an overwhelming amount of data.
- Allows those analyzing system performance to separate and track your module from other system modules that are either expensive or failure prone.

B.9 Using A High Resolution Clock To Increase DMS Precision

By default DMS uses the system clock for measuring time intervals during a PhaseEvent. The default clock reports microsecond precision in C processes such as Apache and reports millisecond precision in Java processes such as OC4J. Optionally, DMS supports a high resolution clock to increase the precision of performance measurements and lets you select the units for reporting time intervals. You can use a high resolution clock when you need to time phase events more accurately than is possible using the default clock or when the system's default clock does not provide the resolution needed for your requirements.

Note: The resolution of the default clock and of the high resolution clock is system dependent. On some systems the default clock may not provide sufficient resolution for timing requirements. In particular, on Windows platforms, many users request greater precision than the default clock provides, because it advances only once every 15 milliseconds. DMS metrics reported for brief events on these systems must be analyzed with care. Consider using the high resolution clock to address this issue.

This section covers the following topics:

- [Configuring DMS Clocks for Reporting Time for OC4J \(Java\)](#)
- [Configuring DMS Clocks for Reporting Time for Oracle HTTP Server](#)

B.9.1 Configuring DMS Clocks for Reporting Time for OC4J (Java)

For Java processes, the default clock uses `java.lang.System.currentTimeMillis()`. Selecting the high resolution clock changes this call for all applications running on the process where the clock is changed. You set the DMS clock and the reporting units globally using the `oracle.dms.clock` and `oracle.dms.clock.units` properties, which control process startup options.

For example, to use the high resolution clock with the default units, set the following property on the Java command line for OC4J.

```
-Doracle.dms.clock=highres
```

Caution: Using the high resolution clock, the default units are different than the value that Application Server Control Console expects (msecs). If you need the Application Server Control Console displays to be correct when using the high resolution clock, then you need to set the units property as follows:

```
-Doracle.dms.clock.units=msecs
```

[Table B–6](#) shows supported values for the `oracle.dms.clock` property.

[Table B–7](#) shows supported values for the `oracle.dms.clock.units` property.

See Also:

Table B–6 *oracle.dms.clock Property Values*

Value	Description
DEFAULT	Specifies that DMS use the default clock. With the default clock, DMS uses the Java call <code>java.lang.System.currentTimeMillis()</code> to obtain times for PhaseEvents. The default value for the units for the default clock is MSECS.
HIGHRES	Specifies that DMS use the high resolution clock. DMS accesses the high resolution clock using JNI (the JNI calls depend on the clocks available on the underlying operating system). The default value for the units for the HIGHRES clock is NSECS.

Note: On Windows platforms with a Pentium processor, DMS uses the `QueryPerformanceCounter` function to provide timing for the high resolution clock (HIGHRES). If you are running on a system without a Pentium processor, DMS uses the DMS C clock to provide timing for the high resolution clock. The DMS C clock has microsecond precision which offers a significant improvement over the default clock available with `System.currentTimeMillis()`.

Table B-7 *oracle.dms.clock.units Property Values*

Value	Description
MSECS	Specifies that the time be converted to milliseconds and reported as "msecs". Note: This is the default value for the default clock.
NSECS	Specifies that the time be converted to nanoseconds and reported as "nsecs". Note: This is the default value for the high resolution clock.
USECS	Specifies that the time be converted to microseconds and reported as "usecs".

Note the following when using the high resolution DMS clock:

- When you set the `oracle.dms.clock` and the `oracle.dms.clock.units` properties, any combination of upper and lower case characters is valid for the value that you select (case is not significant). For example, any of the following values are valid to select the high resolution clock: `highres`, `HIGHRES`, `HighRes`.
- DMS checks the property values at startup. When you set the clock with a value that does not match those listed in [Table B-6](#), then DMS uses the default clock. If the `oracle.dms.clock` property is not set, DMS also uses the default clock.
- If the specified clock units property value does not match those listed in [Table B-7](#), then DMS uses the default units for the specified clock. If the `oracle.dms.clock.units` property is not set, DMS uses the default units for the specified the clock.

[Table B-8](#) lists the platform specific environment variables settings for supported platforms. To use the high resolution DMS clock, the environment variables need to be set appropriately. The high resolution clock uses the DMS C library. On UNIX systems, this requires `libdms2.so` to be in the specified environment variable path. On Windows systems this requires `yod.dll` to be in the `PATH` environment. If a nanosecond clock is not available, high resolution timings use a microsecond clock.

Table B-8 *Library Path Environment Variables for Supported Platforms*

Platform	Environment Variable
AIX	LIBPATH \$ORACLE_HOME/lib/libdms2.so is required in the path LD_LIBRARY_PATH \$ORACLE_HOME/lib/libdms2.so is required in the path

Table B–8 (Cont.) Library Path Environment Variables for Supported Platforms

Platform	Environment Variable
HP-UX	SHLIB_PATH \$ORACLE_HOME/lib/libdms2.so is required in the path LD_LIBRARY_PATH \$ORACLE_HOME/lib/libdms2.so is required in the path
Linux	LD_LIBRARY_PATH \$ORACLE_HOME/lib/libdms2.so is required in the path
Tru64 UNIX	LD_LIBRARY_PATH \$ORACLE_HOME/lib/libdms2.so is required in the path
Solaris	LD_LIBRARY_PATH \$ORACLE_HOME/lib/libdms2.so is required in the path
Windows 2000	%ORACLE_HOME%\Apache\Apache\yod.dll must be in the PATH
Windows 2003	%ORACLE_HOME%\Apache\Apache\yod.dll must be in the PATH
Windows XP	%ORACLE_HOME%\Apache\Apache\yod.dll must be in the PATH

B.9.2 Configuring DMS Clocks for Reporting Time for Oracle HTTP Server

The default clock for measuring Oracle HTTP Server performance has a resolution of microseconds (usecs). You can optionally select a higher resolution clock to monitor C processes running under Oracle HTTP Server. To use the High Resolution clock under Oracle HTTP Server, you need to set configuration options in httpd.conf, or specify environment variables on the command line.

Table B–9 lists the environment variables that control the Oracle HTTP Server DMS clock. Table B–10 describes the httpd.conf configuration options that control the Oracle HTTP Server DMS clock. If you set both the command line options and the httpd.conf configuration options, the configuration options override the values set on the command line.

Table B–9 OHS DMS Clock Environment Variables

Environment Variable	Description
DMS_CLOCK	Specifies the clock to use for DMS timing. The values are interpreted the same as with oracle.dms.clock. Valid Values: DEFAULT, HIGHRES
DMS_CLOCK_UNITS	Specifies the units for reporting DMS timing values. The values are Interpreted the same as with oracle.dms.clock.units. Valid Values: MSECS, NSECS, USECS Default Value: USECS

Table B–10 OHS DMS Clock Configuration Parameters

Parameter	Description
DmsClock	Specifies the clock for HTTP listener processes started by OHS, as the oracle.dms.clock property does for Java processes. Valid Values: DEFAULT, HIGHRES

Table B-10 (Cont.) OHS DMS Clock Configuration Parameters

Parameter	Description
DmsClockUnits	Specifies the time units for HTTP listener processes started by OHS, exactly as the <code>oracle.dms.clock.units</code> property is for Java processes. Valid Values: MSECS, NSECS, USECS Default Value: USECS

Note: On Windows platforms with a Pentium processor, DMS uses the `QueryPerformanceCounter` function to provide timing for the high resolution clock (HIGHRES). If you are running on a system without a Pentium processor, DMS uses the DMS C clock to provide timing for the high resolution clock. The DMS C clock has microsecond precision which offers a significant improvement over the default clock available with `System.currentTimeMillis()`.

For example, if you want to use the high resolution clock and use the same units to show times for Java processes running under OC4J and for `mod_oc4j` running under Oracle HTTP Server, update the Oracle HTTP Server `httpd.conf` file to include the following parameters and values:

```
DmsClock=HIGHRES
DmsClockUnits=MSECS
```

Also, include the following values as startup options for the OC4J process:

```
-Doracle.dms.clock=HIGHRES
-Doracle.dms.clock.units=MSECS
```

Using these options DMS uses a high resolution clock for all the Oracle HTTP Server processes that it monitors, for the Java OC4J processes that it monitors, and DMS reports values using the milliseconds units (msecs).

Caution: Using the high resolution clock for the Oracle HTTP Server, the default units for the high resolution clock are NSECS on most platforms. If you need to use Application Server Control Console, it expects USECS for the units. If you need the Application Server Control Console displays to be correct when using the high resolution clock, then you need to set the units property as follows:

```
DmsClock=HIGHRES
DmsClockUnits=USECS
```

B.10 Rolling Up DMS Data for Descendent Nouns

Oracle Application Server 10g Release 3 (10.1.3) includes the DMS Rollup feature that lets you specify metric aggregation. You can use the Rollup feature to specify metric aggregation during DMS instrumentation; rollup is specified to apply to descendents of a specified noun type. You can specify whether the rollup should only apply to direct descendents or to all descendents. [Example B-11](#) shows code that generates a DMS tree, as represented in [Figure B-2](#). Each noun of type `myContainer` contains the `percentageFull`, `close`, and `open` Sensors (see [Figure B-2](#)).

Note: The code in [Example B-11](#) generates a noun tree hierarchy that violates the guidance described in, "[Choosing Noun Types](#)" on page B-9. In this example, it makes sense for some nouns to have descendents and ancestors of the same noun type. The rollup feature described in this section can collect data which might otherwise be lost.

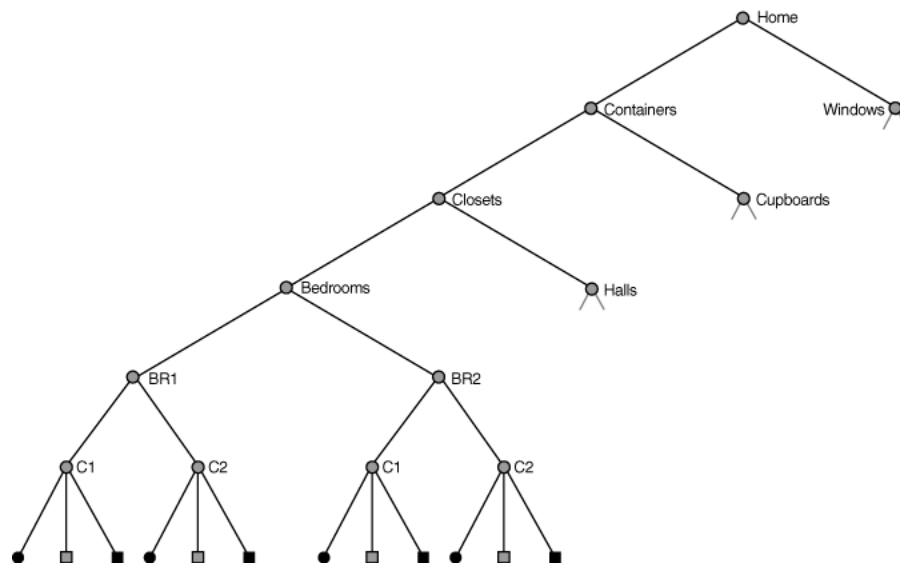
Example B-11 DMS sample code creating noun hierarchy of metrics

```
// Create DMS Noun hierarchy for metrics.
Noun home = Noun.create(Noun.getRoot(), "Home", "myContainer");
Noun containers = Noun.create(home, "Containers", "myContainer");
Noun closets = Noun.create(containers, "Closets", "myContainer");
Noun bedrooms = Noun.create(closets, "Bedrooms", "myContainer");
Noun br1 = Noun.create(bedrooms, "BR1", "myContainer");

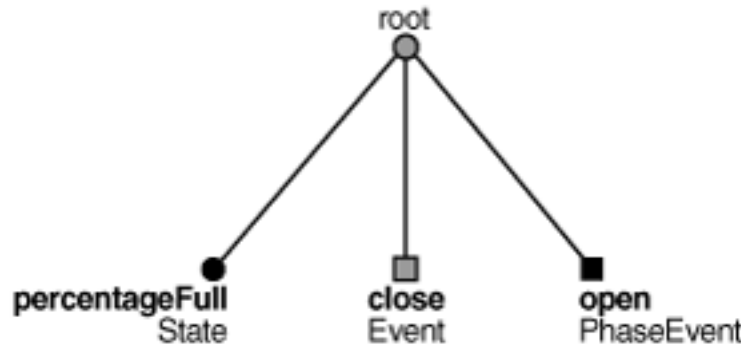
// Create a closet Noun and create Sensors for it.
Noun c1 = Noun.create(br1, "C1", "myContainer");
State percent = State.create(br1, "percentageFull", State.INTEGER, "percent",
"percentage full");
Event close = Event.create(br1, "close", "container closed");
PhaseEvent open = PhaseEvent.create(br1, "open", "open container");

// Derive metrics for State and PhaseEvent Sensors
percent.deriveMetric(Sensor.all);
open.deriveMetric(Sensor.all);
```

Figure B-2 Containers DMS Hierarchy Showing Tree Containing Metrics



[Figure B-3](#) shows a tree with a set of descendent containers. The nouns C1 and C2 under the bedrooms BR1 and BR2 are of type myContainer (see [Figure B-3](#) for a description of myContainer metrics).

Figure B–3 Noun *myContainer* showing Sample Sensors

Using the rollup feature, DMS lets you aggregate a summary for descendent Nouns. For example, you can add the rollup call to a bedrooms noun, as shown in [Example B–11](#). To aggregate *myContainer* type metrics under BR1, use the following call:

```
br1.rollup("myContainer", Noun.DIRECT);
```

This call creates a rollup noun named *myContainer_rollup* under */Home/Containers/Closets/Bedrooms/BR1*. The rollup noun contains the same sensors as the associated noun, including: *percentageFull*, *close*, and *open*.

DMS rollup metrics let you rollup the sensors in all descendent nouns of the given types or only those in the direct descendent nouns. Specifying *Noun.DIRECT* in the rollup call aggregates only direct descendent nouns of the specified type. To aggregate the metrics from all descendent nouns of type *myContainer* instead, use a call such as the following including *Noun.ALL*:

```
closets.rollup("myContainer", Noun.ALL);
```

Rollup metrics include aggregate summary information for their contents. [Table B–11](#) shows the available derived rollup metrics for each Sensor type.

Table B–11 Rollup Metrics Included Derived Metrics

Metric	Description
PhaseEvent	<p>The derived metrics for a PhaseEvent rollup metric include the following:</p> <ul style="list-style-type: none"> time: the sum of time metrics. completed: the sum of the completed metrics. maxTime: the maximum of the maxTime metrics. minTime: the minimum of the minTime metrics. avg: the average time computed for all Sensors. active: the sum of the active metrics.
Event	<p>The derived metrics for a Event rollup metric include the following:</p> <ul style="list-style-type: none"> sum: the total of all count metrics. avg: the average of all count metrics.

Table B–11 (Cont.) Rollup Metrics Included Derived Metrics

Metric	Description
State	The derived metrics for a State rollup metric include the following: <ul style="list-style-type: none"> ■ sum: the total of all value metrics. ■ avg: the average of all value metrics. ■ maxValue: the maximum of the maxValue metrics. ■ minValue: the minimum of the minValue metrics.
descendents	The rollup noun includes a descendents state sensor that reports whether the rollup covers only direct descendents or all descendents.
rolled	The rollup noun includes a rolled state sensor, which reports the number of nouns that are rolled up.
refresh	The rollup noun includes a refresh phase event, which reports the time spent aggregating the metrics for this rollup noun.

[Example B–12](#) shows sample metrics created for the myContainer rollup noun under /Home/Containers/Closets.

Example B–12 Test

```
myContainer_rollup
  descendent.value: all
  percentageFull.sum 40 percent
  percentageFull.avg 10.0 percent
  percentageFull.min 1 percent
  percentageFull.max 29 percent
  close.sum: 3
  close.avg: 0.75
  open.time: 871 msecs
  open.completed: 4 ops
  open.maxTime: 722 msecs
  open.minTime: 23 msecs
  open.avg: 217.7 msecs
  open.active: 0
  rolled.value: 4 nouns
  refresh.maxActive: 1 threads
  refresh.active: 0 threads
  refresh.avg: 0.2857142857142857 msecs
  refresh.maxTime: 1 msecs
  refresh.minTime: 0 msecs
  refresh.completed: 7 ops
  refresh.time: 2 msecs
```

Note that the metrics are similar to the myContainer metrics. The rollup metrics have several key differences, as follows:

1. The rollup noun contains the descendent, rolled, and refresh metrics (see [Table B–11](#) for details).
2. The percentageFull State contains sum and avg metrics rather than the value metric. The name of each metric reflects its content.
3. The close Event contains sum and avg metrics rather than the count metric. The name of each metric reflects its content.
4. The open PhaseEvent does not contain a maxActive metric as it would have no meaning in this context.

See Also: *Oracle Application Server DMS API Reference Javadoc*

Performance Metrics

This appendix lists built-in metrics that can help you analyze Oracle Application Server performance. The metrics fall into several distinct areas, such as Oracle HTTP Server, Oracle Containers for J2EE (OC4J). Each table in this chapter lists the metrics that are included in a corresponding Dynamic Monitoring Services metric table.

This appendix contains:

- [Oracle HTTP Server Metrics](#)
- [JVM Metrics](#)
- [JDBC Metrics](#)
- [OC4J Metrics](#)
- [OC4J JMS Metrics](#)
- [OC4J Task Manager Metrics](#)
- [mod_plsql Metrics](#)
- [Oracle Process Manager and Notification Server - OPMN Metrics](#)
- [DMS Internal Metrics](#)

C.1 Oracle HTTP Server Metrics

The tables, [Table C-1](#) through [Table C-5](#) describe the Oracle HTTP Server metrics.

The metric table name is `ohs_server`.

Table C-1 HTTP Server Metrics (`ohs_server`)

Metric	Description	Unit
<code>busyChildren.value</code>		
<code>childFinish.count</code>	Number of child processes that finish	
<code>childStart.count</code>	Number of child processes that start	
<code>connection.active</code>	Number of connections currently open	threads
<code>connection.avg</code>	Average time spent servicing HTTP connections	usecs
<code>connection.completed</code>	Number of times an HTTP connection was established.	ops
<code>connection.maxTime</code>	Maximum time spent servicing any HTTP connection	usecs
<code>connection.minTime</code>	Minimum time spent servicing any HTTP connection	usecs
<code>connection.time</code>	Total time spent servicing HTTP connections	usecs
<code>error.count</code>		
<code>get.count</code>		
<code>handle.active</code>	Child servers currently in the handle processing phase	threads
<code>handle.avg</code>	Average time spent in module handler	usecs
<code>handle.completed</code>	Number of times the handle processing phase has completed	ops
<code>handle.maxTime</code>	Maximum time spent in module handler	usecs
<code>handle.minTime</code>	Minimum time spent in module handler	usecs
<code>handle.time</code>	Total time spent in module handler	usecs
<code>internalRedirect.count</code>	Number of times a module redirected a request to a new, internal URI	ops
<code>lastConfigChange.value</code>		
<code>numChildren.value</code>		
<code>numMods.value</code>	Number of loaded modules	ops
<code>post.count</code>		
<code>readyChildren.value</code>		
<code>request.active</code>	Child servers currently in the request processing phase	threads
<code>request.avg</code>	Average time required to service an HTTP request	usecs
<code>request.completed</code>	Number of HTTP request completed	ops
<code>request.maxTime</code>	Maximum time required to service an HTTP request	usecs
<code>request.minTime</code>	Minimum time required to service an HTTP request	usecs
<code>request.time</code>	Total time required to service HTTP requests	usecs
<code>responseSize.value</code>		

C.1.1 Oracle HTTP Server Child Server Metrics

[Table C-2](#) describes the child server metrics.

The metric table name is `ohs_child`.

Table C–2 Oracle HTTP Server Child Server Metrics (ohs_child)

Metric	Description	Unit
pid.value	Process ID	
slot.value	Slot	
status.value		
time.value		
url.value		

C.1.2 Oracle HTTP Server Responses Metrics

The Oracle HTTP Server responses metrics are included in the metric table named `ohs_responses`. This metric table includes one metric containing the count, number of times the response was generated, for each HTTP response type.

For example, `Success_OK_200.count: 28 ops`.

C.1.3 Oracle HTTP Server Virtual Host Metrics

The Oracle HTTP Server `ohs_vhostSet` and `ohs_virtualHost` metric tables contain information on virtual host names and locations, and request and response metrics.

Table C–3 Oracle HTTP Server Virtual Host Metrics (ohs_virtualHost)

Metric	Description	Unit
request.active	Number of requests currently being processed by this host	threads
request.avg	Average time spent processing requests for this virtual host	usecs
request.completed	Number of requests processed by this virtual host	ops
request.maxTime	Maximum time spent processing any single request for this virtual host	usecs
request.minTime	Minimum time spent processing any single request for this virtual host	usecs
request.time	Total time spent processing requests for this virtual host	usecs
responseSize.value	Size of response	bytes
vhostType.value	Type of virtual host	

C.1.4 Aggregate Module Metrics

Table C–4 Oracle HTTP Server Modules Metrics

Metric	Description	Unit
numMods.value	Number of loaded modules	

C.1.5 HTTP Server Module Metrics

There is one set of metrics for each module loaded into the server.

The metric table name is `ohs_module`.

Table C-5 Oracle HTTP Server Modules/mod_*.c Metrics (ohs_module)

Metric	Description	Unit
decline.count	Number of requests declined	ops
handle.active	Number of requests currently being handled by this module	requests
handle.avg	Average time required for this module	usecs
handle.completed	Number of requests handled by this module	ops
handle.maxTime	Maximum time required for this module	usecs
handle.minTime	Minimum time required for this module	usecs
handle.time	Total time required for this module	usecs

C.1.6 Oracle HTTP Server mod_oc4j Metrics

Table C-6 shows the mod_oc4j Failure Causes metrics. This table represents the categorization of errors that return an INTERNAL_SERVER_ERROR to the client.

The metric table name is mod_oc4j_request_failure_causes.

Table C-6 HTTP Server mod_oc4j Request Failure Causes Metrics

Metric	Description	Unit
IncorrectReqInit.count	The total number of times an internal error occurred. There could be a number of reasons, including: mod_oc4j not finding a connection endpoint, configuration errors, and others.	ops
Oc4jUnavailable.count	The total number of times that an oc4j JVM could not be found to service requests.	ops
UnableToHandleReq.count	The total number of times mod_oc4j declined to handle a request.	ops

Table C-7 shows the mod_oc4j Mount Point metrics. There is one mount point metric table for each mount point specified in mod_oc4j.conf. This table includes a set of metrics for each mount point specified, with each set grouped under the mntPtid. Where *id* is an integer that is automatically generated during module initialization.

The metric table name is mod_oc4j_mount_pt_metrics.

Table C-7 HTTP Server mod_oc4j Mount Point Metrics

Metric	Description	Unit
Destination.value	Specifies the destination name. For example, with: Oc4jMount /j2ee/* home The Destination.value would be home	String
ErrReq.count	Specifies the total number of requests, both session and non-session, that mod_oc4j failed to route to an OC4J.	ops
ErrReqNonSess.count	Specifies the total number of non session requests that mod_oc4j failed to route to an oc4j process.	ops
ErrReqSess.count	Specifies the total number of session requests that mod_oc4j failed to route to an OC4J process.	ops
Failover.count	Specifies the total number of failovers for both nonsession and session requests.	ops
Name.value	Specifies the echo of the value specified as the path for Oc4jMount directive in mod_oc4j.conf. DMS changes certain characters, including: '/' and '*' to '_'. To preserve the actual path names specified, an internal table containing a mapping between mntPtid and the actual path name is created during mod_oc4j initialization. For example, with: Oc4jMount /j2ee/* home Name.value would be /j2ee/*	String

Table C–7 (Cont.) HTTP Server mod_oc4j Mount Point Metrics

Metric	Description	Unit
NonSessFailover.count	Specifies the total number of failovers for nonsession requests. For example, assume that this mount point was serviced by an OC4J Island with three JVM's (JVM1, JVM2 and JVM3). A new non session request is routed to JVM1. JVM1 fails to service the request, and the request is failed over to JVM2. JVM2 fails to service the request, and so the request is failed over to JVM3. At this point the NonSessFailover.count is incremented by 2.	ops
SessFailover.count	Specifies the total number of failovers for session requests. For example, assume that this mount point was serviced by an OC4J Island with three JVM's (JVM1, JVM2 and JVM3). A session request is routed to JVM1. JVM1 fails to service the request. So, the request is failed over to JVM2. At this point the SessFailover.count is incremented by 1. JVM2 fails to service the request, and so the request is failed over to JVM3. At this point the SessFailover.count is incremented by 2.	ops
SucReq.count	Specifies the total number of requests, both session and non-session, that mod_oc4j successfully routed to an OC4J instance.	ops
SucReqNonSess.count	Specifies the total number of non session requests that mod_oc4j successfully routed to an OC4J process.	ops
SucReqSess.count	Specifies the total number of session requests that mod_oc4j successfully routed to an OC4J process.	ops

Table C–8 shows the mod_oc4j Destination Metrics. This table includes a set of metrics for a specific destination. Each destination can have multiple mount points. There is one mntPts subtree for each mount point specified in mod_oc4j.conf.

The metric table name is mod_oc4j_destination_metrics.

Table C–8 HTTP Server mod_oc4j Destination Metrics

Metric	Description	Unit
ErrReq.count	Specifies the total number of requests, both session and non-session, that mod_oc4j failed to route to an OC4J.	ops
ErrReqNonSess.count	Specifies the total number of non session requests that mod_oc4j failed to route to an OC4J process.	ops
ErrReqSess.count	Specifies the total number of session requests that mod_oc4j failed to route to an OC4J process.	ops
Failover.count	Specifies the total number of failovers for both nonsession and session requests.	ops
JVMCnt.value	Specifies the total number of routable OC4J JVMs that belong to this destination.	Number of JVMs
Name.value	Specifies the echo of the value specified as destination for Oc4jMount directive in mod_oc4j.conf, a single destination may appear several times in mod_oc4j.conf. Example: Oc4jMount /j2ee/* home,oc4jinstance2 Name.value would be home,oc4jinstance2	String
NonSessFailover.count	Specifies the total number of failovers for non session requests.	ops
SessFailover.count	Specifies the total number of failovers.	ops
SucReq.count	Specifies the total number of requests, both session and non-session, that mod_oc4j successfully routed to an OC4J.	ops
SucReqNonSess.count	Specifies the total number of non session requests that mod_oc4j successfully routed to an OC4J process.	ops
SucReqSess.count	Specifies the total number of session requests that mod_oc4j successfully routed to an OC4J process.	ops

C.1.7 Oracle HTTP Server SSL Metrics

Table C–9 describes the OSSL metrics. The metric table type ohs_oss1.

Table C–9 OHS_OSSL Metrics

Metric	Description	Unit
entercache.time	SSL entercache was invoked	
handshake.time	SSL handshake was invoked	
closessl.time	SSL connection was closed	
connectssl.time	SSL connection was established	
receive.time	an encrypted message was received	
dataReceive.value	OSSL Data received	
dataSent.value	OSSL Data Sent	
receiveErrors.count	an error occurred in receive	
sendErrors.count	an error occurred in send	
send.time	an encrypted message was sent	
setfixup.time	SSL setfixup was invoked	
checkcrl.time	SSL checkcrl was invoked	
getcache.time	SSL getcache was invoked	

C.2 JVM Metrics

There is one set of metrics for each Java process (OC4J) currently running in the site. The metric table name is JVM.

Table C–10 JVM Metrics (JVM)

Metric	Description	Unit
activeThreadGroups.value	The number of active thread groups in the JVM	integer
activeThreadGroups.minValue	The minimum number of active thread groups in the JVM	integer
activeThreadGroups.maxValue	The maximum number of active thread groups in the JVM	integer
activeThreads.value	The number of active threads in the JVM	threads
activeThreads.minValue	The minimum number of active threads in the JVM	threads
activeThreads.maxValue	The maximum number of active threads in the JVM	threads
upTime.value	Up time for the JVM	msecs
freeMemory.value	The amount of heap space free in the JVM	KB
freeMemory.minValue	The minimum amount of heap space free in the JVM	KB
freeMemory.maxValue	The maximum amount of heap space free in the JVM	KB
totalMemory.value	The total amount of heap space in the JVM	KB
totalMemory.minValue	The minimum amount of total heap space in the JVM	KB
totalMemory.maxValue	The maximum amount of total heap space in the JVM	KB

C.2.1 JVM Properties Metrics

Oracle Application Server creates a metric to track the value of each Java Property available through a call to `System.getProperties()` on any Java process. For each Java Property, a metric is created under the `/JVM/Properties` noun.

For example, each process should have a metric that contains the value of the `java.version` system property named, `/JVM/Properties/java_version.value`. The system converts property name components with a period, '.' to '_'.
_.

If, during the life of a process, a property is deleted from the JVM system properties, the corresponding metric is deleted. If the value changes, this is reflected in the metric value the next time it is accessed. If a new property is added to the system properties, a new metric is created.

Note: The JVM Properties metrics are only available for viewing using the Spies `text` link in AggreSpy, or using the `dmstool` command to display metrics.

Table C–11 *JVM/Properties - JVM System Properties Metrics*

Metric	Description	Unit
A metric is created for each system property. Each property name has any of the "." characters in the name replaced with "_".	Contains the value of the Java system property.	String

C.3 JDBC Metrics

The following tables list the Oracle Application Server JDBC metrics.

C.3.1 JDBC Driver Metrics

There is one set of JDBC Driver metrics per JVM.

The metric table name is `JDBC_Driver`.

Table C–12 */JDBC/Driver - JDBC_Driver Metrics*

Metric	Description	Unit
<code>ConnectionCloseCount.count</code>	Total number of connections that have been closed.	ops
<code>ConnectionCreate.active</code>	Current number of threads creating connections.	ops
<code>ConnectionCreate.avg</code>	Average time spent creating connections.	msecs
<code>ConnectionCreate.completed</code>	Number of times this PhaseEvent has started and ended.	ops
<code>ConnectionCreate.maxTime</code>	Maximum time spent creating connections.	msecs
<code>ConnectionCreate.minTime</code>	Minimum time spent creating connections.	msecs
<code>ConnectionCreate.time</code>	Time spent creating connections.	msecs
<code>ConnectionOpenCount.count</code>	Total number of connections that have been opened.	ops

C.3.2 JDBC Data Source Metrics

The metric table name is `JDBC_DataSource`.

There is one set of data source metrics per data source.

Table C–13 */JDBC/data-source-name - JDBC Data Source Metrics*

Metric	Description	Unit
ConnectionCloseCount.count	Total number of connections that have been closed.	ops
ConnectionCreate.active	Current number of threads creating connections.	ops
ConnectionCreate.avg	Average time spent creating connections.	msecs
ConnectionCreate.completed	Number of times this PhaseEvent has started and ended.	ops
ConnectionCreate.maxTime	Maximum time spent creating connections.	msecs
ConnectionCreate.minTime	Minimum time spent creating connections.	msecs
ConnectionCreate.time	Time spent creating connections.	msecs
ConnectionOpenCount.count	Total number of connections that have been opened.	ops

C.3.3 JDBC Driver Specific Connection Metrics

There is one set of JDBC Connection metrics per connection.

The metric table name is JDBC_Connection.

Table C–14 */JDBC/Driver/CONNECTION - JDBC Driver Connection Metrics*

Metric	Description	Unit
CreateNewStatement.avg	Average time spent creating a new statement.	msecs
CreateNewStatement.completed	Number of times a request for a statement failed to be satisfied from the cache.	ops
CreateNewStatement.maxTime	Maximum time spent creating a new statement.	msecs
CreateNewStatement.minTime	Minimum time spent creating a new statement.	msecs
CreateNewStatement.time	Time spent creating a new statement (this does not include the time required to parse the statement. For information on the metric that includes the parse time see Execute.Time in Table C–17).	msecs
CreateStatement.avg	Average time spent getting a statement from the statement cache.	msecs
CreateStatement.completed	Number of times a request for a statement was satisfied from the cache.	ops
CreateStatement.maxTime	Maximum time spent getting a statement from the statement cache.	msecs
CreateStatement.minTime	Minimum time spent getting a statement from the statement cache.	msecs
CreateStatement.time	Time spent getting a statement from the statement cache.	msecs
JDBC_Connection_URL	Url specified for the connection	
JDBC_Connection_Username	User name used for the connection	
LogicalConnection.value	If this is a physical connection, then this refers to its logical connection, if any.	
StatementCacheHit.count	Statement found in cache	ops
StatementCacheMiss.count	Statement not found in cache	ops

C.3.4 JDBC Data Source Specific Connection Metrics

There is one set of JDBC data source specific connection metrics per data source per connection. The metric table name is JDBC_Connection.

Table C–15 */JDBC/data-source-name/CONNECTION - JDBC Datasource Connection Metrics*

Metric	Description	Unit
CreateNewStatement.avg	Average time spent creating a new statement.	msecs
CreateNewStatement.completed	Number of times a request for a statement failed to be satisfied from the cache.	ops
CreateNewStatement.maxTime	Maximum time spent creating a new statement.	msecs

Table C–15 (Cont.) /JDBC/data-source-name/CONNECTION - JDBC Datasource Connection Metrics

Metric	Description	Unit
CreateNewStatement.minTime	Minimum time spent creating a new statement.	msecs
CreateNewStatement.time	Time spent creating a new statement (this time does not include the time required to parse the statement. For information on the metric that includes the parse time see Execute.Time in Table C–18).	msecs
CreateStatement.avg	Average time spent getting a statement from the statement cache.	msecs
CreateStatement.completed	Number of times a request for a statement was satisfied from the cache.	ops
CreateStatement.maxTime	Maximum time spent getting a statement from the statement cache.	msecs
CreateStatement.minTime	Minimum time spent getting a statement from the statement cache.	msecs
CreateStatement.time	Time spent getting a statement from the statement cache.	msecs
JDBC_Connection_Url	Url specified for the connection	
JDBC_Connection_Username	User name used for the connection	
LogicalConnection.value	If this is a physical connection, then this refers to its logical connection, if any.	
StatementCacheHit.count	Statement found in cache	
StatementCacheMiss.count	Statement not found in cache	

C.3.5 JDBC ConnectionSource Metrics

There is a set of connection source metrics.

The metric table name is JDBC_ConnectionSource.

Table C–16 JDBC Connection Source Metrics

Metric	Description	Unit
CacheFreeSize.count	Number of free slots in the connection cache.	ops
CacheFreeSize.maxValue	Maximum number of free slots in the connection cache.	connections
CacheFreeSize.minValue	Minimum number of free slots in the connection cache.	connections
CacheFreeSize.value	Number of free slots in the connection cache.	connections
CacheGetConnection.active		threads
CacheGetConnection.avg	Average time spent getting a connection from the cache.	msecs
CacheGetConnection.completed	Number of times this PhaseEvent has started and ended.	ops
CacheGetConnection.maxTime	Maximum time spent getting a connection from the cache.	msecs
CacheGetConnection.minTime	Minimum time spent getting a connection from the cache.	msecs
CacheGetConnection.time	Time spent getting a connection from the cache or not.	msecs
CacheHit.count	Number of times a request for a connection has been satisfied from the cache.	
CacheMiss.count	Number of times a request for a connection failed to be satisfied from the cache.	
CacheSize.value	Total size of the connection cache.	

C.3.6 JDBC Driver Statement Metrics

There is a set of statement metrics per connection per statement.

The metric table name is JDBC_Statement.

Note: The JDBC statement metrics are only available for JDBC connections that have enabled statement caching, and set the property `oracle.jdbc.DMSStatementCachingMetrics` to the value `true`. When JDBC statement caching is disabled, you can make the JDBC statement metrics available by setting the property `oracle.jdbc.DMSStatementMetrics` to `true`. To improve performance and to avoid collecting expensive metrics, by default these properties are both set to `false`.

Table C–17 */JDBC/Driver/CONNECTION/STATEMENT JDBC Statement Metrics*

Metric	Description	Unit
<code>Execute.time</code>	The time this statement has spent executing the SQL including the first fetch and the time required to parse the statement.	msecs
<code>Fetch.time</code>	The time this statement has spent in other fetches.	msecs
<code>SQLText.value</code>	The SQL being executed.	

C.3.7 JDBC Data Source Statement Metrics

The metric table name is `JDBC_Statement`.

There is a set of statement metrics per data source per connection per statement.

Note: the JDBC data source metrics are only available for non-emulated data sources.

Note: The JDBC statement metrics are only available for JDBC connections that have enabled statement caching and set the property `oracle.jdbc.DMSStatementCachingMetrics` to the value `true`. When JDBC statement caching is disabled, you can make the JDBC statement metrics available by setting the property `oracle.jdbc.DMSStatementMetrics` to `true`. To improve performance and to avoid collecting expensive metrics, by default these properties are set to `false`.

Table C–18 */JDBC/data-source-name/CONNECTION/STATEMENT JDBC Statement Metrics*

Metric	Description	Unit
<code>Execute.time</code>	The time this statement has spent executing the SQL including the first fetch and the time required to parse the statement.	msecs
<code>Fetch.time</code>	The time this statement has spent in other fetches.	msecs
<code>SQLText.value</code>	The SQL being executed.	

C.4 OC4J Metrics

This section lists the OC4J J2EE application related metrics.

This section covers the following metrics:

- [Web Module Metrics](#)
- [Web Context Metrics](#)
- [OC4J Servlet Metrics](#)

- [OC4J JSP Metrics](#)
- [OC4J EJB Metrics](#)
- [OC4J OPMN Info Metrics](#)
- [OC4J Work Management Pool Metrics](#)

C.4.1 Web Module Metrics

There is one set of metrics for each Web module within each J2EE application.

The metric table name is `oc4j_web_module`.

Table C–19 *OC4J/application/WEBs Metrics*

Metric	Description	Unit
<code>parseRequest.active</code>	Current number of threads trying to read/parse AJP or HTTP requests	
<code>parseRequest.avg</code>	Average time spent to read/parse requests	msecs
<code>parseRequest.completed</code>	Number of web requests that have been parsed	ops
<code>parseRequest.maxActive</code>	Maximum number of threads trying to read/parse AJP or HTTP requests	threads
<code>parseRequest.maxTime</code>	Maximum time spent to read/parse requests	msecs
<code>parseRequest.minTime</code>	Minimum time spent to read/parse requests	msecs
<code>parseRequest.time</code>	Total time spent to read/parse requests from the socket	msecs
<code>processRequest.active</code>	Current number of threads servicing web requests	
<code>processRequest.avg</code>	Average time spent servicing web requests	msecs
<code>processRequest.completed</code>	Number of web requests processed by this application	ops
<code>processRequest.maxActive</code>	Maximum number of threads servicing web requests	threads
<code>processRequest.maxTime</code>	Maximum time spent servicing a web request	msecs
<code>processRequest.minTime</code>	Minimum time spent servicing a web request	msecs
<code>processRequest.time</code>	Total time spent servicing this application's web requests	msecs
<code>resolveContext.active</code>	Current number of threads trying to create/find the servlet context	
<code>resolveContext.avg</code>	Average time spent to create/find the servlet context	msecs
<code>resolveContext.completed</code>	Count of completed context resolves	ops
<code>resolveContext.maxActive</code>	Maximum number of threads trying to create/find the servlet context	threads
<code>resolveContext.maxTime</code>	Maximum time spent to create/find the servlet context	msecs
<code>resolveContext.minTime</code>	Minimum time spent to create/find the servlet context	msecs
<code>resolveContext.time</code>	Total time spent to create/find the servlet context. Each web module (WAR) maps to a servlet context	msecs

C.4.2 Web Context Metrics

There is one set of metrics for each Web context module within each J2EE application.

The metric table name is `oc4j_context`.

Table C–20 *OC4J/application/WEBs/context Metrics*

Metric	Description	Unit
resolveServlet.time	Total time spent to create/locate servlet instances (within the servlet context). This includes the time for any required authentication.	msecs
resolveServlet.completed	Total Number of lookups for a servlet by OC4J	ops
resolveServlet.minTime	Minimum time spent to create/locate the servlet instance (within the servlet context)	msecs
resolveServlet.maxTime	Maximum time spent to create/locate the servlet instance (within the servlet context)	msecs
resolveServlet.avg	Average time spent to create/locate the servlet instance (within the servlet context)	msecs
sessionActivation.active	Number of active sessions	ops
sessionActivation.time	Total time in which sessions have been active	msecs
sessionActivation.completed	Number of session activations	ops
sessionActivation.minTime	Minimum time a session was active	msecs
sessionActivation.maxTime	Maximum time a session was active	msecs
sessionActivation.avg	Average session lifetime	msecs
service.time	Total time spent servicing requests. The service metrics for the servlet include any time spent in the calls to the database. If you need to determine just the oc4j service time, subtract the appropriate execution times (see the JDBC metrics in Table C–18).	msecs
service.completed	Total number of requests serviced	ops
service.minTime	Minimum time spent servicing requests	msecs
service.maxTime	Maximum time spent servicing requests	msecs
service.avg	Average time spent in servicing the servlet	msecs
service.active	Current number of requests active	ops

C.4.3 OC4J Servlet Metrics

There is one set of metrics for each servlet in each Web module within each J2EE application.

The metric table name is `oc4j_servlet`.

Table C–21 *OC4J/application/WEBs/context /SERVLETS/servlet Metrics*

Metric	Description	Unit
service.active	Current number of threads servicing this servlet	threads
service.avg	Average time spent in servicing the servlet	msecs
service.completed	Total number of calls to service()	
service.maxActive	Maximum number of threads servicing this servlet	threads
service.maxTime	Maximum time spent on a servlet's service() call	ops
service.minTime	Minimum time spent on a servlet's service() call	msecs
service.time	Total time spent on the servlet's service() call	msecs

C.4.4 OC4J JSP Metrics

C.4.4.1 JSP Runtime Metrics

There is one set of metrics for each Web context for each J2EE application.

The metric table name is `oc4j_jspExec`.

Table C–22 *OC4J/application/WEBs/context /JSP Metrics*

Metric	Description	Unit
<code>processRequest.time</code>	Time spent processing requests for JSPs Only used for Context/Application name	msecs
<code>processRequest.completed</code>	Number of requests for JSPs processed by this application	ops
<code>processRequest.minTime</code>	Minimum time spent processing requests for JSPs	msecs
<code>processRequest.maxTime</code>	Maximum time spent processing requests for JSPs	msecs
<code>processRequest.avg</code>	Average time spent processing requests for JSPs	msecs
<code>processRequest.active</code>	Current number of active requests for JSPs	ops

C.4.4.2 JSP Metrics

There is one set of metrics for each JSP in each Web module.

The metric table names are `oc4j_jsp(threadsafe=true)` and `oc4j_jsp(threadsafe=false)`.

To list these metrics using `dmstool`, enclose the metric table name in quotation marks.

For example:

```
dmstool -table "oc4j_jsp(threadsafe=true)"
```

Table C–23 *OC4J/application/WEBs/context /JSPjsp_name Metrics*

Metric	Description	Unit
<code>activeInstances.value</code>	Number of active instances. Only used when <code>threadsafe=false</code>	instances
<code>availableInstances.value</code>	Number of available (that is, created) instances. This value is only provided when <code>threadsafe=false</code> .	instances
<code>service.active</code>	Current number of active requests for the JSP	
<code>service.avg</code>	Average time spent servicing the JSP	msecs
<code>service.completed</code>	Number of requests for JSPs processed by this JSP	ops
<code>service.maxTime</code>	Maximum time spent servicing the JSP	msecs
<code>service.minTime</code>	Minimum time spent servicing the JSP	msecs
<code>service.time</code>	Time to serve a JSP (that is, actual execution time of the JSP)	msecs

C.4.5 OC4J EJB Metrics

C.4.5.1 OC4J EJB Session Bean Metrics

The `oc4j_ejb_session_bean` metric table includes information on a session bean.

Table C–24 *OC4J EJB Session Bean Metrics*

Metric	Description	Unit
<code>session-type.value</code>	Provides information on the session type: Stateless or Stateful	String
<code>transaction-type.value</code>	Provides information on the transaction type: Container or Bean	String

C.4.5.2 EJB Bean Metrics

Oracle Application Server provides a set of these metrics for each type of bean in each EJB jar file in each J2EE application.

The metric table name is `oc4j_ejb_entity_bean`.

Table C–25 *OC4J/application/EJBs/ejb-jar-module/ejb-name Metrics*

Metric	Description	Unit
<code>transaction-type.value</code>	Possible values: container or bean	
<code>session-type.value</code>	Possible values: stateful or stateless	
<code>bean-type.value</code>	Possible values: session or entity bean	
<code>exclusive-write-access.value</code>	Possible values: true or false	
<code>isolation.value</code>	Possible values: serializable, uncommitted, committed, repeatable_read, none, DB-determined The value is DB-determined when the isolation attribute is omitted.	
<code>persistence-type.value</code>	Possible values: container or bean or	

C.4.5.3 EJB Method Metrics

There is one set of metrics for each method within each type of EJB bean.

The metric table name is `oc4j_ejb_method`.

The `client.*` metrics show values for the actual implementation of the method. The `wrapper.*` metrics show values for the wrapper that was automatically generated for the method.

See Also: Chapter 6, "Advanced EJB Subjects" in *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide* for information on automatically generated wrappers.

Table C–26 *OC4J/application/EJBs/ejb-jar-module/ejb-name/method-name Metrics*

Metric	Description	Unit
<code>client.active</code>	Current number of threads accessing the actual implementation of this method	ops
<code>client.avg</code>	Average time spent inside the actual implementation of this method	msecs
<code>client.completed</code>	Number of requests for beans processed by this application	ops
<code>client.maxActive</code>	Maximum number of threads accessing the actual implementation of this method	ops
<code>client.maxTime</code>	Maximum time spent inside the actual implementation of this method	msecs
<code>client.minTime</code>	Minimum time spent inside the actual implementation of this method	msecs
<code>client.time</code>	Time spent inside the actual implementation of this method	msecs
<code>ejbPostCreate.active</code>	Current number of threads executing <code>ejbPostCreate</code>	ops
<code>ejbPostCreate.avg</code>	Average time spent in <code>ejbPostCreate</code>	msecs
<code>ejbPostCreate.completed</code>	Number of times this <code>ejbPostCreate</code> has been called	ops
<code>ejbPostCreate.maxTime</code>	Maximum time spent in <code>ejbPostCreate</code>	msecs
<code>ejbPostCreate.minTime</code>	Minimum time spent in <code>ejbPostCreate</code>	msecs
<code>ejbPostCreate.time</code>	Time spent in the <code>ejbPostCreate</code> method (entity beans)	msecs
<code>trans-attribute.value</code>	Transaction attribute. Possible values: NotSupported, Supports, RequiresNew, Mandatory, and Never	

Table C–26 (Cont.) OC4J/application/EJBs/ejb-jar-module/ejb-name/method-name Metrics

Metric	Description	Unit
wrapper.active	Current number of threads accessing the automatically generated wrapper method	
wrapper.avg	Average time spent inside the automatically generated wrapper method	msecs
wrapper.completed	Number of requests for beans processed by this application	ops
wrapper.maxActive	Maximum number of threads that access the wrapper	ops
wrapper.maxTime	Maximum time spent inside the automatically generated wrapper method	msecs
wrapper.minTime	Minimum time spent inside the automatically generated wrapper method	msecs
wrapper.time	Time spent inside the automatically generated wrapper method. Note: Not all the wrapper methods invoke the actual bean implementation at runtime (for example, create method in a stateless bean). This means that the time spent in the wrapper code could be less than the time spent in the bean implementation	msecs

C.4.6 OC4J OPMN Info Metrics

Table C–27 shows the OC4J OPMN information metrics. The metric table type is `oc4j_opmn`.

Table C–27 OC4J OPMN Information Metrics

Metric	Description	Unit
default_application_log.value	Specifies the default application log file path.	
ias_cluster.value	Specifies the Oracle Application Server cluster name.	String
ias_instance.value	Specifies the Oracle Application Server instance name.	String
jms_log.value	Specifies the JMS log file path.	String
oc4j_instance.value	Specifies the OC4J instance ID.	String
oc4j_island.value	Specifies the OC4J island ID.	String
opmn_group.value	Specifies the OPMN group ID.	String
opmn_sequence.value	Specifies the OPMN sequence ID.	String
rmi_log.value	Specifies the RMI log file path name.	String
server_log.value	Specifies the application server log file path.	String

C.4.7 OC4J Work Management Pool Metrics

Table C–28 shows the OC4J Work management pool metrics. The metric table type is `oc4j_workManagementPool`.

Table C–28 OC4J Work Management Pool Metrics

Metric	Description	Unit
idleThreadCount	number of idle threads in the pool. This is a current thread pool state metric.	threads
keepAlive	Time before idle threads are removed from available pool. This is a configuration value metric	milliseconds
maxPoolSize	maximum number of threads in the pool. This is a configuration value metric	threads
maxQueueSize	maximum queue size. This is a configuration value metric.	work_requests
minPoolSize	minimum number of threads in the pool. This is a configuration value metric.	threads

Table C–28 (Cont.) OC4J Work Management Pool Metrics

Metric	Description	Unit
queueFullEvent	number of work submission failures due to full queue. This is a current thread pool state metric.	ops
queueSize	current queue size. This is a current thread pool state metric.	work_requests
totalThreadCount	Total number of threads in the pool. This is a current thread pool state metric.	threads
workStartDuration	Duration between work accepted and work started events. This is a current thread pool state metric. Waiting time is defined as the time period between the work submission is accepted and the execution of the work starts. This metric measures the duration between a work request submission is accepted by the pool and the time when a thread is allocated from the thread pool to run the work. If a thread is readily available, this would measure the processing overhead of the threadpool in finding an available thread and setting up the proper context for processing the work. If all available threads are busy handling other work requests, this time would also include the queuing time.	ops

C.5 OC4J JMS Metrics

OC4J JMS metrics are organized into metric tables and fall into two categories:

- JMS API-level metrics: collected on objects visible to the JMS API (for example, connections, sessions, producers, consumers, and browsers). JMS API-level metrics are collected and maintained only for Web and EJB clients (application clients also collect API-level metrics, but do so in their own JVM; these metrics are not available on the OC4J JMS server).
- JMS Server-level metrics: collected by the OC4J JMS server and maintained independent of client-state. JMS Server-level metrics are collected and maintained for all types of clients: Application, Web, and EJB.

Each OC4J JMS metric table (metric table type) contains metrics for instances of the same type; different instances have unique names. For each instance in a metric table, a set of metrics is collected. The names for metrics in each instance are unique IDs that OC4J JMS generates.

Instances may have one or more metrics whose value is the name of another metric instance. For example, the JMS session instances contain metrics that point to the parent containing JMS connection instance. You can use the pointers to navigate through the metrics.

A parent metric instance usually includes a counter metric indicating the number of child metrics of a certain type that have been created. Child metric instances may appear and disappear as the underlying objects are created and destroyed; the counter keeps track of the total number of such instances that were created during the lifetime of the parent.

Note: Oracle Application Server JMS metrics are available only for OC4J JMS (thus, metrics are not available for OJMS).

See Also: *Oracle Containers for J2EE Services Guide* for more information on OC4J JMS

C.5.1 JMS Metric Tables

OC4J JMS metrics are divided into three types, based on how they are updated:

1. **CTOR Metrics:** Metrics that are set in the constructor or initialization routine of the associated JMS object, and are never changed during the lifetime of the object.
2. **Normal Metrics:** Object level state metrics that are updated as soon as the associated state of the JMS object changes.
3. **Lazy Metrics:** these state metrics are updated lazily, that is, not as soon as the underlying metric value changes, but only periodically (these are typically server store metrics and are updated each time the store is cleaned up of expired messages).

Table C–29 shows a summary of the organization of the OC4J JMS metric tables.

Table C–29 OC4J JMS Metric Tables

JMS Metric Table Type	Parent Table Type	Number of Instances	Description
JMSStats	none	1	Statistics for the OC4J JMS Server
JMSRequestHandlerStats	JMSStats	1 per remote JMS connection	Statistics for the request handler thread servicing a remote JMS connection.
JMSConnectionStats	JMSStats	1 per JMS connection	Statistics for the JMS connections active in this server
JMSSessionStats	JMSConnectionStats	1 per JMS session	Statistics for the JMS sessions active in this server
JMSMessageProducerStats	JMSSessionStats	1 per JMS message producer	Statistics for the JMS producers active in this server
JMSMessageBrowserStats	JMSSessionStats	1 per JMS queue browser	Statistics for the JMS queue browsers in this server
JMSMessageConsumerStats	JMSSessionStats	1 per JMS message consumer	Statistics for the JMS consumers active in this server
JMSDurableSubscriberStats	JMSStats	1 per JMS durable subscriber	Statistics for each JMS durable subscription known to this server
JMSDestinationStats	JMSStats	1 per permanent JMS destination	Statistics for each permanent JMS destination known to the OC4J JMS server
JMSTemporaryDestinationStats	JMSStats	1 per temporary JMS destination	Statistics for each temporary JMS destination known to the OC4J JMS server
JMSStoreStats	JMSDestinationStats JMSTemporaryDestinationStats	1 per server-side message store	Statistics for each message store (one per queue, one per subscription per topic) on the OC4J JMS server
JMSPersistenceStats	JMSDestinationStats	1 per server-side persistent destination	Statistics for operations on the persistence file for each persistent destination

C.5.2 JMS Stats Metric Table

Table C–30 shows the JMS Stats metrics.

The metric table type is JMSStats.

Table C–30 JMSSStats Metric Table

Metric	Description	Update	Unit
activeConnections.time			
activeHandlers.time			
address.value	The hostname(s) from which the JMS server accepts remote connections	ctor	string

Table C–30 (Cont.) JMSStats Metric Table

Metric	Description	Update	Unit
closeConnection.time			
closeConsumer.time			
commit.time			
connections			
createConsumer.time			
deqMessage.time			
enqMessage.time			
host.value	The explicit hostname on which the OC4J JMS server is running.	ctor	string
listMessages			
messageCommitted			
messageCount			
messageDequeued			
messageDiscarded			
messageEnqueued			
messageExpired			
messagePagedIn			
messagePagedOut			
messageRecovered			
messageRolledback			
oc4j.jms.checkPermissions			
oc4j.jms.debug.value	Value of the oc4j.jms.debug OC4J JMS control knob	ctor	bool
oc4j.jms.forceRecovery.value	Value of the oc4j.jms.forceRecovery OC4J JMS control knob	ctor	bool
oc4j.jms.j2ee14			
oc4j.jms.listenerAttempts.	Value of the oc4j.jms.listenerAttempts OC4J JMS control knob	ctor	int
oc4j.jms.maxOpenFiles.value	Value of the oc4j.jms.maxOpenFiles OC4J JMS control knob	ctor	int
oc4j.jms.messagePoll.value	Value of the oc4j.jms.messagePoll OC4J JMS control knob	ctor	msecs
oc4j.jms.noDms.value	Value of the oc4j.jms.noDms OC4J JMS control knob	ctor	bool
oc4j.jms.noJmx			
oc4j.jms.pagingThreshold			
oc4j.jms.printStackTrace			
oc4j.jms.reconnectAttempts			
oc4j.jms.reconnectWait			
oc4j.jms.rememberALLXids			
oc4j.jms.saveAllExpired.val	Value of the oc4j.jms.saveAllExpired OC4J JMS control knob	ctor	bool
oc4j.jms.serverPoll.value	Value of the oc4j.jms.serverPoll OC4J JMS control knob	ctor	msecs
oc4j.jms.socketBufsize.val	Value of the oc4j.jms.socketBufsize OC4J JMS control knob	ctor	int
pendingMessageCount			

Table C–30 (Cont.) JMSStats Metric Table

Metric	Description	Update	Unit
port.value	The TCP/IP port on which the JMS server listens for incoming connections	ctor	int
registerConnection			
rollback			
startTime.value	System.currentTimeMillis() when the OC4J JMS server was started	ctor	msecs
stats			
storeSize			
taskManagerInterval.value	The scheduling interval of the OC4J task manager (and the scheduling interval for the OC4J JMS expiration task)	ctor	msecs
instance			
uid			
method-name	An interval timer metric (PhaseEvent Sensor) for every major method call in the OC4J JMS server	normal	

C.5.3 JMS Request Handler Stats

Table C–31 shows the JMS Request Handler Stats.

The metric table name is JMSRequestHandlerStats.

Table C–31 JMSRequestHandlerStats Metrics

Metric	Description	Update	Unit
address.value	The hostname from which the remote connection originates (may be an implicit, special address)	ctor	string
connectionID.value	The ID of the JMSConnectionStats instance	ctor	string
host.value	The explicit hostname from which the remote connection originates	ctor	string
port.value	The TCP/IP port from which the remote connection originates	ctor	int
startTime.value	System.currentTimeMillis() when the request handler was started	ctor	string

C.5.4 JMS Connection Stats

Table C–32 shows the JMS Connection Stats.

The metric table name is JMSConnectionStats.

Table C–32 JMSConnectionStats Metrics

Metric	Description	Update	Unit
address.value	The implicit hostname of the remote JMS server host for this connection as specified in the connection factory used to create this connection; set only for non-local connections.	ctor	string
clientID.value	The administratively configured (for ctor) or programmatically set (for normal) clientID for this connection	ctor/normal	string
domain.value	The JMS domain ("queue", "topic", or "unified") of this connection	ctor	string
exceptionListener.value	The stringified name of the current exception listener for this connection	normal	string
host.value	The explicit hostname of the remote JMS server host for this connection; set only for non-local connections	ctor	string

Table C–32 (Cont.) JMSConnectionStats Metrics

Metric	Description	Update	Unit
isLocal.value	"true" if and only if the JMS connection is local to the OC4J JMS server in the same JVM	ctor	boolean
isXA.value	"true" if and only if the connection is in XA mode	ctor	boolean
port.value	The remote JMS server port for this connection; set only for non-local connections	ctor	int
startTime.value	System.currentTimeMillis() when this connection was created	ctor	msecs
user.value	The user identity for this connection	ctor	string
method-name	An interval timer metric (PhaseEvent Sensor) for every major method call in this connection object.	normal	

C.5.5 JMS Session Stats

Table C–33 shows the JMS Session Stats.

The metric table name is JMSSessionStats.

Table C–33 JMSSessionStats Metrics

Metric	Description	Update	Unit
acknowledgeMode.value	The acknowledge mode of this session. The valid modes are: AUTO_ACKNOWLEDGE, CLIENT_ACKNOWLEDGE, DUPS_OK_ACKNOWLEDGE, and SESSION_TRANSACTED.	ctor	string
domain.value	The JMS domain ("queue", "topic", or "unified") of this session	ctor	string
isXA.value	"true" if and only if the session is in XA mode	ctor	boolean
sessionListener.value	The stringified name of the current distinguished listener for this session	normal	string
startTime.value	System.currentTimeMillis() when this session was created	ctor	msecs
transacted.value	"true" if and only if the session is transacted	ctor	boolean
txid.value	The integer count of the current local transaction associated with this session; the counter is increment each time a local transaction is committed/rolledback; not set for non-transacted session	normal	int
xid.value	The Xid of the current distributed transaction associated with this session; set to a null/empty string when in a local transaction mode; not set if the session never participates in a global transaction	normal	string
method-name	An interval timer metric (PhaseEvent Sensor) for every major method call in this session object	normal	

C.5.6 JMS Message Producer Stats

Table C–34 shows the JMS Producer Stats.

The metric table name is JMSProducerStats.

Table C–34 JMSProducerStats Metrics

Metric	Description	Update	Unit
deliveryMode.value	The current delivery mode of this producer. The valid delivery mode values are: PERSISTENT and NON_PERSISTENT.	normal	string
destination.value	The name of the identified destination for this producer; null/empty for an unidentified producer	ctor	string
disableMessageID.value	The value is true when message IDs are disabled for the producer	normal	boolean
disableMessageTimestamp.value	The value is true when message timestamps are disabled for the producer	normal	boolean
domain.value	The JMS domain ("queue", "topic", or "unified") of this producer	ctor	string

Table C–34 (Cont.) JMSProducerStats Metrics

Metric	Description	Update	Unit
<code>priority.value</code>	The current priority of this producer	normal	int
<code>startTime.value</code>	<code>System.currentTimeMillis()</code> when this producer was created	ctor	msecs
<code>timeToLive.value</code>	The current <code>timeToLive</code> of this producer	normal	msecs
<i>method-name</i>	A phase timer (PhaseEvent Sensor) metric for every major method call in this producer object	normal	

C.5.7 JMS Message Browser Stats

Table C–35 shows the JMS Browser Stats.

The metric table name is `JMSBrowserStats`.

Table C–35 JMSBrowserStats Metrics

Metric	Description	Update	Unit
<code>destination.value</code>	The name of the destination for this browser	ctor	string
<code>selector.value</code>	The message selector for this browser; null/empty string if unspecified	ctor	string
<code>startTime.value</code>	<code>System.currentTimeMillis()</code> when this browser was created	ctor	msecs
<i>method-name</i>	An interval timer metric (PhaseEvent Sensor) for every major method call in this browser object; calls to "hasMoreElements" and "nextElement" are made on individual enumeration objects, but counted as PhaseEvents in the browser object to simplify data collection, multiple enumerations can be active on the same browser	normal	

C.5.8 JMS Message Consumer Stats

Table C–36 shows the JMS Message Consumer Stats.

The metric table name is `JMSMessageConsumerStats`.

Table C–36 JMSMessageConsumerStats

Metric	Description	Update	Unit
<code>destination.value</code>	The name of the destination for this consumer	ctor	string
<code>domain.value</code>	The JMS domain ("queue", "topic", or "unified") of this consumer	ctor	string
<code>messageListener.value</code>	The stringified name of the current message listener for this consumer	normal	string
<code>name.value</code>	The name of the durable subscriber for this consumer; set only for durable topic subscriptions	ctor	string
<code>noLocal.value</code>	The <code>noLocal</code> setting of a subscription; set only for topic consumers	ctor	boolean
<code>selector.value</code>	The message selector for this consumer; null/empty string if unspecified	ctor	string
<code>startTime.value</code>	<code>System.currentTimeMillis()</code> when this consumer was created	ctor	msecs
<i>method-name</i>	An interval timer metric (PhaseEvent Sensor) for every major method call in this consumer object	normal	

C.5.9 JMS Durable Subscription Stats

Table C–37 shows the JMS Durable Subscription Stats.

The metric table name is `JMSDurableSubscriptionStats`.

Table C–37 JMSDurableSubscriptionStats Metrics

Metric	Description	Update	Unit
clientID.value	The clientID associated with this durable subscriptions	ctor	string
destination.value	The name of the topic for this durable subscription	ctor	string
isActive.value	"true" if and only if the durable subscription is currently active (being used by a consumer)	normal	boolean
name.value	The user-provided name of the durable subscription	ctor	string
noLocal.value	The noLocal flag for this durable subscription	ctor	boolean
selector.value	The JMS message selector for this durable subscription	ctor	string

C.5.10 JMS Destination Stats

[Table C–38](#) shows the JMS Destination Stats metrics

The metric table name is JMSDestinationStats.

Table C–38 JMSDestinationStats Metrics

Metric	Description	Update	Unit
domain.value	JMS domain, "queue" or "topic", of the destination	ctor	string
name.value	The configured name of the destination. As defined in <code>jms.xml</code>	ctor	string
locations.value	A comma-delimited list of JNDI names bound to the destination. As defined in <code>jms.xml</code>	ctor	string
method-name	An interval timer metric (PhaseEvent Sensor) for every major method call in the destination object	normal	

C.5.11 JMS Temporary Destination Stats

[Table C–39](#) shows the JMS Temporary Destination Stats.

The metric table name is JMSTemporaryDestinationStats.

Table C–39 JMSTemporaryDestinationStats Metrics

Metric	Description	Update	Unit
connectionID.value	The ID of the JMSConnectionStats instance from which this temporary destination was created	ctor	string
domain.value	JMS domain, for example "queue" or "topic", of the destination	ctor	string
method-name	An interval timer metric (PhaseEvent Sensor) for every major method call in the destination object	normal	

C.5.12 JMS Store Stats

[Table C–40](#) shows the JMS StoreStats metric table.

The metric table name is JMSStoreStats.

Table C–40 JMSStoreStats Metric

Metric	Description	Update	Unit
destination.value	A pretty-printed name of the JMS destination associated with this message store	ctor	string
messageCount.value	Total number of messages contained in this store	lazy	int
messageDequeued.count	Total number of message dequeues (transacted or otherwise)	normal	ops
messageDiscarded.count	Total number of message discarded after the rollback of an enqueue	normal	ops
messageEnqueued.count	Total number of message enqueues (transacted or otherwise)	normal	ops

Table C–40 (Cont.) JMSSStoreStats Metric

Metric	Description	Update	Unit
messageExpired.count	Total number of message expirations	normal	ops
messagePagedIn.count	Total number of message bodies paged in	normal	ops
messagePagedOut.count	Total number of message bodies paged out	normal	ops
messageRecovered.count	Total number of messages recovered (either from a persistence file, or after the rollback of a dequeue)	normal	ops
pendingMessageCount.value	Total number of messages part of an enqueue/dequeue of an active transaction	lazy	int
storeSize.value	Total size, in bytes, of the message store.	lazy	bytes
method-name	An interval timer metric (PhaseEvent Sensor) for every major method call in the message store object	normal	

The following identity holds:

$$\text{messageCount} = \text{messageRecovered} + \text{messageEnqueued} - \text{messageDequeued} - \text{messageDiscarded} - \text{messageExpired}$$

If a message is both enqueued and dequeued in the same transaction, the messageEnqueued and messageDequeued events occur, but the messageRecovered and messageDiscarded events do not.

C.5.13 JMS Persistence Stats

Table C–41 shows the JMS Persistence Stats.

The metric table name is JMSPersistenceStats.

Table C–41 JMSPersistenceStats Metrics

Metric	Description	Update	Unit
destination.value	A pretty-printed name for the JMS destination associated with this persistence file	ctor	string
holePageCount.value	The number of 512b pages currently free in this file	normal	int
isOpen.value	"true" iff the persistence file descriptor is currently open (for LRU caching)	normal	boolean
lastUsed.value	System.currentTimeMillis() when this persistence file was last used (for LRU caching)	normal	msecs
persistenceFile.value	The absolute path name of the persistence file used for this persistent destination. This value differs depending on the operating system where OC4J is running.	ctor	string
usedPageCount.value	The number of 512b pages currently in use in this file	normal	int
method-name	An interval timer metric (PhaseEvent Sensor) for every major method call in the persistence file object	normal	

C.6 OC4J Task Manager Metrics

The metric table type is oc4j_task.

Table C–42 *OC4J_taskManager Metrics*

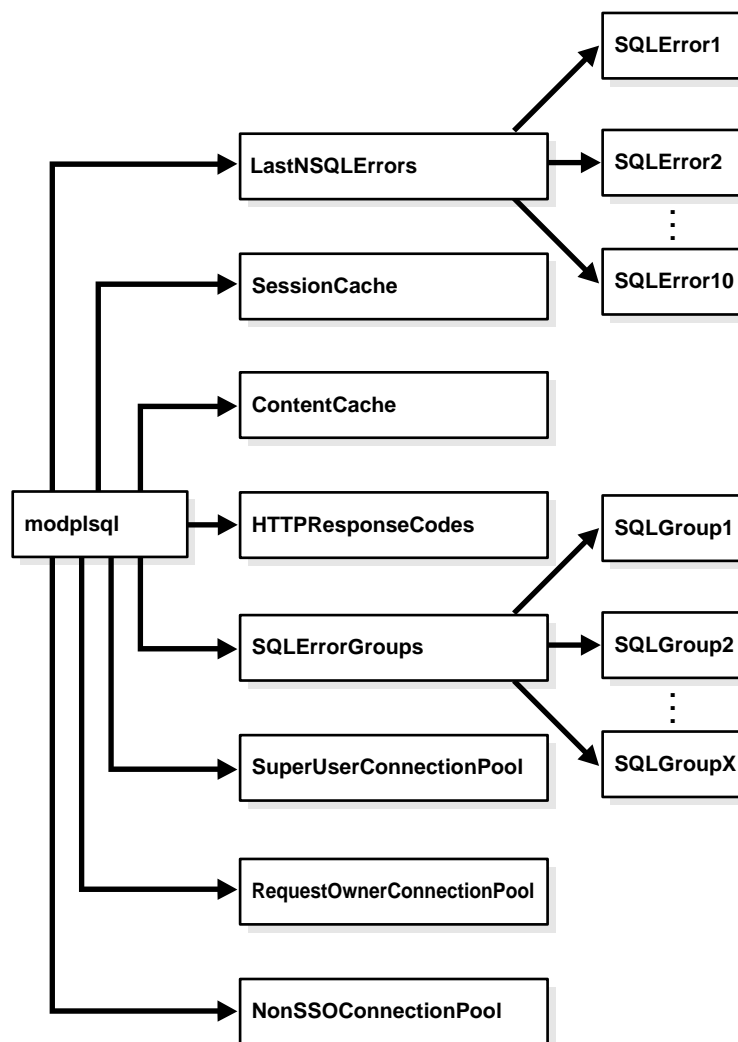
Metric	Description	Unit
<code>interval.value</code>	Shows how often the task should run. The task manager executes all the tasks in a round-robin fashion. If the interval is zero, then the task manager executes the task when it is selected in the round robin.	msecs (Milliseconds)
<code>run().active</code>	Number of active threads.	threads
<code>run().avg</code>	Average time for the taskmanager to run the task	msecs
<code>run().completed</code>	Number of times the taskmanager has run the task.	ops
<code>run().maxActive</code>	Maximum number of active tasks.	threads
<code>run().maxTime</code>	Maximum time for the task to run.	msecs
<code>run().minTime</code>	Minimum time for the task to run.	msecs
<code>run().time</code>	Total time spent running the task manager	msecs

C.7 mod_plsql Metrics

This section describes the Oracle Application Server mod_plsql metrics.

[Figure C–1, "mod_plsql Metric Tree"](#) shows the structure of the mod_plsql metrics. The tables in this section describe the relevant metrics.

Figure C–1 mod_plsql Metric Tree



The `/modplsql/HTTPResponseCodes` Metrics lists the response codes returned by `mod_plsql`.

The metric table name is `modplsql_HTTPResponseCodes`. This metric table includes one metric containing the count, number of times the response was generated, for each HTTP response type.

```
[type=modplsql_HTTPResponseCodes]
```

For example, the `http404.count` metric holds a count of the "HTTP 404: Not found" response codes.

[Table C–43](#) lists the set of metrics for the `mod_plsql` session cache.

The metric table name is `modplsql_Cache`.

Table C–43 mod_plsql/SessionCache Metrics

Metric	Description	Unit
<code>cacheStatus.value</code>	Status of the cache. This can be either enabled or disabled.	status
<code>newMisses.count</code>	Number of session cache misses (new)	ops

Table C–43 (Cont.) mod_plsql/SessionCache Metrics

Metric	Description	Unit
staleMisses.count	Number of session cache misses (stale)	ops
hits.count	Number of session cache hits	ops
requests.count	Number of requests to the session cache	ops

[Table C–44](#) lists the set of metrics for the mod_plsql content cache.

The metric table name is modplsql_ContentCache.

Table C–44 mod_plsql/ContentCache Metrics

Metric	Description	Unit
cacheStatus.value	Status of the cache, either enabled or disabled.	
newMisses.count	Number of content cache misses (new)	ops
staleMisses.count	Number of content cache misses (stale)	ops
hits.count	Number of content cache hits	ops
requests.count	Number of requests to the content cache	ops

The `SQLErrorGroups` metrics show the predefined groupings of SQL errors. For each group, the metrics in [Table C–45](#) are recorded.

The metric table name is modplsql_SQLErrorGroup:

```
/modplsql/SQLErrorGroups/group [type=modplsql_SQLErrorGroup]
```

The *group* is based on the groupings in the Oracle Database Error Messages guide. For example, the metric name `Ora24280Ora29249` represents the grouping Ora-24280 to Ora-29249. Each SQL error that occurs as a result of executing a request is put into the appropriate group based on its error code. If you are getting a high number of the same errors, you should investigate what is causing the problem, using the Oracle Database Error Messages guide for further details on the error message.

Table C–45 mod_plsql/SQLErrorGroups Metrics

Metric	Description	Unit
lastErrorDate.value	Date of the last request to cause the SQL error	date
lastErrorRequest.value	Last request to cause the SQL error	url
lastErrorText.value	SQL error text of the last error	error
error.count	Number of errors that have occurred within the group	ops

The `LastNSQLErrors` statistics show the last 10 SQL errors that have occurred while executing requests. These are updated in a round robin fashion. For each error, the metrics in [Table C–46](#) are recorded.

The metric table name is modplsql_LastNSQLError:

```
/modplsql/LastNSQLErrors/<SQL Error Slot> [type=modplsql_LastNSQLError]
```

If you are getting a large number of the same errors, you should investigate what is causing the problem. Refer to the Oracle Database Error Messages guide for further details of the error represented by the `errorText.value` metric.

Table C–46 *mod_plsql/LastNSQLErrors Metrics*

Metric	Description	Unit
errorDate.value	Date the request caused the SQL error	date
errorRequest.value	Request causing the SQL error	url
errorText.value	SQL error text	error

[Table C–47](#) lists the set of metrics for the Non-SSO connection pool.

The metric table name is modplsql_DatabaseConnectionPool:

```
/modplsql/NonSSOConnectionPool [type=modplsql_DatabaseConnectionPool]
```

Table C–47 *mod_plsql/NonSSOConnectionPool Metrics*

Metric	Description	Unit
connFetch.maxTime	Maximum time to fetch a connection from the pool	usecs
connFetch.minTime	Minimum time to fetch a connection from the pool	usecs
connFetch.avg	Average time to fetch a connection from the pool	usecs
connFetch.active	Child servers currently in the pool fetch phase	threads
connFetch.time	Total time spent fetching connections from the pool	usecs
connFetch.completed	Number of times a connection has been requested from the pool	ops
newMisses.count	Number of connection pool misses (new)	ops
staleMisses.count	Number of connection pool misses (stale)	ops
hits.count	Number of connection pool hits	ops

[Table C–48](#) lists the set of metrics for the request owner connection pool.

The metric table name is modplsql_DatabaseConnectionPool:

```
/modplsql/RequestOwnerConnectionPool [type=modplsql_DatabaseConnectionPool]
```

Table C–48 *mod_plsql/RequestOwnerConnectionPool Metrics*

Metric	Description	Unit
connFetch.maxTime	Maximum time to fetch a connection from the pool	usecs
connFetch.minTime	Minimum time to fetch a connection from the pool	usecs
connFetch.avg	Average time to fetch a connection from the pool	usecs
connFetch.active	Child servers currently in the pool fetch phase	threads
connFetch.time	Total time spent fetching connections from the pool	usecs
connFetch.completed	Number of times a connection has been requested from the pool	ops
newMisses.count	Number of connection pool misses (new)	ops
staleMisses.count	Number of connection pool misses (stale)	ops
hits.count	Number of connection pool hits	ops

[Table C–49](#) lists the set of metrics for the super user connection pool.

The metric table name is modplsql_DatabaseConnectionPool:

```
/modplsql/SuperUserConnectionPool [type=modplsql_DatabaseConnectionPool]
```

Table C–49 *mod_plsql/SuperUserConnectionPool Metrics*

Metric	Description	Unit
<code>connFetch.maxTime</code>	Maximum time to fetch a connection from the pool	usecs
<code>connFetch.minTime</code>	Minimum time to fetch a connection from the pool	usecs
<code>connFetch.avg</code>	Average time to fetch a connection from the pool	usecs
<code>connFetch.active</code>	Threads currently in the pool fetch phase	threads
<code>connFetch.time</code>	Total time spent fetching connections from the pool	usecs
<code>connFetch.completed</code>	Number of times a connection has been requested from the pool	ops
<code>newMisses.count</code>	Number of connection pool misses (new)	ops
<code>staleMisses.count</code>	Number of connection pool misses (stale)	ops
<code>hits.count</code>	Number of connection pool hits	ops

C.8 Oracle Process Manager and Notification Server - OPMN Metrics

This sections lists the Oracle Process Manager and Notification Server (opmn) metrics.

This section includes the following:

- [OPMN_PM Metric Table](#)
- [OPMN_OC4J_PROC Table](#)
- [OPMN_HOST_STATISTICS Metric Table](#)
- [OPMN_IAS_INSTANCE Metric Table](#)
- [OPMN_IAS_COMPONENT Table](#)
- [OPMN ONS Metrics](#)
- [OPMN_APPCTX Table](#)

C.8.1 OPMN_PM Metric Table

The `opmn_pm` metric table is the root of the process manager subtree for the OPMN DMS metrics. The metrics in this metric table contain statistics about OPMN requests. An OPMN request is a command that has been issued to OPMN from a client, for example DCM, to perform an operation on one or more OPMN managed processes.

Requests can have one of three possible results:

- Success – success means OPMN handles the request successfully.
- Partial Success – partial Success means OPMN only handles part of the request successfully. For example, if a client wants OPMN to start three OC4J processes, and only two are successfully started, the request result is partial success.
- Failure – failure means the request fails.

[Table C–50](#) shows the metric table type `opmn_pm`.

Table C–50 OPMN_PM Metrics

Metric	Description	Unit
jobWorkerQueue.value	Specifies the number of jobs in the OPMN worker queue	ops
lReq.count	Specifies the number of local HTTP requests which OPMN handles	ops
procDeath.count	Specifies the number of processes which die after the process manager starts them	ops
procDeathReplace.count	Specifies the number of processes which are restarted after the process manager detects they are dead	ops
reqFail.count	Specifies the number of HTTP requests which fail	ops
reqPartialSucc.count	Specifies the number of HTTP requests which partially succeed	ops
reqSucc.count	Specifies the number of HTTP requests which succeed	ops
rReq.count	Specifies the number of remote HTTP requests which OPMN handles	ops
workerThread.value	Specifies the number of worker threads	threads

C.8.2 OPMN_OC4J_PROC Table

The OPMN OC4J proc metrics table provides information on the OC4J process.

Table C–51 OPMN_OC4J_proc Metrics

Metric	Description	Unit
oc4jinstance.value		
oc4jIsland.value		

C.8.3 OPMN_HOST_STATISTICS Metric Table

The OPMN host statistics metric table provides information on the host running the OPMN process.

[Table C–52](#) shows the metric table type `opmn_host_statistics`.

Table C–52 OPMN_HOST_STATISTICS Metrics

Metric	Description	Unit
cpuIdle.value	Specifies the number of milliseconds the cpu(s) have been idle since an unspecified time.	milliseconds
freePhysicalMem.value	Specifies the amount of free physical memory on the host machine.	kilobytes
numProcessors.value	Specifies the number of processors available on the host machine.	integer
timestamp.value	Specifies the time that host statistics are taken. The timestamp is the number of milliseconds from an unspecified time.	milliseconds from an unspecified time
totalPhysicalMem.value	Specifies the total physical memory available on the host machine.	kilobytes

C.8.4 OPMN_IAS_INSTANCE Metric Table

The OPMN IAS instance subtree shows the Oracle Application Server instance node name.

[Table C–53](#) shows the metric table type `opmn_ias_instance`.

Table C–53 OPMN_IAS_INSTANCE Metrics

Metric	Description	Unit
iasCluster.value	Specifies the Oracle Application Server cluster name for the Oracle Application Server instance.	String

C.8.5 OPMN_IAS_COMPONENT Table

The OPMN IAS component subtree represents an Oracle Application Server component. The OPMN IAS component subtree includes several metric tables containing component information.

[Table C–54](#) shows the metric table type `opmn_process_type`.

Table C–54 OPMN_PROCESS_TYPE Metrics

Metric	Description	Unit
moduleId.value	Specifies the values of attribute module-IDs, as specified in the process-type tag in the <code>opmn.xml</code> configuration file.	String

[Table C–55](#) shows the metric table type `opmn_process_set`.

Table C–55 OPMN_PROCESS_SET Metrics

Metric	Description	Unit
numProcConf.value	Specifies the number, or maximum number, of processes configured for this process set.	String (integer)
numProcs.value	Number of process that exist for this process set	
IsService.value	Process set is configured as a service	String
reqFail.count	Specifies the number of HTTP requests which fail for this process set.	ops
reqPartialSucc.count	Specifies the number of HTTP requests which partially succeed for this process set.	ops
reqSucc.count	Specifies the number of HTTP requests which succeed for this process set	ops
restartOnDeath.value	Specifies whether, when a process dies, OPMN should restart the process.	String (boolean)

[Table C–56](#) shows the metric table type `opmn_process`.

Table C–56 OPMN_PROCESS Metrics

Metric	Description	Unit
cpuTime.value	Shows the amount of CPU time used by the process.	CPU msecs
heapSize.value	Shows the heap size of the process.	Kilobytes
iasCluster.value	Shows the Oracle Application Server cluster name for the process	String
iasInstance.value	Shows the Oracle Application Server instance name for the process	String
indexInSet.value	Shows the process index in the process set. This value is only valid for OPMN managed processes, for OPMN unmanaged processes, this value has no meaning, and the value is always 0.	String (integer)

Table C–56 (Cont.) OPMN_PROCESS Metrics

Metric	Description	Unit
memoryUsed.value	<p>The amount of memory used by the process.</p> <p>This metric is calculated in an operating system specific manner.</p> <p>On UNIX, this is the process image memory used value. This is all the memory in use by the process.</p> <p>On Windows, this is the working set memory used value. This is the same value that is reported by the Task Manager under the mem usage column. The working set is the set of memory pages touched recently by the threads in the process. If free memory in the system is over a certain threshold, pages are left in the working set of a process, even if they are not in use. When free memory falls below a certain threshold, pages are trimmed from the working sets. If needed, pages are soft-faulted back into the working set before they leave main memory.</p>	
pid.value	The process ID for the process.	
privateMemory.value	The private memory of the process.	Kilobytes
sharedMemory.value	The shared memory for the process	Kilobytes
startTime.value	The start time of the process.	msecs
status.value	<p>The status of the process. The status can have the following values:</p> <ul style="list-style-type: none"> ■ NONE – New process slot, no operations have been applied yet (no status). ■ Init – process has been started, opmn is waiting for initialization to complete. ■ Alive – process is fully started. ■ Stop – process stop operation is in progress. ■ Stopped – process has been fully stopped. ■ Bounce – non-terminating process restart is in progress. ■ Restart – process stop operation is in progress, prior to a new start being issued. ■ InitFail – failure before init timeout reached, a stop and start will be attempted in the retry limit has not been reached. ■ BounceFail – non-terminating process restart failed, as stop and start will be attempted if the retry limit has not been reached. 	String
type.value	The type of the process. See Table C–54 for information on process types.	
uid.value	The OPMN assigned ID for the process.	
upTime.value	The uptime for the process.	msecs

[Table C–57](#) shows the metric table type opmn_connect.

Table C–57 OPMN_CONNECT Metrics

Metric	Description	Unit
desc.value	Shows the port description, if available	String
host.value	Shows the host name	String (host name)
protocol.value		
port.value	Shows the port number	String (port number)

C.8.6 OPMN ONS Metrics

The Oracle Process Manager and Notification Server ONS subtree contains Oracle Notification System (ONS) information.

[Table C–58](#) shows the metric table type opmn_ons.

Table C–58 OPMN_ONS Metrics

Metric	Description	Unit
notifProcessed.value	The number of notifications processed by ONS.	ops
notifProcessQueue.value	The number of notifications in the process queue.	ops
notifReceived.value	The number of notifications received by ONS.	ops
notifReceiveQueue.value	The number of notifications in the receive queue.	ops
workerThread.value	The number of worker threads.	String (threads)

[Table C–59](#) shows the local_port metrics. The `.. /ons/local_port` subtree shows information about the ONS local port.

The metric table type is `opmn_connect`

Table C–59 OPMN ONS LOCAL_PORT Metrics

Metric	Description	Unit
desc.value	Port description	String
host.value	Host name	String
port.value	Port number	String

[Table C–60](#) shows the remote_port metrics. The `.. /ons/remote_port` subtree shows information about the ONS remote port.

The metric table type is `opmn_connect`

Table C–60 OPMN ONS REMOTE_PORT Metrics

Metric	Description	Unit
desc.value	Port description	String
host.value	Host name	String
port.value	Port number	String

[Table C–61](#) shows the request_port metrics. The `.. /ons/request_port` subtree shows information about the ONS request port.

The metric table type is `opmn_connect`

Table C–61 OPMN ONS REQUEST_PORT Metrics

Metric	Description	Unit
desc.value	Port description	String
host.value	Host name	String
port.value	Port number	String

[Table C–62](#) shows the `opmn_ons_topo_entry` metrics.

Table C–62 OPMN ONS TOPO Entry Metrics

Metric	Description	Unit
protocol.value	ons protocol version	
port.value	port value	
ip.value	ip address	

C.8.7 OPMN_APPCTX Table

[Table C–63](#) shows the opmn_appctx metrics.

Table C–63 OPMN APPCTX Metrics

Metric	Description	Unit
rtid.value		
routable.value		
state.value		

C.9 DMS Internal Metrics

Table C–64 DMS-Internal Clock Metrics

Metric	Description	Unit
logicalTime.value	The current time as measured with the DMS clock.	ticks
measuredFrequency.value	Number of clock ticks per second - measured.	ticks
measuredResolution.value	Time between ticks as measured with this clock.	
name.value		
overheadPerCall.value	The average duration of a call to get the time with this clock.	
reportedFrequency.value	The number of ticks per second the clock time is reported in.	ticks
requestedUnits.value	The string description of the units that times are reported in.	

Table C–65 DMS-Internal Log Metrics

Metric	Description	Unit
initLogging.count		ops
messagesLogged.count		ops
status.value		

Table C–66 DMS-Internal Measurement Metrics

Metric	Description	Unit
createNoun.count		ops
createSensor.count		ops
destroyNoun.count		ops
destroySensor.count		ops
lastTreeNodeID.value		
sampleMetric.count		ops

Table C–66 (Cont.) DMS-Internal Measurement Metrics

Metric	Description	Unit
sensorWeight.value		
treeNodes.maxValue		
treeNodes.value		

Table C–67 DMS-Internal Collector Metrics

Metric	Description	Unit
logger.count		ops
logger.logged		ops
responseGenerateTime.active		threads
responseGenerateTime.avg		
responseGenerateTime.completed		
responseGenerateTime.maxActive		
responseGenerateTime.maxTime		
responseGenerateTime.minTime		
responseGenerateTime.time		

Table C–68 DMS-Internal Transtrace Metrics

Metric	Description	Unit
expireMessages.avg		
expireMessages.completed		
expireMessages.maxActive		
expireMessages.maxTime		
expireMessages.minTime		
expireMessages.time		
messageCount.value		
pendingMessageCount.value		
s_debugEnabled.value		
s_dumpEnabled.value		
s_ecidEnabled.value		
s_transTraceEnabled.value		
storeSize.value		

A

access logging, 5-3
AggreSpy
 access control, A-4
 performance monitoring, A-2
 URL, A-4
 using, A-2
 using with standalone OC4J, A-10
Application Server Control
 monitoring Oracle Application Server with, 2-2

B

built-in performance metrics, 2-2

C

capacity, 1-6
com.evermind.server.ejb.TimeoutExpiredException
 from EJB, 3-16
concurrency
 defined, 1-2
 limiting, 1-6
contention, 1-4
 defined, 1-2
CPUs
 insufficient, 1-4

D

data-source
 stmt-cache-size attribute, 3-8
datasources
 inactivity-timeout option, 3-6
 initial-limit option, 3-6
 max-connections attribute, 3-15
 min-connections option, 3-6, 3-15
 num-cached-statements attribute, 3-8
directives
 See also httpd.conf directives
DMS
 coding tips, B-16
 conditional instrumentation, B-15
 Event sensors, B-4
 using, B-10
 getSensorWeight, B-15

instrumentation
 definition of, B-2
 using, B-8
metrics
 definition of, B-4
 dumping to files, B-15
monitoring metrics, B-2
naming conventions, B-6
nouns, B-3, B-5
 naming conventions, B-7
 using, B-9
PhaseEvent sensors, B-4
 using, B-10
rollup
 descendents, B-24
 refresh, B-24
 rolled, B-24
rollup noun, B-6
security, B-15
sensors, B-3
 definition of, B-4
 destroying, B-16
 resetting, B-16
State sensors, B-4
 using, B-11
terminology, B-3
testing metrics, B-14
validating metrics, B-13
dmstool
 access control, A-6
 address option, A-7, A-10
 count option, A-7
 dump option, A-7, A-9
 format=xml option, A-7
 interval option, A-7
 list option, A-8
 options, A-6
 reset option, A-8
 table option, A-8
 using, A-6
DNS
 domain name server, 5-3

E

EJBs

- metrics, C-13
- ErrorLog
 - directive, 5-4
- Event sensors, B-4

F

- functional demand, 1-6

G

- garbage collection options, 3-4
- global-thread-pool element, 3-13

H

- hash
 - defined, 1-2
- heap size
 - setting, 3-3
- HostNameLookups
 - directive, 5-3
- httpd.conf
 - directives
 - ErrorLog, 5-4
 - HostNameLookups, 5-3
 - KeepAlive, 5-3
 - KeepAliveTimeout, 5-3
 - ListenBacklog, 5-2
 - LogLevel, 5-4
 - MaxClients, 3-7, 5-2
 - MaxKeepAliveRequests, 5-3
 - MaxRequestsPerChild, 5-2
 - MaxSpareServers, 5-2
 - MinSpareServers, 5-2
 - StartServers, 5-2
 - ThreadsPerChild, 3-8
 - Timeout, 5-2

I

- inactivity-timeout attribute, 3-6
- INFO logging level, 3-10
- initial-limit attribute, 3-6

J

- J2EE
 - metrics, C-10
- Java option
 - XX+AggressiveHeap, 3-5
 - XX+DisableExplicitGC, 3-5
- Java options
 - Xms, 3-4
 - Xmx, 3-4
- JDBC
 - metrics, C-7
 - stmt-cache-size, 3-8
- JSPs
 - metrics, C-12
- JVM

- metrics, C-6
 - setting heap size, 3-3
- JVM garbage collection, 3-4
- JVM metrics
 - Properties metrics, C-6
- JVM system properties metrics, C-6

K

- KeepAlive directive, 3-7
- KeepAlive httpd.conf directive, 5-3
- KeepAliveTimeout httpd.conf directive, 5-3

L

- latency
 - defined, 1-2
- ListenBacklog httpd.conf directive, 5-2
- load variances, 1-7
- logging
 - access, 5-3
 - error, 5-4
 - performance and, 5-3
 - performance implications of, 5-3
- logging levels
 - setting, 3-10
- LogLevel directive, 5-4
- logresolve utility, 5-3

M

- MaxClients directive, 3-7
- MaxClients httpd.conf directive, 3-7, 5-2
- max-connections attribute, 3-15
- MaxKeepAliveRequests directive, 3-7
- MaxKeepAliveRequests httpd.conf directive, 5-3
- MaxRequestsPerChild httpd.conf directive, 5-2
- MaxSpareServers httpd.conf directive, 5-2
- memory
 - JVM heap size, 3-3
- metric table types
 - JDBC_Connection, C-8
 - JDBC_ConnectionSource, C-9
 - JDBC_DataSource, C-7
 - JDBC_Driver, C-7
 - JDBC_Statement, C-9
 - JMSBrowserStats, C-21
 - JMSConnectionStats, C-19
 - JMSDestinationStats, C-22
 - JMSDurableSubscriptionStats, C-21
 - JMSMessageConsumerStats, C-21
 - JMSPersistenceStats, C-23
 - JMSProducerStats, C-20
 - JMSRequestHandlerStats, C-19
 - JMSSessionStats, C-20
 - JMSStats, C-17
 - JMSStoreStats, C-22
 - JMSTemporaryDestinationStats, C-22
- JVM, C-6
 - mod_oc4j_destination_metrics, C-5
 - mod_oc4j_mount_pt_metrics, C-4

- mod_oc4j_request_failure_causes, C-4
- modplsql_Cache, C-25, C-26
- modplsql_DatabaseConnectionPool, C-27
- modplsql_HTTPResponseCodes, C-3, C-25
- modplsql_LastNSQLError, C-26
- modplsql_SQLErrorGroup, C-26
- oc4j_context, C-11
- oc4j_ejb_entity_bean, C-14
- oc4j_ejb_method, C-14
- oc4j_jsp(threadsafe=false), C-13
- oc4j_jsp(threadsafe=true), C-13
- oc4j_jspExec, C-13
- oc4j_opmn, C-15
- oc4j_servlet, C-12
- oc4j_task, C-23
- oc4j_web_module, C-11
- ohs_module, C-3
- ohs_server, C-2
- opmn_connect, C-31, C-32
- opmn_host_statistics, C-29
- opmn_ias_instance, C-29
- opmn_ons, C-31
- opmn_pm, C-28
- opmn_process, C-30
- opmn_process_set, C-30
- opmn_process_type, C-30
- metric tables, A-3
- metrics
 - acknowledgeMode.value, C-20
 - activeInstances.value, C-13
 - activeThreadGroups.maxValue, C-6
 - activeThreadGroups.minValue, C-6
 - activeThreadGroups.value, C-6
 - activeThreads.maxValue, C-6
 - activeThreads.minValue, C-6
 - activeThreads.value, C-6
 - address.value, C-17, C-19
 - availableInstances.value, C-13
 - bean-type.value, C-14
 - busyChildren.value, C-2
 - CacheFreeSize.value, C-9
 - CacheGetConnection.avg, C-9
 - CacheHit.count, C-9
 - CacheMiss.count, C-9
 - CacheSize.value, C-9
 - cacheStatus.value, C-25, C-26
 - childFinish.count, C-2
 - childStart.count, C-2
 - client.active, C-14
 - client.avg, C-14
 - client.completed, C-14
 - clientID.value, C-19, C-22
 - client.maxTime, C-14
 - client.minTime, C-14
 - client.time, C-14
 - connection.active, C-2
 - connection.avg, C-2
 - ConnectionCloseCount.count, C-7, C-8
 - connection.completed, C-2
 - ConnectionCreate.active, C-7, C-8
 - ConnectionCreate.avg, C-7
 - ConnectionCreate.completed, C-7
 - ConnectionCreate.maxTime, C-7
 - ConnectionCreate.minTime, C-7
 - ConnectionCreate.time, C-7
 - connectionID.value, C-19, C-22
 - connection.maxTime, C-2
 - connection.minTime, C-2
 - ConnectionOpenCount.count, C-7, C-8
 - connection.time, C-2
 - connFetch.active, C-27, C-28
 - connFetch.avg, C-27, C-28
 - connFetch.completed, C-27, C-28
 - connFetch.maxTime, C-27, C-28
 - connFetch.minTime, C-27, C-28
 - connFetch.time, C-27, C-28
 - cpuIdle.value, C-29
 - cpuTime.value, C-30
 - CreateNewStatement.avg, C-8
 - CreateStatement.avg, C-8, C-9
 - default_application_log.value, C-15
 - deliveryMode.value, C-20
 - desc.value, C-32
 - Destination.value, C-4
 - destination.value, C-20, C-21, C-22, C-23
 - disableMessageID.value, C-20
 - disableMessageTimestamp.value, C-20
 - domain.value, C-19, C-20, C-21, C-22
 - EJB, C-13
 - ejbPostCreate.active, C-14
 - ejbPostCreate.avg, C-14
 - ejbPostCreate.completed, C-14
 - ejbPostCreate.maxTime, C-14
 - ejbPostCreate.minTime, C-14
 - ejbPostCreate.time, C-14
 - error.count, C-2, C-26
 - errorDate.value, C-27
 - errorRequest.value, C-27
 - errorText.value, C-27
 - ErrReq.count, C-4, C-5
 - ErrReqNonSess.count, C-4, C-5
 - ErrReqSess.count, C-4, C-5
 - exceptionListener.value, C-19
 - exclusive-write-access.value, C-14
 - Execute.time, C-10
 - Failover.count, C-4, C-5
 - Fetch.time, C-10
 - freeMemory.maxValue, C-6
 - freeMemory.minValue, C-6
 - freeMemory.value, C-6
 - freePhysicalMem.value, C-29
 - get.count, C-2
 - handle.active, C-2, C-4
 - handle.avg, C-2, C-4
 - handle.completed, C-2, C-4
 - handle.maxTime, C-2, C-4
 - handle.minTime, C-2, C-4
 - handle.time, C-2, C-4
 - heapSize.value, C-30
 - hits.count, C-26, C-27, C-28

holePageCount.value, C-23
 host.value, C-18, C-19, C-32
 ias_cluster.value, C-15
 ias_instance.value, C-15
 iasCluster.value, C-30
 iasInstance.value, C-30
 IncorrectReqInit.count, C-4
 indexInSet.value, C-30
 internalRedirect.count, C-2
 interval.value, C-24
 isActive.value, C-22
 isLocal.value, C-20
 isolation.value, C-14
 isOpen.value, C-23
 isXA.value, C-20
 J2EE, C-10
 JDBC_Connection_URL, C-8
 JDBC_Connection_Url, C-9
 JDBC_Connection_Username, C-8, C-9
 jms_log.value, C-15
 jobWorkerQueue.value, C-29
 JSP, C-12
 JVM, C-6
 JVMCnt.value, C-5
 lastConfigChange.value, C-2
 lastErrorDate.value, C-26
 lastErrorRequest.value, C-26
 lastErrorText.value, C-26
 lastUsed.value, C-23
 locations.value, C-22
 LogicalConnection.value, C-8, C-9
 lReq.count, C-29
 memoryUsed.value, C-31
 messageCount.value, C-22
 messageDequeued.count, C-22
 messageDiscarded.count, C-22
 messageEnqueued.count, C-22
 messageExpired.count, C-23
 messageListener.value, C-21
 messagePagedIn.count, C-23
 messagePagedOut.count, C-23
 messageRecovered.count, C-23
 mod_plsql, C-24
 moduleId.value, C-30
 Name.value, C-4, C-5
 name.value, C-21, C-22
 newMisses.count, C-25, C-26, C-27, C-28
 noLocal.value, C-21, C-22
 NonSessFailover.count, C-5
 notifProcessed.value, C-32
 notifProcessQueue.value, C-32
 notifReceived.value, C-32
 numChildren.value, C-2
 numMods.value, C-2, C-3
 numProcConf.value, C-30
 numProcessors.value, C-29
 oc4j_instance.value, C-15
 oc4j_island.value, C-15
 oc4j.jms.debug.value, C-18
 oc4j.jms.forceRecovery.value, C-18
 oc4j.jms.listenerAttempts.value, C-18
 oc4j.jms.maxOpenFiles.value, C-18
 oc4j.jms.messagePoll.value, C-18
 oc4j.jms.noDms.value, C-18
 oc4j.jms.saveAllExpired.value, C-18
 oc4j.jms.serverPoll.value, C-18
 oc4j.jms.socketBufsize.value, C-18
 Oc4jUnavailable.count, C-4
 opmn_group.value, C-15
 opmn_sequence.value, C-15
 Oracle Application Server performance, C-1
 parseRequest.active, C-11
 parseRequest.avg, C-11
 parseRequest.completed, C-11
 parseRequest.maxTime, C-11
 parseRequest.minTime, C-11
 parseRequest.time, C-11
 pendingMessageCount.value, C-23
 persistenceFile.value, C-23
 persistence-type.value, C-14
 pid.value, C-31
 port.value, C-19, C-20, C-32
 post.count, C-2
 priority.value, C-21
 privateMemory.value, C-31
 procDeath.count, C-29
 procDeathReplace.count, C-29
 processRequest.active, C-11, C-13
 processRequest.avg, C-11, C-13
 processRequest.completed, C-11, C-13
 processRequest.maxTime, C-11, C-13
 processRequest.minTime, C-11, C-13
 processRequest.time, C-11, C-13
 readyChildren.value, C-2
 reqFail.count, C-29, C-30
 reqPartialSucc.count, C-29, C-30
 reqSucc.count, C-29, C-30
 request.active, C-2
 request.avg, C-2
 request.completed, C-2
 requestHandlers.count, C-19
 request.maxTime, C-2
 request.minTime, C-2
 requests.count, C-26
 request.time, C-2
 resolveContext.active, C-11
 resolveContext.avg, C-11
 resolveContext.completed, C-11
 resolveContext.maxTime, C-11
 resolveContext.minTime, C-11
 resolveContext.time, C-11
 resolveServlet.avg, C-12
 resolveServlet.completed, C-12
 resolveServlet.maxTime, C-12
 resolveServlet.minTime, C-12
 resolveServlet.time, C-12
 responseSize.value, C-2
 restartOnDeath.value, C-30
 rmi_log.value, C-15
 rReq.count, C-29

- run().active, C-24
- run().avg, C-24
- run().completed, C-24
- run().maxActive, C-24
- run().maxTime, C-24
- run().minTime, C-24
- run().time, C-24
- selector.value, C-21, C-22
- server_log.value, C-15
- service.active, C-12, C-13
- service.avg, C-12, C-13
- service.completed, C-12, C-13
- service.maxTime, C-12, C-13
- service.minTime, C-12, C-13
- service.time, C-12, C-13
- SessFailover.count, C-5
- sessionActivation.avg, C-12
- sessionActivation.completed, C-12
- sessionActivation.maxTime, C-12
- sessionActivation.minTime, C-12
- sessionActivation.time, C-12
- sessionListener.value, C-20
- session-type.value, C-14
- sharedMemory.value, C-31
- SQLText.value, C-10
- staleMisses.count, C-26, C-27, C-28
- startTime.value, C-19, C-20, C-21
 - opmn_process, C-31
- StatementCacheHit.count, C-8
- StatementCacheMiss.count, C-8, C-9
- status.value, C-31
- storeSize.value, C-23
- SucReq.count, C-5
- SucReqNonSess.count, C-5
- SucReqSess.count, C-5
- taskManagerInterval.value, C-19
- timestamp.value, C-29
- timeToLive.value, C-21
- totalMemory.maxValue, C-6
- totalMemory.minValue, C-6
- totalMemory.value, C-6
- totalPhysicalMem.value, C-29
- transacted.value, C-20
- transaction-type.value, C-14
- trans-attribute.value, C-14
- txid.value, C-20
- type.value, C-31
- uid.value, C-31
- UnableToHandleReq.count, C-4
- upTime.value, C-6, C-31
- usedPageCount.value, C-23
- user.value, C-20
- workerThread.value, C-29, C-32
- wrapper.active, C-15
- wrapper.avg, C-15
- wrapper.completed, C-15
- wrapper.maxTime, C-15
- wrapper.minTime, C-15
- wrapper.time, C-15
- xid.value, C-20

- min-connections attribute, 3-6
- min-connections datasources option, 3-15
- min-instances attribute, 3-11
- MinSpareServers httpd.conf directive, 5-2
- mod_plsql metrics, C-24
- modplsql_Cache
 - metric table type, C-25, C-26
- modplsql_DatabaseConnectionPool
 - metric table type, C-27
- modplsql_HTTPResponseCodes
 - metric table type, C-3, C-25
- modplsql_LastNSQLError
 - metric table type, C-26
- modplsql_SQLErrorGroup
 - metric table type, C-26
- monitoring
 - performance statistics, 2-2

N

- naming conventions
 - DMS, B-6
- noun
 - rollup, B-6
- nouns
 - creating, B-9
 - DMS, B-5
 - naming conventions, B-7
 - type, B-5
- num-cached-statements attribute, 3-8

O

- OC4J
 - monitoring performance statistics, 2-2
- OC4J thread pool
 - global-thread-pool element, 3-13
- oc4j_context
 - metric table type, C-11
- oc4j_ejb_entity_bean
 - metric table type, C-14
- oc4j_ejb_method
 - metric table type, C-14
- oc4j_jspExec
 - metric table type, C-13
- oc4j_servlet
 - metric table type, C-12
- oc4j_web_module
 - metric table type, C-11
- Oc4jCacheSize, 3-8
- opmn.xml file
 - setting Java options, 3-4
- Oracle HTTP Server
 - configuring with directives, 5-1
- oracle.j2ee.rmi.loadBalance property, 3-19

P

- parameters
 - KeepAlive, 5-3
 - KeepAliveTimeout, 5-3

- ListenBacklog, 5-2
- MaxClients, 3-7, 5-2
- MaxKeepAliveRequests, 5-3
- MaxRequestsPerChild, 5-2
- MaxSpareServers, 5-2
- MinSpareServers, 5-2
- Oc4jCacheSize, 3-8
- StartServers, 5-2
- ThreadsPerChild, 3-8
- Timeout, 5-2
- performance
 - goals, 1-6
 - monitoring
 - native operating system, 2-3
 - network monitoring tools, 2-3
- persistent connections
 - KeepAlive directive, 5-3
- PhaseEvent sensors, B-4
- pool-cache-timeout attribute, 3-11

R

- response time, 1-4
 - defined, 1-2
 - goal, 1-6
 - improving, 1-3
 - peak load, 1-7
- rollup
 - DMS, B-21

S

- scalability
 - defined, 1-2
- service time, 1-3, 1-4
 - defined, 1-2
- StartServers httpd.conf directive, 5-2
- State sensors, B-4
- system properties
 - DMS metrics, C-6

T

- think time
 - defined, 1-2
- thread pool options, 3-12
- ThreadsPerChild, 3-8
- ThreadsPerChild directive, 3-8
- throughput
 - defined, 1-2
 - demand limiter and, 1-5
 - increasing, 1-4
- Timeout httpd.conf directive, 5-2
- TimeoutExpiredException
 - from EJB, 3-16

U

- unit consumption, 1-6

W

- wait time
 - contention and, 1-4
 - defined, 1-2
 - parallel processing and, 1-3
- Windows
 - performance counters, A-11
- Windows metrics
 - enabling Performance Counters, A-11
- work-manager-thread-pool, 3-15

X

- xml format output for dmstool, A-7
- Xms option, 3-3
- XX+AggressiveHeap option, 3-5
- XX+DisableExplicitGC option, 3-5
- XXAppendRatio option, 3-19