February 2003
This book provides detailed information about optimizing configuration model designs and setting up the Web server for optimal performance at runtime, as well as Configurator-specific setup parameters for LoadRunner and case studies that illustrate the Configurator data relevant to performance testing.
# Contents

<table>
<thead>
<tr>
<th>Send Us Your Comments</th>
<th>xv</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Preface</strong></td>
<td>xvii</td>
</tr>
<tr>
<td>Intended Audience</td>
<td>xvii</td>
</tr>
<tr>
<td>Documentation Accessibility</td>
<td>xviii</td>
</tr>
<tr>
<td>Structure</td>
<td>xviii</td>
</tr>
<tr>
<td>Related Documents</td>
<td>xx</td>
</tr>
<tr>
<td>Conventions</td>
<td>xxi</td>
</tr>
<tr>
<td>Product Support</td>
<td>xxi</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Scope of this Guide ................................................................. 1-1
1.2 Relevant Terms ....................................................................... 1-2

## 2 Tuning

2.1 Fundamentals of Performance Tuning......................................... 2-1
2.1.1 Methodology .................................................................... 2-2
2.1.2 Documenting Results.......................................................... 2-3
2.1.3 Defining Success Criteria .................................................... 2-3
2.2 The Tuning Process .................................................................. 2-4
2.2.1 Steps in the Process ............................................................. 2-4
2.3 Tuning Key Components of the Oracle Configurator Architecture 2-6
2.3.1 Components Affecting Performance ............................................................... 2-6
  2.3.1.1 Configuration Model Design ................................................................. 2-7
  2.3.1.2 Configuration Interface Object (CIO) .................................................. 2-7
  2.3.1.3 Custom User Interface Design and Implementation ......................... 2-7
  2.3.1.4 Web Server ......................................................................................... 2-8
  2.3.1.5 Oracle Configurator Servlet ............................................................... 2-8
  2.3.1.6 Database Server .................................................................................. 2-8
  2.3.1.7 The Network ....................................................................................... 2-9
  2.3.2 Server Machine Tuning ........................................................................... 2-9
  2.3.3 Client Machine Tuning .......................................................................... 2-10
  2.4 Isolating Performance Problems and Intervening ........................................ 2-11
    2.4.1 System Transactions ........................................................................... 2-11
    2.4.2 Pricing and ATP .................................................................................. 2-12
    2.4.3 Database Commits ............................................................................... 2-12
    2.4.4 Oracle Applications Debugger Tool ................................................... 2-13
    2.4.5 Tuning Hot Spots ............................................................................... 2-13
      2.4.5.1 If Your Configuration Model Has Changed ................................... 2-13
      2.4.5.2 If Your Web Server Setup or Configuration Has Changed .......... 2-14
      2.4.5.3 If You Upgraded Oracle Applications ......................................... 2-14
      2.4.5.4 If You Applied Oracle Configurator Patches ............................... 2-14
      2.4.5.5 If End-User Activity or the Client Machine Has Changed .......... 2-15
      2.4.5.6 If Your User Community or Site Traffic Has Changed ............... 2-15

3 Configuration Model Design

  3.1 Overview ..................................................................................................... 3-1
  3.2 Model Structure ......................................................................................... 3-2
    3.2.1 Size ....................................................................................................... 3-2
    3.2.2 Hierarchical Depth .............................................................................. 3-3
    3.2.3 Complexity .......................................................................................... 3-3
    3.2.4 Option Features and Options .............................................................. 3-4
  3.3 User Interface ............................................................................................. 3-5
    3.3.1 Number and Type of Screens and Controls ......................................... 3-6
    3.3.2 Visibility Settings ............................................................................... 3-6
    3.3.3 Memory Usage and Model Load Setting ............................................ 3-6
    3.3.4 Graphics ............................................................................................. 3-7
3.3.5 Custom User Interface ................................................................................................... 3-7
3.4 Rules ........................................................................................................................................ 3-7
3.4.1 Number and Complexity of Rules .................................................................................. 3-8
3.4.2 Redundancy ....................................................................................................................... 3-9
3.4.3 Rule Types and Relations .................................................................................................. 3-10
3.5 Functional Companions ......................................................................................................... 3-10
3.5.1 Runtime Nodes ................................................................................................................. 3-11
3.5.2 Functional Companion Event Sequence ......................................................................... 3-13
3.5.3 Adding and Deleting Optional Components ..................................................................... 3-13
3.5.4 Optimization of Validation Functional Companions ..................................................... 3-18

4 Web Server Configuration
4.1 Overview ........................................................................................................................................ 4-1
4.2 Apache Web Server Parameters ............................................................................................. 4-2
4.2.1 The jserv.conf File ............................................................................................................. 4-2
4.2.1.1 ApJServManual ........................................................................................................... 4-2
4.2.1.2 ApJServVMTimeout .................................................................................................... 4-3
4.2.2 The zone.properties File .................................................................................................. 4-3
4.2.2.1 session.timeout ......................................................................................................... 4-3
4.2.3 The httpd.conf and httpds.conf Files ............................................................................. 4-4
4.2.3.1 Timeout ...................................................................................................................... 4-4
4.2.3.2 KeepAlive .................................................................................................................. 4-5
4.2.3.3 MaxKeepAliveRequests ............................................................................................ 4-5
4.2.3.4 KeepAliveTimeout ....................................................................................................... 4-5
4.2.3.5 MaxSpareServers ........................................................................................................ 4-6
4.2.3.6 MinSpareServers ........................................................................................................ 4-6
4.2.3.7 MaxClients .................................................................................................................. 4-6
4.2.4 The jserv.properties File ................................................................................................ 4-7
4.2.4.1 Heap Size .................................................................................................................. 4-7
4.2.4.2 cz.uiservlet.dio_share ............................................................................................... 4-8
4.2.4.3 cz.uiserver.lfalse_is_not_available .......................................................................... 4-8
4.2.4.4 cz.uiserver.lazyload ................................................................................................. 4-9
4.2.4.5 cz.uiservlet.pre_load_filename ............................................................................... 4-10
4.2.4.6 cz.activemodel ........................................................................................................... 4-10
4.2.4.7 cz.uiserver.poll_timeout_applet .............................................................................. 4-11
5 Tools and Collecting Data

5.1 Data Collection and Analysis ................................................................. 5-1
  5.1.1 Areas To Examine ........................................................................... 5-1
  5.1.2 Benchmarking ................................................................................. 5-2
  5.1.3 Analyzing Data ............................................................................... 5-3
5.2 Data Collection Timing and Scope ......................................................... 5-4
  5.2.1 Server Side Data ............................................................................. 5-4
  5.2.2 Client Side Data ............................................................................. 5-4
5.3 Tool: LoadRunner .................................................................................. 5-6
  5.3.1 Settings Specific to Oracle Configurator ......................................... 5-8
  5.3.2 General Tips and Guidelines ......................................................... 5-8

A Summary of Web Configuration Parameters

B Case Studies
  B.1 Machine Configuration for Cases A and B ............................................. B-2
     B.1.1 Web Server .................................................................................. B-2
     B.1.2 Database Server ......................................................................... B-2
     B.1.3 LoadRunner Client Machine ....................................................... B-2
     B.1.4 Software ..................................................................................... B-3
     B.1.5 Network ...................................................................................... B-3
  B.2 Case A: Small Configuration Model .................................................... B-3
  B.3 Case B: Medium Configuration Model ................................................ B-7
  B.4 SQL*Plus Queries For Determining Number of Nodes ....................... B-10
     B.4.1 Number of Nodes by Type in a Configuration Model .................... B-10
B.4.2  Number of Features by Type in a Configuration Model.................................  B-12
B.4.3  Number of Rules by Type in a Configuration Model......................................  B-13
B.4.4  Number of Rules by Relation in a Configuration Model..................................  B-14

C  LoadRunner Settings

C.1  Virtual User (Vuser) Generator ...........................................................................  C-1
C.1.1  Tools Menu: Recording Options .................................................................  C-2
C.1.2  Tools Menu: General Options ........................................................................  C-2
C.1.3  Vuser: Runtime Settings ..................................................................................  C-2
C.2  Vuser Parameterization .........................................................................................  C-4
C.3  Modifying Vuser Scripts........................................................................................  C-4
C.3.1  Parameterize the Session ID ............................................................................  C-4
C.3.2  Think Times ......................................................................................................  C-5
C.3.3  Use Host and Port Parameter ..........................................................................  C-5
C.3.4  Parameterize User Name and Password ............................................................  C-5
C.3.5  Leverage Cookies ..............................................................................................  C-6
C.3.6  Optimize Vuser Scripts.....................................................................................  C-6
C.4  Debugging Vuser Scripts.......................................................................................  C-7

Glossary of Terms and Acronyms

Index
List of Examples

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3–1</td>
<td>Rules With Common Subexpressions</td>
<td>3-8</td>
</tr>
<tr>
<td>3–2</td>
<td>Redundant Rules That Create Cycles</td>
<td>3-9</td>
</tr>
<tr>
<td>3–3</td>
<td>Approach A (Slower)</td>
<td>3-15</td>
</tr>
<tr>
<td>3–4</td>
<td>Approach B (Faster)</td>
<td>3-15</td>
</tr>
<tr>
<td>3–5</td>
<td>Detailed Approach A (Slower)</td>
<td>3-16</td>
</tr>
<tr>
<td>3–6</td>
<td>Detailed Approach B (Faster)</td>
<td>3-16</td>
</tr>
<tr>
<td>3–7</td>
<td>Setting Features (Slower Code)</td>
<td>3-17</td>
</tr>
<tr>
<td>3–8</td>
<td>Setting Features (Faster Code)</td>
<td>3-17</td>
</tr>
<tr>
<td>3–9</td>
<td>Minimizing Validation Test on a Configuration Model</td>
<td>3-18</td>
</tr>
<tr>
<td>3–10</td>
<td>Causing More Validation Tests on a Configuration Model</td>
<td>3-18</td>
</tr>
<tr>
<td>B–1</td>
<td>Determine Number of Nodes by Type in a Configuration Model</td>
<td>B-11</td>
</tr>
<tr>
<td>B–2</td>
<td>Determine Number of BOM Nodes in a Configuration Model</td>
<td>B-11</td>
</tr>
<tr>
<td>B–3</td>
<td>Determine Number of non-BOM Nodes in a Configuration Model</td>
<td>B-12</td>
</tr>
<tr>
<td>B–4</td>
<td>Determine DEVL_PROJECT_ID of a Configuration Model</td>
<td>B-12</td>
</tr>
<tr>
<td>B–5</td>
<td>Determine Number of Features by Type in a Configuration Model</td>
<td>B-12</td>
</tr>
<tr>
<td>B–6</td>
<td>Determine Number of Rules by Type in a Configuration Model</td>
<td>B-13</td>
</tr>
<tr>
<td>B–7</td>
<td>Determine Number of Rules by Relation in a Configuration Model</td>
<td>B-14</td>
</tr>
<tr>
<td>C–1</td>
<td>Obtaining Parameterized Session ID in LoadRunner Vuser Script</td>
<td>C-5</td>
</tr>
<tr>
<td>C–2</td>
<td>Code for Printing Out Cookie Names</td>
<td>C-6</td>
</tr>
<tr>
<td>C–3</td>
<td>Transactions in Test Scripts</td>
<td>C-6</td>
</tr>
</tbody>
</table>
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3–1</td>
<td>Model Structure for Adding Components</td>
<td>3-17</td>
</tr>
<tr>
<td>5–1</td>
<td>Elapsed Times in <em>czSource.htm</em> File</td>
<td>5-6</td>
</tr>
<tr>
<td>5–2</td>
<td>Screen Capture of LoadRunner CPU Utilization Analysis</td>
<td>5-7</td>
</tr>
</tbody>
</table>
List of Tables

3–1 Configuration Model Size ........................................................................................................... 3-3
A–1 Web Server Parameters That Affect Oracle Configurator Performance .......................... A-1
B–1 Case A Model Structure .............................................................................................................. B-3
B–2 Case A Rule Types and Operators ............................................................................................ B-4
B–3 Case A User Interface Controls .................................................................................................. B-5
B–4 Case A Server Performance Times ............................................................................................. B-6
B–5 Case A CPU Utilization ................................................................................................................ B-7
B–6 Case B Model Structure ............................................................................................................... B-7
B–7 Case B Rule Types and Operators .............................................................................................. B-8
B–8 Case B User Interface Controls .................................................................................................. B-8
B–9 Case B Server Performance Times .............................................................................................. B-9
B–10 Case B CPU Utilization ............................................................................................................... B-10
B–11 Numeric Identification of PS_NODE_TYPE .............................................................................. B-10
B–12 Numeric Identification of FEATURE_TYPE ............................................................................. B-12
B–13 Numeric Identification of RULE_TYPE ................................................................................... B-13
B–14 Numeric Identification of EXPR_RULE_TYPE ......................................................................... B-14
Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

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

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

- Electronic mail: czdoc_us@oracle.com
- FAX: 781-238-9898. Attn: Oracle Configurator Documentation
- Postal service:
  Oracle Corporation
  Oracle Configurator Documentation
  10 Van de Graaff Drive
  Burlington, MA 01803-5146
  USA

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

If you have problems with the software, please contact your local Oracle Support Services.
Welcome to the Oracle Configurator Performance Guide. This book contains information you need for optimizing runtime performance of Oracle Configurator. Using this document, you can prepare for and implement high performance configuration model designs, Web server configuration, and runtime testing. This book should not be used alone, but together with the other books in the Oracle Configurator documentation set.

This preface describes how the book is organized, who the intended audience is, and how to interpret the typographic conventions.

For details about what is and is not included in this book, see Section 1.1, "Scope of this Guide" on page 1-1.

Intended Audience

This guide is intended for Oracle Consultants and implementers who are deploying a Web-based Configurator. You should have a working knowledge of your business processes, tools, configuration models, and the other books in the Oracle Configurator documentation set. You should also be familiar with Oracle Applications and the Oracle Applications database.

If you have never used a configurator application, we suggest you attend one or more of the Oracle Configurator training classes available through Oracle University.

Additional specifics about audience are included in the Structure section of this preface.
Documentation Accessibility

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

Accessibility of Code Examples in Documentation  JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

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

Structure

This book contains a table of contents, lists of examples, tables, and figures, a reader comment form, a preface, several chapters, appendixes, a glossary, and an index.

Introduction

Useful for anyone planning, implementing, or testing Oracle Configurator, this chapter explains the scope of the guide, defines terms, and provides background information about Oracle Configurator that is relevant to evaluating and tuning runtime performance.

Tuning

Useful for anyone preparing Oracle Configurator for deployment, this chapter describes the tuning process and best practices for tuning performance over the
lifetime of your project, as well as important tuning recommendations not included in Configuration Model Design and Web Server Configuration.

Configuration Model Design
Intended principally as a reference for implementers using Oracle Configurator Developer, this chapter contains details about how to avoid having your configuration models be the cause of performance problems at runtime.

Web Server Configuration
Intended principally as a reference for the Web server administrator or the database administrator responsible for configuring and maintaining Web servers, this chapter lists default and recommended values for parameters of the Web server that affect Oracle Configurator performance.

Tools and Collecting Data
Intended principally as a reference for anyone involved in testing Oracle Configurator, this chapter explains what and under what conditions to collect data for performance evaluation and tuning. LoadRunner (Mercury Interactive Corporation) is the sample tool. Appendix C, “LoadRunner Settings” lists specific LoadRunner settings.

Case Studies
Intended principally as a reference for anyone involved in planning, implementing, or testing a production Oracle Configurator, this appendix contains examples of performance data for various Oracle Configurator deployments based on types of configuration models and environments. The appendix includes the queries used for determining number of nodes in these models. The appendix also contains the specifications of the database server, Web server, and client machine configuration used in conducting these case studies.

LoadRunner Settings
Intended principally as a reference for anyone setting up LoadRunner tests, this appendix describes some LoadRunner parameters that are specific to Oracle Configurator and that must be set in order to collect performance data.

Summary of Web Configuration Parameters
Intended for the Web server administrator or the database administrator responsible for configuring and maintaining Web servers, as an overview at a
glance of the Apache Web server parameters that affect Oracle Configurator, their recommended values, and the file in which the parameter is set.

**Glossary of Terms and Acronyms**
This manual contains a glossary of terms used throughout the Oracle Configurator documentation set.

**Related Documents**
For more information about Oracle Configurator, see the following manuals in the Oracle Configurator documentation set:


The following documents may also be useful:

- *Oracle Configurator Release Notes* (Part No. B10620-01)
- *LoadRunner: Creating GUI Vuser Scripts Online Guide*
- *Oracle Applications Developer’s Guide* (Part No. A75545–03)
- *Maintaining Oracle Applications* (Part No. A90810–01)
- *Oracle 8i Tuning* (Part No. A67775-01)
- *Oracle 8i Designing and Tuning for Performance* (Part No. A76992-01)
- *Oracle 9i Database Performance Methods* (Part No. A87504-02)
- *Oracle 9i Database Performance Planning* (Part No. A96532-01)
- *Oracle 9i Database Performance Tuning Guide and Reference* (Part No. A96533-01)

The following documents represent a possible starting point in researching tuning and benchmarking methodology:
Additional white papers and information about benchmarking and tuning are available at:
- http://www.benchmarkresources.com

Other sites of interest are:
- http://httpd.apache.org/docs-project/
- http://apache.us.oracle.com

**Conventions**

In examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the Return key at the end of a line of input.

The following conventions are also used in this manual:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>. . .  .</td>
<td>Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.</td>
</tr>
<tr>
<td>. . . . . .</td>
<td>Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted</td>
</tr>
<tr>
<td><strong>boldface text</strong></td>
<td>Boldface type in text indicates a new term, a term defined in the glossary, specific keys, and labels of user interface objects. Boldface type also indicates a menu, command, or option, especially within procedures</td>
</tr>
<tr>
<td><em>italics</em></td>
<td>Italic type in text, tables, or code examples indicates user-supplied text. Replace these placeholders with a specific value or string.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Brackets enclose optional clauses from which you can choose one or none.</td>
</tr>
</tbody>
</table>
Product Support

The mission of the Oracle Support Services organization is to help you resolve any issues or questions that you have regarding Oracle Configurator Developer and Oracle Configurator.

To report issues that are not mission-critical, submit a Technical Assistance Request (TAR) using Metalink, Oracle’s technical support Web site at:

http://www.oracle.com/support/metalink/

Log into your Metalink account and navigate to the Configurator TAR template:

1. Choose the TARs link in the left menu.
2. Click on Create a TAR.
3. Fill in or choose a profile.
4. In the same form:
   a. Choose Product: Oracle Configurator or Oracle Configurator Developer
   b. Choose Type of Problem: Oracle Configurator Generic Issue template
5. Provide the information requested in the iTAR template.

You can also find product-specific documentation and other useful information using Metalink.

For a complete listing of available Oracle Support Services and phone numbers, see:

www.oracle.com/support/
This **Introduction** explains the scope of the *Oracle Configurator Performance Guide* and relevant terms used in the book.

For an overview of Oracle Configurator architecture, see the *Oracle Configurator Implementation Guide*.

### 1.1 Scope of this Guide

This guide focuses primarily on the Applications or middle tier of Oracle Configurator’s classic three-tier architecture. See Section 2.3, "Tuning Key Components of the Oracle Configurator Architecture" on page 2-6 for details about what the middle tier includes.

Within the Applications tier, this guide focuses on the issues involved in accurately measuring performance and scalability, as well as the analysis tools used to measure performance. The guide discusses how to plan and conduct a comprehensive benchmark for a Oracle Configurator implementation.

The main areas of potential performance gain are those where most of the processing takes place:

- Configuration Model Design
- Web Server Configuration

This guide focuses on information specific to Oracle Configurator in these areas, but also provides explanations of tuning potential in other Oracle Configurator components of a deployment (Chapter 2, "Tuning").

To help you assess what configuration model data is relevant to performance testing, this guide includes several case studies of performance tests by Oracle Configurator Development.
Relevant Terms

The following topics are also important for tuning performance, but not the focus of this book:

- Tuning the database or database server (some suggestions are presented in Chapter 2, “Tuning”)
- Tuning the client machine setup (some suggestions are presented in Chapter 2, “Tuning”)
- Determining machine sizes for your deployed application
- Tuning non-runtime performance such as Oracle Configurator Developer, configuration model publication, or data import and refresh performance
- Tuning the host application

1.2 Relevant Terms

For a complete list of terms used in Oracle Configurator documentation, see the Glossary at the end of this book. To locate more information about the highlighted concepts in this section, refer to this book’s Table of Contents and Index.

**Runtime** refers to a launched Oracle Configurator window, either while unit testing directly from within Oracle Configurator Developer, or invoked from a hosting application. When an Oracle Configurator is deployed, it is invoked from a hosting application either for system testing or production.

Starting up Oracle Configurator makes a configuration model accessible to end users. A **configuration model** consists of model structure and rule definitions. It also commonly includes user interface definitions and possibly UI customizations, as well as **Functional Companions**. Functional Companions extend the behavior of the configuration model beyond what can be implemented in Oracle Configurator Developer. These elements of the model are also called the **Active Model** and the **UI**. The Active Model represents the valid configurations allowable by the rules defined in the configuration model.

Launching or initializing an Oracle Configurator window **loads a configuration model**, unless the model is **preloaded**. Preloading the configuration model means it is cached in memory before launching the Oracle Configurator window and starting a **configuration session**. **Load time** can be a critical element in performance.

Once a configuration model is loaded, end users work through a **configuration session** by making selections on a **client** machine connected to the **Web server** running the Oracle Configurator Servlet.
Response time is a critical element in session performance. The total end-user response time is the total throughput time under load, including the server response time, display time, and network latency.

Normally, a configuration session results in a completed or valid configuration. A single configuration session is limited to the one configuration model that is loaded when the session is initialized.

Performance measures the operation of a product, in this case the runtime Oracle Configurator. For the purposes of this guide, performance is the speed with which the product loads and responds to end-user actions under various conditions. When performance is poor—commonly too slow—the configuration model, Web server, or other elements of the deployment can be tuned to improve performance.

The configuration model data in the database and the application software running on a Web server should be running on different machines in a distributed computing environment on a single network.

Project goals, end-user expectations, and previous performance influence the decision to tune the various components of a deployment. A benchmark represents performance data collected during a runtime test under various conditions that emulate expected or extreme use of the product.

Based on throughput and other performance data, you can refine the sizing of your deployment environment, meaning determine the machines and software configuration needed to run your tuned application.
This chapter explains the fundamentals of performance tuning and how various components of Oracle Configurator architecture affect performance.

---

**Note:** Good tuning starts with good design. You can reduce your tuning efforts by designing your configuration model with performance in mind (see Chapter 3, "Configuration Model Design").

---

The sections are:

- Fundamentals of Performance Tuning
- The Tuning Process
- Tuning Key Components of the Oracle Configurator Architecture
- Isolating Performance Problems and Intervening

See the Section "Structure" on page xviii in the Preface to verify the audience for whom this chapter is intended.

### 2.1 Fundamentals of Performance Tuning

Performance tuning is a complex, iterative process. Deriving meaningful results requires using a proven methodology and understanding the environment, how the application will be used, and the required throughput. Using a proven methodology minimizes confusion and gaps in the process. By understanding the environment, application, and throughput, you will be able to define the kind of benchmarks you need to accurately reflect the scenario in which you operate.
2.1.1 Methodology

A methodology provides a general approach to performance tuning that helps you avoid gaps or missed opportunities. At the core of a useful methodology is a repeatable process for evaluating, diagnosing, and intervening to improve performance.

In the case of Oracle Configurator, intervention can consist of

- modifying the configuration model design
- changing the Web server configuration
- upgrading to a more recent release
- applying available patches
- modifying the hardware setup

Additional intervention can occur in the database and integration points with hosting applications.

Depending on your needs, intervention should generally maximize the largest potential benefit for the broadest range of circumstances.

Evaluating or analyzing the problem includes understanding what has been done to date and comparing performance with baselines and previous benchmarks. Comprehensive documentation of baselines and benchmarks is prerequisite to effective analysis and comparisons.
Diagnosing performance bottlenecks or hot spots and deciding on appropriate intervention requires access to expertise in your business model, the application, and the environment and technology stack.

Successful diagnosis and tuning requires:

- understanding your expectations of success and determining what are reasonable and achievable goals
- dividing the application’s key components into measurable units and evaluating their performance separately, as well as collectively
- having access to expert knowledge of your business model, the life of the application, the workflow of operation, and the environment and technology stack
- having access to expert knowledge of Oracle Configurator, including Oracle Configurator Developer, for understanding the configuration model’s role in performance
- benchmarking the application at various usage levels
- repeating the tuning process when measurements fall short
- diligently recording steps and documenting results to make the process reliable and repeatable

For more information about collecting and analyzing data, see Chapter 5, “Tools and Collecting Data”.

2.1.2 Documenting Results

It is important to maintain detailed notes during each phase of the data collection, analysis, and tuning process. Detailed documentation of setup, actions, and results helps ensure that the process is reproducible and reliable. See Appendix B, “Case Studies” for examples of how to document final tuning results.

2.1.3 Defining Success Criteria

You will need to determine at what point in the tuning process your configuration model performance is good enough for deployment. This requires balancing what can be expected from your software and hardware with what your end users find acceptable. Conduct usability studies with sample end users to determine additionally what your performance goals should be.
2.2 The Tuning Process

The tuning process, following a repeatable and reliable methodology, allows you to make decisions and changes based on appropriate analysis of the data. Tuning Oracle Configurator effectively requires following a proven, iterative sequence of tasks and specific steps within those tasks.

2.2.1 Steps in the Process

Tuning Oracle Configurator is a step-by-step process.

1. Collect performance data for various operations within Oracle Configurator using the tools described in Chapter 5, "Tools and Collecting Data". See Section 2.4, "Isolating Performance Problems and Intervening" on page 2-11 for a list of runtime events and other functionality where performance problems may occur.

2. Document the results. If the performance numbers are not satisfactory, begin tuning.

   If the Oracle Configurator implementation has never been performance tested or tuned before, continue on with the process described here.

   If the Oracle Configurator implementation has previously performed better, either repeat this process or see the suggestions in Section 2.4.5, "Tuning Hot Spots" on page 2-13.

3. Start with tuning the configuration model design, as described in Chapter 3, "Configuration Model Design".

   The sequence of tuning tasks within the configuration model design should be, in order:
   - Model Structure
   - User Interface
   - Rules
   - Functional Companions

4. Collect incremental performance numbers after every few changes. Document the changes made and the resulting performance numbers.

5. Iterate over the previous two steps until either the performance numbers are satisfactory or no more changes are possible to the model structure, rules, user interface, or Functional Companions.
6. Focus next on the Web server configuration. Web server tuning is generally, but not exclusively indicated when performance for a single user is good, but significantly degraded with multiple users.

Monitor CPU utilization and memory usage for the most costly events:

a. model load
b. configuration save
c. Functional Companions

7. Start the iterative process of tuning the Web server.

The sequence of tuning tasks within the Web server configuration should be, in order:

- change configuration parameters as described in Chapter 4, "Web Server Configuration".
- modify number of Java Virtual Machines (JVM) per CPU
- modify maximum number of end users per JVM to avoid 100% CPU utilization and dropping the JVM

8. Document the changes made to the Web server and collect the resulting performance numbers.

9. Iterate over the previous two steps until either the performance numbers are satisfactory or no more changes are possible.

10. For performance improvement in the area of loading or preloading configuration models and saving configurations, attention to database tuning would be helpful. Database tuning information is not within the scope of this book. See also Section 3.3.3, "Memory Usage and Model Load Setting" on page 3-6 for considerations in loading and preloading configuration models.

11. For additional performance benefits, tune the setup of the client machine. Although client machine setup tuning is not within the scope of this book, see Section 2.3.3, "Client Machine Tuning" on page 2-10 for some suggestions.

The sequence of tuning tasks within the client machine setup should be, in order:

- browser settings
- swap space
- memory
12. For additional performance improvement in the area of rendering UI controls, screens, and propagations of selection changes in the Oracle Configurator window, attention to network tuning between the client and the Web server could be helpful.

Although network tuning information is not within the scope of this book, see Section 2.3.1.7, "The Network" on page 2-9 for suggestions.

It is important to maintain detailed notes during each phase of the tuning process. This makes the process reproducible and reliable. Appendix B, "Case Studies" presents such documentation of final results in several example benchmarks.

2.3 Tuning Key Components of the Oracle Configurator Architecture

Oracle Configurator operates in a classic three-tier environment. The configuration model, user interface definitions and business rules are stored in the Database tier. These definitions are processed in the Applications tier and rendered in the User Interface tier or layer. Most of the processing for Oracle Configurator is performed in the middle or Applications tier. In that sense, Oracle Configurator is a typical middle-tier application.

See the Oracle Configurator Implementation Guide for details about the architecture and application flow among tiers and elements of an Oracle Configurator deployment.

For best performance, set up your Oracle Configurator deployment with different server machines for the database and the application or Web server (middle tier). For details about this, see Section 2.3.2, "Server Machine Tuning" on page 2-9. Size the middle tier (where the OC Servlet with the Oracle Configurator engine and UI Server are installed) according to the complexity and size of your configuration model, as well as the end-user demands on your runtime Oracle Configurator.

2.3.1 Components Affecting Performance

The optimal performance of a Web-based Oracle Configurator depends on the performance of the following elements.

- Configuration Model Design
- Configuration Interface Object (CIO)
- Custom User Interface Design and Implementation
- Web Server
Tuning Key Components of the Oracle Configurator Architecture

- Oracle Configurator Servlet
- Database Server
- The Network

These components require careful analysis to get the best possible performance out of the system. Tuning requires iterating over all these essential components, tuning each of them independently and collectively.

2.3.1.1 Configuration Model Design
All parts of a configuration model participate in performance. You benefit from designing each part thoroughly to optimize performance before starting model construction. For a review of what a configuration model entails, review Section 1.2, "Relevant Terms" on page 1-2. For details about tuning model design, see Chapter 3, "Configuration Model Design".

2.3.1.2 Configuration Interface Object (CIO)
The CIO is an API layer that handles communication between the Active Model and the UI. Functional Companions and custom UIs communicate with the Active Model through the CIO.

There is generally no tuning that can be done in the CIO interactions. However, the design and implementation of code that uses CIO methods to access the configuration model can be tuned to use the CIO most effectively.

Functional Companions and custom UI code can affect runtime performance by less than optimal use of the CIO methods. You can optimize (or minimize) the sequence of changes to a configuration model that a Functional Companion invokes by avoiding expensive intermediary transactions in the CIO. For details on tuning Functional Companions, see Section 3.5, "Functional Companions" on page 3-10.

2.3.1.3 Custom User Interface Design and Implementation
In some deployments, the user interface is entirely custom, meaning it is not generated as a UI using Oracle Configurator Developer. If the custom UI code interfaces with the Configuration Interface Object (CIO), then it is important to optimize how the UI code calls the CIO. For details about designing a custom UI for better performance, see Section 3.3.5, "Custom User Interface" on page 3-7.
2.3.1.4 Web Server
Most of Oracle Configurator processing occurs in the Web server portion of the middle tier. Data retrieved from the database is processed in memory in the Web server.

Oracle Configurator uses the high performance Apache Web server in the Oracle Internet Application Server (iAS).

For details about the parameters in various configuration files that are used to tune performance, see Chapter 4, "Web Server Configuration".

2.3.1.5 Oracle Configurator Servlet
The Oracle Configurator Servlet is the machinery responsible for rendering the User Interface and brokering the communication between the configuration model, the database, and the client. The Oracle Configurator Servlet is installed on the Web server.

Typical OC Servlet transactions are longer than database transactions and compete for CPU time. See Section 2.4.1, "System Transactions" on page 2-11 for details.

There is generally no tuning that can be done in the OC Servlet interactions. However, the design and implementation of the configuration model can be tuned to use the OC Servlet most effectively.

2.3.1.6 Database Server
Oracle Configurator in the middle tier accesses the database during the initialization and saving of configurations. Data is retrieved from the database in intervals and processed in memory in the Web server. See Section 2.4.1, "System Transactions" on page 2-11 for data transaction scheduling.

For more efficient use of database server resources, purge records flagged for deletion before creating the production-ready version. See Oracle Configurator Implementation Guide for more information about purging records.

While there is no specific database tuning required for Oracle Configurator, following the general recommendations for Oracle Applications is likely to improve performance.

**Note:** Tune first the configuration model design and then the Web server configuration before tuning the database.
2.3.1.7 The Network

Network latency may affect production end users. Network latency may not be accounted for in benchmarking test results.

The Web server and database servers should be on separate machines, but on the same high bandwidth network. Timeout settings of any firewall should be set to none to prevent killing the database connection.

Network latency between client and server could be an issue if there is a big difference between client side response times (recorded in the czSource.htm file) and the time the same action takes on the Web server (recorded in the appropriate cz_session log file). See Section 5.2.2, "Client Side Data" for details about the czSource.htm file.

See the Oracle Configurator Implementation Guide for additional details about network considerations in an Oracle Configurator deployment.

2.3.2 Server Machine Tuning

Because Oracle Configurator processes compete with database processes for CPU time and OC Servlet transactions are longer than typical database transactions, the Web server and database server are never installed on the same machine.

Note: Oracle strongly recommends that the Web server and the database server and the client all be installed on separate machines, but on the same network.

See Section 2.4.1, "System Transactions" on page 2-11 for more information about how Oracle Configurator and database processes compete.

Garbage collection (GC) is an important part of tuning a Java Web server application. Be sure to increase the memory as you increase the number of processors, since allocation can be parallelized, but GC is not parallel.

If server machine memory is an issue, a router may be needed to spread the end users to particular CPUs dedicated to supporting a subset of the possible configuration models. Routing end user access to specific servers running a subset of models requires additional planning and testing. See the Oracle Configurator Implementation Guide for additional details about using routers in an Oracle Configurator deployment.

Appendix B, "Case Studies" describes the Web server and database server configuration used internally for testing by Oracle Development.
2.3.3 Client Machine Tuning

The client machine configuration is not generally under the control of the Web-based Oracle Configurator in a typical business-to-consumer production environment. However, it is possible to include guidance on the host application Web site recommending machine and browser specifications that provide the best performance.

When client machines are under your control, such as during testing or for internal deployments, tune the client machines for optimal performance. Tuning the client machine configuration is likely to produce performance improvements.

Appendix B, "Case Studies" describes a client configuration used internally for testing by Oracle Development.

If you have control over the client machine setup, you can improve client side performance by:

- Setting up adequate swap space and keeping disks de-fragmented on the client machine
- Increasing the client machine memory, virtual memory, CPU
- Verify that browser settings support DHTML (see the Oracle Configurator Implementation Guide for details)
- Tuning browser settings, browser types, browser versions

For optimal runtime performance on a client running Microsoft Internet Explorer, make sure the following browser settings are in place:

1. In the Tools menu, Internet Options command, General tab, Temporary Internet Files section:
   a. Verify that Check for newer versions of stored pages is set to Automatically.
      Setting Check for newer versions of stored pages to Every visit to the page slows runtime performance by from 10 to 20 seconds per screen flip
   b. Verify that Temporary Internet Files Folder section is set to a large temporary area (at least 20 MB, preferably 50-100 MB) for storing Configurator pages.

2. If the end user repeatedly accesses a configuration model, it is helpful to preserve the browser cache. In the Tools menu, Internet Options command, Advanced tab, Security heading (bottom of the list):
a. Verify that **Do not save encrypted pages to disk** is *not* selected. In other words, you want encrypted pages saved to disk.

b. Verify that **Empty Temporary Internet Files folder when browser is closed** is *not* selected In other words, when you close the browser, the temporary Internet Files folder is not emptied and the browser cache is preserved.

If these two settings *are* selected, you will not get optimal performance because the effect is the same as clearing the browser cache.

### 2.4 Isolating Performance Problems and Intervening

Optimizing the following can lead to significant performance gains the next time you test the system or run a benchmark:

- System Transactions
- Pricing and ATP
- Database Commits
- Oracle Applications Debugger Tool
- Tuning Hot Spots

Beyond these common problem areas, you can explore and tune hot spots, as presented in Section 2.4.5 on page 2-13.

#### 2.4.1 System Transactions

Runtime performance involves distinct events:

- Transmission of user action
- Processing of UI event by the Oracle Configurator Servlet
- Processing of logic changes by the Oracle Configurator engine
- Creation of UI changes by Oracle Configurator Servlet
- Transmission to and rendering by the client browser

Commonly, most of the processing occurs in the client to Web server network and UI rendering. Logic transactions are *relatively* faster when measured against that network and UI processing. However, rule design and number do affect performance of logic transactions in the OC engine (see Section 3.4, "Rules").
Oracle Configurator is a Central Processing Unit (CPU) intensive application. Data is retrieved from the database in intervals and processed in memory in the Web server. The default UNIX "round robin" CPU scheduling favors short transactions. Typical database transactions are short transactions. Typical Oracle Configurator Servlet transactions are long-running transactions.

---

**Note:** In a busy system where CPU utilization is high, Oracle Configurator processes lose out to the database processes and get only a small slice of the CPU at a time. This radically affects performance (server response time and display time). For this reason, running the database and Web server on separate machines in a distributed computing environment is critically important. See also Section 2.3.2, “Server Machine Tuning” on page 2-9.

---

### 2.4.2 Pricing and ATP

You can improve performance by fetching pricing data only when it is needed for the end user of your application. Runtime performance problems could arise from pricing and Available to Promise (ATP) processing within Oracle Configurator.

If pricing and ATP are needed, isolate them as a performance problem by turning them off. Intervene by tuning the frequency of transactions involving retrieval of pricing and ATP data. See Section 4.2.4.6, "cz.activemodel” and Section 4.2.4.10, "cz.activemodel.lazyloadlistprice” for additional details.

For more information about pricing and ATP, see the Oracle Configurator Implementation Guide.

### 2.4.3 Database Commits

In some cases, hosting applications must commit end-user changes before launching Oracle Configurator. Initializing and loading the configuration model can be slow if an implicit save occurs. If the end user explicitly saves work in the host application before trying to start a configuration session, the perceived performance of the Oracle Configurator improves. Order Management is an example where an explicit save before clicking the Configurator button significantly improves the configuration model load time.
2.4.4 Oracle Applications Debugger Tool

The Oracle Applications Forms Debugger tool (Help -> Diagnostics -> Debug if the form was started in debug mode) can influence performance of Oracle Configurator. Isolate the Forms Debugger as a performance problem by turning it off. Intervene by keeping the Forms Debugger off during production deployment.

2.4.5 Tuning Hot Spots

Ideally, you tune Oracle Configurator following the overall sequence outlined in Section 2.2.1, "Steps in the Process". When tuning for hot spots, the sequence may be different, particularly if limited changes have occurred in a runtime Oracle Configurator that previously performed well.

Hot spot tuning involves identifying the area of change in the implementation since the previous performance test, and tuning that area if the results show degradation. It is critical to have baseline performance data with which to compare the changes. In Section 2.4.5.1 through Section 2.4.5.6, the assumption is that only one factor in the implementation has changed. For example, if your configuration model has changed and nothing else has, why might your performance results have changed.

2.4.5.1 If Your Configuration Model Has Changed

If your configuration model has changed, performance problems could manifest themselves in the following areas:

- Model load time, if the number of nodes has increased significantly (see Section 3.2.1, "Size" on page 3-2)
- Web server performance, if the rules or UI customizations have changed
- Client machine performance, if the UI customizations or number of options to render in option lists have changed.

Reducing the number of nodes in a configuration model is not usually an option. However, you can optimize configuration model load time by changing the way a large model is loaded, or by pre-loading it (see Section 3.3.3, "Memory Usage and Model Load Setting" on page 3-6).

If rule and UI customizations changes adhere to the design suggestions in Chapter 3, "Configuration Model Design" and there are no further design optimizations possible, you can improve Web server performance by adjusting parameters in the Web server configuration. For example, you may need to adjust the timeout parameters in the Web server configuration. See Chapter 4, "Web Server Configuration".
If there are no further optimizations possible in your UI customization changes, and the Web server performance cannot be improved, consider the role the client machine setup may have in the degraded performance (Section 2.3.3, "Client Machine Tuning" on page 2-10).

2.4.5.2 If Your Web Server Setup or Configuration Has Changed
Document any changes made to the Web server so that you can evaluate why your performance may have changed and so you can go back to a Web server configuration that performed better. For example, routine maintenance or installation of a database or additional software could adversely affect performance. Changing one `jserv.properties` parameter to resolve a new issue may have runtime implications that require other parameters to be adjusted as well in order to sustain an acceptable level of performance.

Inadvertent changes could be the cause of performance degradation. For example, one of the CPUs has failed and needs to be repaired or replaced.

2.4.5.3 If You Upgraded Oracle Applications
After an upgrade of your Oracle Applications release (for example from 11.5.7 to 11.5.9) identify what now performs more slowly. If there were no changes in the Oracle Configurator setup, but its runtime performance has changed, the change could be in the hosting application.

For example, if it takes longer to save a configuration (performance of the Done button in Oracle Configurator), then the Order Management upgrade may have introduced a change that needs to be examined. Or if you have pricing enabled and you now see slower rendering of Oracle Configurator screens, the Advanced Pricing upgrade may have introduced a change that needs to be examined. In this case, you can easily isolate pricing as the problem by disabling it temporarily (see Section 2.4.2, "Pricing and ATP").

2.4.5.4 If You Applied Oracle Configurator Patches
Absent any other changes, a patch upgrade may introduce functionality that causes changes in performance.

---

**Note:** Before applying an Oracle Configurator patch, read the ARU Patch Readme and the *Oracle Configurator Release Notes*.  
---
Based on information in the ARU Patch Readme and the Oracle Configurator Release Notes, you may need to make adjustments to new functionality and changes that are introduced by the patch, such as disabling or changing the default value of new functionality.

If you are applying an Oracle Configurator patch that upgrades the Oracle Configurator release without upgrading the rest of Oracle Applications, you may encounter problems in reading and writing to the Oracle Applications database. For example, a patch upgrade may cause indexes to be dropped, which affect performance in initializing Oracle Configurator and saving configurations. A database analysis can reveal such defects and areas of deadlock in the database transactions. Applying a subsequent patch can fix the problem.

2.4.5.5 If End-User Activity or the Client Machine Has Changed

Compare the degraded performance on a changed or different client machine to a machine where performance was acceptable. If additional software was installed on the client machine, this could affect performance. If other applications are also running more slowly, this could be an indication of a full disk or insufficient memory.

2.4.5.6 If Your User Community or Site Traffic Has Changed

It is important to plan for user volume appropriately and monitor site use. If you are rolling your implementation out to more clients or the number of users hitting the site has increased, you probably need to adjust the Web server configuration parameters. For instance, you can control the number of clients that can simultaneously connect to the server by adjusting the MaxClients parameter (see Section 4.2.3.7, "MaxClients").
Isolating Performance Problems and Intervening
The design of the configuration model is critically important to the performance of your runtime Configurator. Good design that considers performance implications could avoid the need for tuning the design during performance testing. Careful design and tuning of the configuration model, with special emphasis on size and complexity, yields large performance dividends.

Configuration models should be based on production-ready data. Developing a configuration model based on BOM data that is under development may easily result in poor design, which in turn may negatively affect performance and usability.

Furthermore, the breadth of functionality offered by Oracle Configurator Developer allows you to define a wide range of model types, but does not prevent you from designing ones that perform poorly at runtime. For that reason, this chapter presents known best practices for optimizing configuration model design. See the Oracle Configurator Developer User’s Guide for details about using Oracle Configurator Developer to implement the suggestions in this chapter.

See the Section "Structure" on page xviii in the preface to verify the audience for whom this chapter is intended.

3.1 Overview

You develop the configuration model in Oracle Configurator Developer based on designs that best map your business needs to the characteristics of the Oracle Configurator product. Where various design choices result in equivalent runtime effects and behavior, it is prudent to choose the ones proven to perform best.

The key areas that influence the performance of the configuration model at runtime are:
Model Structure

Oracle Configurator Performance Guide

Tuning the configuration model design involves an iterative process of tuning structure, UI customizations, rules, and the role of Functional Companions in the model. Each of these areas is explored in this chapter.

Configuration model complexity affects performance. Complex configuration models result from the following factors:

- Large BOMs or model structure
- Complex rules
- Large numbers of rules (depending on the rule design)

See Section 3.4, "Rules" on page 3-7 for details about complex, redundant, and large numbers of rules.

3.2 Model Structure

In general, the size, hierarchical depth, or complexity of the model structure is likely to affect the performance at load time and during the configuration session. Additionally, the number and logic state of Options in an Option Features affect performance.

3.2.1 Size

The size of the model directly affects the time it takes to load the model and display the UI. The bigger the model, the longer the initial load. Oracle recommends preloading the model so that end users experience better performance throughout the configuration session. See also Section 3.3.3, "Memory Usage and Model Load Setting" on page 3-6.

The database size of a configuration model is measured in total number of nodes. Section B.4, "SQL*Plus Queries For Determining Number of Nodes" on page B-10 presents queries used to find out how many nodes are in your configuration model. Table 3–1 shows general guidelines for identifying the size of your models.
Typically, configuration models are small, or 800 to 1,000 nodes. Configuration models containing 5,000 to 10,000 nodes are rare and considered to be very large.

### 3.2.2 Hierarchical Depth

The hierarchical depth and number of Reference nodes affect performance at runtime. Hierarchical depth or nesting is generally not an issue, since typically configuration models are not deeply nested. If user requirements are best satisfied with nesting, there is only a slight performance cost in implementing the configuration model with greater hierarchical depth. Using mandatory Components (min=1, max=1) improves performance of configuration models with nesting.

Reference nodes should only be used for structure that is being used repeatedly in the model. Sometimes implementers design and create reference models expecting to reuse the structure, but finally only refer to that reference model once. This results in unnecessary performance loss. Tuning a configuration model design should include making single-use references explicit structure in the model.

The cost of uncovering this performance tuning improvement is having to manually rebuild the referenced structure in the parent Model. Good design and project planning should avoid the need for making underused referenced structure explicit.

### 3.2.3 Complexity

The number of elements in support of intermediary rules, effectivity and Usage settings, Totals and Resources, and the ratio of imported BOM nodes to non-imported nodes, do not affect performance at runtime. See Section 3.4.1, "Number and Complexity of Rules" on page 3-8 for more details about structure in support of intermediary rules.

It is more efficient to assign constant values to Properties than in Options or Numeric Features.

<table>
<thead>
<tr>
<th>Size</th>
<th>Number of Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>less than 2,000 nodes</td>
</tr>
<tr>
<td>Medium</td>
<td>2,000 to 4,000 nodes</td>
</tr>
<tr>
<td>Large</td>
<td>greater than 4,000 nodes</td>
</tr>
</tbody>
</table>

Table 3-1 Configuration Model Size
3.2.4 Option Features and Options

An Option Feature is a Feature whose data type is a List of Options.

- Option Features with large numbers of Options can cause slow performance.
  In the Oracle Configurator engine, propagation of a rule that has as a participant one of the Options of an Option Feature triggers an evaluation of all that Feature’s other Options. This does not apply to Option Classes and Standard Items in an Oracle BOM.
  A possible solution is to use a Component having Boolean Features instead of Option Features. Another alternative is to use BOM structure for all selectable Options.
  Option Features with a very large number of Options (hundreds) and MAX value can cause slow performance. This also applies to Option Classes and Standard Items in an Oracle BOM.

The best way to improve performance is to restructure the Model or BOM Model to contain Option Features or Options Classes with fewer Options or Items. If your business requirements demand large numbers of options, consider setting MAX to no value.

In the Oracle Configurator engine, propagation of a rule that has as a participant one of the Options of an Option Feature triggers a count of the current number of selected Options. If the maximum number of Options is reached, this rule propagates Logic False to all unselected Options.

If you need the remaining options to be LFALSE after a maximum is reached, then you must set a MAX and incur the performance cost. However, since both LFALSE (from MAX reached) and UNKOWN states are displayed as Available in the UI, your logic requirements may permit you to set the MAX to no value.

You can limit the number of allowable selections even without setting MAX to a value by creating a rule to count the selected Options:

Always_True REQUIRES Feature.#Selection.#Count <= MAX

This rule expresses a hard constraint to set Options Logic True. In this rule, Feature.#Selection.#Count is an advanced expression of an Option Feature. Always_True is the only Option of a (1,1) Option Feature. MAX is the maximum number of options the end user can select. A maximum number of selections can be enforced by combining it with a Contributes rule.

- Large configuration models with many Options and Features might be inefficient if reaching a solution requires a minimal number of Unknowns.
Design your configuration model so that valid configurations are allowed even when numerous Options are Unknown. That way the CIO and the Oracle Configurator engine process fewer number of delta changes on each end user action. For example, use the suggested approach for enforcing the MAX to minimize the propagated Logic False states, instead of the regular MAX.

If performance is not an issue and it doesn’t matter to your implementation whether Options are Unknown or Logic False, then having them be Logic False is better. Instead of enforcing a MAX, create a separate rule that applies min/max (1,1).

### 3.3 User Interface

For Web-based Oracle Configurators, it is possible to create a user interface several ways:

- generate the UI using the Create New UI command in Oracle Configurator Developer
- customize the generated User Interface (DHTML) in Oracle Configurator Developer
- create an entirely custom UI that accesses the model through custom code invoking the CIO API.

A custom UI is created using tools not available in Oracle Configurator Developer or Oracle Applications and is necessarily an entirely custom project.

The design of the user interface may affect performance. The key areas of influence are:

- **Number and Type of Screens and Controls**
- **Visibility Settings**
- **Memory Usage and Model Load Setting**
- **Graphics**

Additionally, the client browser and server architecture affect UI performance. For details, see Section 2.3.3, "Client Machine Tuning" on page 2-10 and Section 2.3.2, "Server Machine Tuning" on page 2-9.
3.3.1 Number and Type of Screens and Controls

The number of Oracle Configurator screens and the number of UI controls on each screen, influences runtime performance. Increasing the number of controls increases the time needed to render or paint the screen. This is due to browser resource limitations, not an inherent limitation in Oracle Configurator. The recommended maximum number of UI controls per screen is 8 to 10. The recommended maximum number of graphics per screen is 5.

The type of controls on a page does not influence server performance. Dropdown Lists and Selection Lists take the same amount of time to render. When being painted in the client browser, it is possible that Dropdown Lists paint faster than Selection Lists. Painting time of a Selection List UI control is dependent on how many Options are displayed.

When configuring Option Classes with a very large number (for example, hundreds) of Items, the Java applet does not perform faster than the DHTML window. With the DHTML window you can use multiple screens to display Option Classes with large numbers of Items. If you are using guided selling (DHTML only), hide the Items and have the guided selling questions drive the selections.

3.3.2 Visibility Settings

Using dynamic visibility is generally more expensive than allowing all options to be displayed. Hiding unavailable Options incurs additional server processing because Oracle Configurator must distinguish between purely unavailable (false) options and locally negated options (based on the maximum of the OptionFeature).

3.3.3 Memory Usage and Model Load Setting

Depending on the size of the configuration model, initial load could take a long time to render the first screen. It is possible to set load to on-demand or lazy loading, but this delays throughput of individual screens and screen elements, resulting in overall slow performance of Oracle Configurator.

Oracle recommends preloading models. In a production environment operating on nonstop (24X7) schedule, choose times of least end user activity for preloading your configuration models.

For details about loading, preloading, and lazy loading models, see Chapter 4, "Web Server Configuration", in particular Section 4.2.4.4, "cz.uiserver.lazyload" and Section 4.2.4.5, "cz.uiservlet.pre_load_filename".
3.3.4 Graphics

The number and size of GIFs in a page does not increase the time needed to render the screen on the server. Reducing the number of controls and GIFs on the page may improve the performance of painting the browser page on the client machine.

The HTML template affects UI performance. If you modify the template to do more than the default OC HTML template, performance may be worse depending on elements such as screen layout, UI controls, and custom images.

For additional details about changing the HTML template, see the Oracle Configurator Implementation Guide.

3.3.5 Custom User Interface

A custom UI is not created using Create New UI in Oracle Configurator Developer. Since it is created by writing custom code to interface with the Configuration Interface Object (CIO), it is important to optimize the CIO calls to get the best performance. For guidance, see how Functional Companions use CIO calls effectively (Section 3.5, “Functional Companions” on page 3-10).

If you are using a custom UI, note that DHTML controls that use JavaScript for browser events take longer to display on the client than straight HTML does. If the dynamism of JavaScript is not needed in your application, then implementing straight HTML using a custom HTML template could result in a significant performance gain.

3.4 Rules

Rule design has a critical effect on Configurator performance at runtime. The key areas of influence are:

- Number and Complexity of Rules
- Redundancy
- Rule Types and Relations

The number and complexity of the rules affect the scope and complexity of the generated Active Model. Many complex rules slow runtime performance. Complex rules do not slow load time of the configuration model.

Large numbers of rules is a deceptive factor in performance because many rules that use intermediate nodes effectively or avoid unnecessary redundancy can perform better than fewer rules that repeat identical subexpressions or result in cycles.
Similarly, rule complexity is a deceptive factor because the greater complexity introduced by use of intermediate nodes can perform better than the simpler rule design of repeating an identical subexpression in many rules. See Section 3.4.1, "Number and Complexity of Rules" on page 3-8 for details.

For a review of logic states and other information related to rules, see the Oracle Configurator Developer User’s Guide.

### 3.4.1 Number and Complexity of Rules

Complex or large numbers of rules within a configuration model could cause slow performance. Whenever possible, use effectivity dates or Usages to turn off rules that are not necessary in certain contexts of the calling application. That lets the Oracle Configurator engine ignore those rules and propagate among the enabled rules more efficiently. When debugging configuration models, disable rules explicitly at the rule or rule folder level to let the Oracle Configurator engine ignore them.

Large numbers of rules with commonly used subexpressions also cause slow performance. When the Oracle Configurator engine propagates such rules, even if there are commonly used patterns of operators and operands, the engine needs to evaluate and propagate each instance of the common subexpression separately.

Consider Example 3–1 showing two rules with a common calculation contributing to a Resource.

**Example 3–1 Rules With Common Subexpressions**

\[(A + B) * C\] contributes to R1  
\[(A + B) * C\] + D contributes to R2

Create an intermediate node (I1) for the commonly used subexpression and create an intermediate rule that contains the subexpression \((A + B) * C\) and the intermediate node.

\[(A + B) * C\] contributes to I1

The original rules can then be rewritten by replacing the common subexpression with the new node that expresses the intermediary rule.

I1 contributes to R1  
I1 + D contributes to R2
Although this technique creates an additional rule, the net effect of replacing repeating patterns in rules with nodes that represent those patterns is that the Oracle Configurator only computes the subexpression once, thereby reducing the amount of calculation required for each rule that contains the common subexpression.

**WARNING:** When you use the technique of replacing common subexpressions with intermediate nodes, you must customize the violation message of the intermediary rule to explain the contradiction in terms of the rules that contain the intermediary rule. In other words, the default violation message for I1 in Example 3–1, which is displayed to the end user when Oracle Configurator encounters a problem with R1 or R2, describes a contradiction in I1 unless you customize the message to explain the error in terms of R1 and R2.

### 3.4.2 Redundancy

Redundant rules create logic or numeric cycles, which cause slow performance. Example 3–2 shows two rules that would cause the Oracle Configurator engine to cycle.

**Example 3–2  Redundant Rules That Create Cycles**

(A AND B AND C) implies X

((A AND B AND C) OR D) implies X

Use case analysis to explore the possible propagation paths and eliminate the redundant rules. Case analysis allows you to examine the list of all possible inputs and their results, showing where the propagation path does not settle on a result but cycles through the rule again and again. The generic diagram below shows a case analysis where x is the input and f(x) is the result.

<table>
<thead>
<tr>
<th>x</th>
<th>f(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>
When using case analysis, set one node \((x)\) to a state such as Logic True and determine how that affects the states of other nodes \((f(x))\) in the model, thus exposing redundancy.

Optimize the configuration problem to avoid creating rules that only create cycles.

To turn on or off rules during debugging, explicitly disable them at the rule or rule folder level.

### 3.4.3 Rule Types and Relations
When the rule implementers intention is to express compatibility, it is better to use Compatibility rules than Logic rules. The Oracle Configurator engine performs the calculations over a large number of Options faster with Compatibility rules than with any other rule type.

The following rule relations and types typically cause slow performance of the configuration model:

- Defaults rules
- Boolean Features with initial values
- Numeric rules driving quantities of one

Implementing many Defaults rules can be detrimental to runtime performance and you must be careful not to create overlapping conditions. For example, selections initially made by a Defaults rule could be changed by the end user later in the configuration session. This causes the system to reevaluate all parts of the model and reset the logic states of all affected options before the end user can continue. Therefore, Defaults rules should be used as infrequently as possible. In some rare cases, an Implies rule can be used instead of Defaults.

When designing a configuration model, add Default rules at the end of the design process and weigh the performance cost of adding them against the perceived end-user benefit.

Boolean Features with initial values cause slow performance because they behave like Defaults rules.

### 3.5 Functional Companions
The design of Functional Companions and how they interact with the configuration model as well as other software may affect performance.

The key areas of influence are:
### Functional Companions

- Functional Companion Event Sequence
- Adding and Deleting Optional Components
- Optimization of Validation Functional Companions

Functional Companions must use the Configuration Interface Object (CIO) to interact with the configuration model. See Section 2.3.1.2, "Configuration Interface Object (CIO)" on page 2-7 and the Oracle Configuration Interface Object (CIO) Developer’s Guide for additional information about the CIO.

#### 3.5.1 Runtime Nodes

- The method `RuntimeNode.getChildByID()` is likely to cause slow performance if used on any runtime nodes with many children. It is better to use `RuntimeNode.getChildByName()`, which has been optimized.
- Use Java `HashMap` objects to map from names, IDs, and paths to `RuntimeNodes` so that you can use `RuntimeNode.getChildren()`. If `HashMap`s cannot be used, use localized search by starting only at the Components where you can certainly find the nodes. For better performance, use localized search only once.
- Design your structure appropriately to tune the performance. See Section 3.2, "Model Structure" on page 3-2 for details about structure.
- Functional Companions or custom code invoking the CIO that make many state and count assertions can be inefficient. Use `IState.getState()` or `IState.getCount()` and ignore the assertion if the node is already in the desired state or count.

You can also ignore the assertion if the node is in the opposite state, which if asserted may cause a contradiction, and there is no requirement for getting the contradiction message back.

---

**WARNING:** When you use this technique of ignoring assertions, the contradiction is not obvious for opposite states if the contradictory state is due to defaults, in which case the assertion might succeed.

---

- Functional Companions and custom CIO code making many calls to `IState.getState()` to check for certain states can be slow.
Functional Companions

For each node on which to check the state, call `IState.getState()` only once. Then use the following static methods on the returned int:

- `StateNode.isTrueState()`
- `StateNode.isFalseState()`
- `StateNode.isUserState()`
- `StateNode.isLogicState()`
- `StateNode.isUnknownState()`

You can also use the following methods to combine state checks:

- `StateNode.isSelected()`
- `StateNode.isTrue()`
- `StateNode.isFalse()`
- `StateNode.isUser()`
- `StateNode.isLogic()`
- `StateNode.isUnknown()`

Functional Companions and custom CIO code making many calls to `IState.getState()` or `OptionFeature.getSelectedOptions()` to get the Options selected as a result of a transaction can be slow. Instead, use `Configuration.getSelectedItems()`, which uses already-calculated information about the all the TRUE items in the whole configuration.

If this is done in addition to any number of user selections directly to the CIO (setState, setCount, setValue, and so on) use a transaction to wrap all those assertions.

In order for the CIO mechanism to be able to update the selected item list, make sure you commit the transaction before calling `Configuration.getSelectedItems()`.

CIO methods may or may not be expensive, depending on the circumstances of the configuration model. Making many calls to any CIO method that is expensive can be slow. Call the method once and then cache the results for reuse. For example, the information from `Configuration.getAvailableNodes` should be queried only when necessary, such as on screen flips and after user assertions.

Use profiling tools to evaluate the performance cost of called CIO methods.
3.5.2 Functional Companion Event Sequence

Programmatic changes to the configuration model are dependent on the sequence of events. For example, to instantiate some Components and programmatically set some Features in each Component, you could use either of these approaches:

- set the Features as you add each of the Components
- add all Components and then set all the Features

It turns out that the second approach is faster, because the creation of each Component requires the retraction of all previous inputs (all user selections and all default selections). In the second approach, you can assert inputs and defaults only once, between when you add all the Components and when you set all the Features in all the added Components. The expensive events that you are avoiding in this second approach are the assertions and retractions of inputs and defaults, which are processed every time that a Component is added.

3.5.3 Adding and Deleting Optional Components

This section explains performance problems that can arise when adding and deleting several optional Components, and provides a generalized solution, with examples.

Problems

Adding or deleting an instance of a ComponentSet can be very expensive in terms of performance:

- Adding an instance entails a retraction of all previous requests, then the addition and reassertion of the requests.
- Adding an instance is particularly expensive when there are default values set by configuration rules. retracting default assertions is time-consuming and iterative. The initial values set in Configurator Developer for Features, Resources, and Totals should also be regarded as default values.
- Deleting instances is a very expensive operation, and may not be necessary at the end of the Functional Companion event sequence, since the Functional Companion may need to add the same number of Components back into the Component Set anyway.

Solution Guidelines

To avoid these performance problems, follow these guidelines for adding instances of a ComponentSet:
Try to delay setting the states or values of nodes until after you add all your instances. Add an instance when there are as few settings as possible.

Before adding instances, roll back all the requests that the end user made before triggering the Functional Companion (for instance, by clicking the Auto-Configuration button). To ensure retaining the user’s requests correctly, get the nodes’ states or values before you retract the requests. After adding the instances, reapply the requests.

When deleting instances, and if you have retracted all the requests, you can reuse the instantiated Components by computing the number of children in the ComponentSet and keeping track of the index number of the Component that you are setting Feature values for. Then you can delete the Components that you have not reused, if any.

**Strategy for Structure**

Based on the preceding guidelines, the strategy for optimizing the performance of an Auto-Configuration Functional Companion is to structure it according to the following sequence:

1. Store all needed values in temporary variables.
   These are those values that are needed for later calculations and which would be lost by retracting requests.

2. Retract all requests.
   a. Get user requests. Use `Configuration.getUserRequests()`.
   b. Iterate over the user requests, getting the runtime node for each. Use `Request.getRuntimeNode()`.
   c. If these are state nodes, retract them. Use `IState.unset()`.

3. Add or reuse Component instances. Use `ComponentSet.add()`.

4. Reassert applicable requests.
   a. Iterate over the user requests, getting the runtime node for each. Use `Request.getRuntimeNode()`.
   b. If the request is a state request, use `Request.isStateRequest()`.
      Query the state by using `Request.isTrueStateRequest()`, `Request.isFalseStateRequest()`, and so on.
      Set the node value by using `((IState) IRuntimeNode).setState(state)`. 
c. If the request is a numeric request, use Request.isNumericRequest(). Query the value by using Request.getNumericValue().

   Set the node value by using
   IRuntimeNode.setDecimalCount(value),.setCount(...),
   .setIntValue(...),.setDecimalValue(...).

d. If the request is a text request, query it with Request.getValue().

   Set the node value by using IRuntimeNode.setTextValue(value).

5. Set values on Components instances. Use .setState().
6. Delete extra Components

Pseudocode Example
Consider an example of creating instances of a certain number of optional Components, and setting the values of Features in each one. Example 3–3, “Approach A (Slower)” on page 3-15 creates a Component, then sets its Features before creating the next Component. Example 3–4, “Approach B (Faster)” on page 3-15 creates all the Components, then sets all of their Features.

Example 3–3  Approach A (Slower)
Add Component1 to ComponentSet
Set Feature1 in Component1
Set Feature2 in Component1
Add Component2 to ComponentSet
Set Feature3 in Component2
...

Example 3–4  Approach B (Faster)
Add Component1 to ComponentSet
Add Component2 to ComponentSet

Set Feature1 in Component1
Set Feature2 in Component1
Set Feature3 in Component2
...

Approach A (Slower) is subject to the negative performance effects identified in Problems on page 3-13. Approach B (Faster) demonstrates the principles identified in Solution Guidelines on page 3-13. Approach B (Faster) results in significantly faster runtime performance.
The reason why Approach A (Slower) is slower is shown by comparing Example 3–5, "Detailed Approach A (Slower)" on page 3-16 with Example 3–6, "Detailed Approach B (Faster)" on page 3-16. The operations that have to be performed by the CIO when adding ComponentSet instances are highlighted in boldface. Detailed Approach A (Slower) repeats all of the operations for each addition of an instance, so that each addition takes longer than the preceding one, and the total processing time is proportional to the number of instances to be added. If there are many instances to be added, the impact is great. Detailed Approach B (Faster) performs the operations once, greatly lessening the effect of adding many instances.

Example 3–5  Detailed Approach A (Slower)

Retract requests
Add Component1 to ComponentSet

Assert requests
Set Feature1 in Component1
Set Feature2 in Component1
...

Retract requests
Add Component2 to ComponentSet

Assert requests
Set Feature3 in Component2
...

Example 3–6  Detailed Approach B (Faster)

Retract requests
Add Component1 to ComponentSet
Add Component2 to ComponentSet
...

Assert requests
Set Feature1 in Component1
Set Feature2 in Component1
Set Feature3 in Component2
...

Code Example
Consider the simple runtime Model structure shown in Figure 3–1 on page 3-17. At run time, this Model contains a ComponentSet object and several BooleanFeature objects.
Assume that you have written an Auto-Configuration Functional Companion that sets the state of the Boolean Features as illustrated in Example 3–7, "Setting Features (Slower Code)" on page 3-17.

**Example 3–7 Setting Features (Slower Code)**

```java
for(int i=0; i<100; i++) {
    comp = Comp1.add();
    ((BooleanFeature)comp.getChildByName("BoolFeat1")).setState(IState.TRUE);
    ((BooleanFeature)comp.getChildByName("BoolFeat2")).setState(IState.FALSE);
    ((BooleanFeature)comp.getChildByName("BoolFeat3")).setState(IState.TRUE);
}
```

You can significantly improve the performance of this operation by modifying the code as shown in Example 3–8, "Setting Features (Faster Code)" on page 3-17. The differences are highlighted in **boldface**.

**Example 3–8 Setting Features (Faster Code)**

```java
for(int i=0; i<100; i++){
    comp[i] = Comp1.add();
}
for(int i=0; i<100; i++){
    ((BooleanFeature)comp[i].getChildByName("BoolFeat1")).setState(IState.TRUE);
    ((BooleanFeature)comp[i].getChildByName("BoolFeat2")).setState(IState.FALSE);
    ((BooleanFeature)comp[i].getChildByName("BoolFeat3")).setState(IState.TRUE);
}
```
Example 3–8 achieves a performance improvement by adding all the ComponentSet instances, then setting all the BooleanFeature values. This follows the principles identified in Solution Guidelines on page 3-13, and the example shown in Approach B (Faster) under Pseudocode Example on page 3-15.

3.5.4 Optimization of Validation Functional Companions

In general, Oracle Configurator applies validation tests and defaults more often than would be expected. Defaults should be used as judiciously as possible. Validation tests must be minimal. You should arrange the code so that it is only called when necessary, and the condition test is quick.

Example 3–9 and Example 3–10 show two ways of applying a validation test. Example 3–9 performs better because it sets up the validation (finding the node in each call) within the initialize () method rather than within the validation test, and then does the validation only after all other calls are completed.

Example 3–9  Minimizing Validation Test on a Configuration Model

```java
OptionFeature f = null
initialize () {
    f = (OptionFeature) root.getChildByName("feature_name");
}
validate () {
    if (f.isTrue()) {
    }
}
```

Example 3–10  Causing More Validation Tests on a Configuration Model

```java
initialize () {
    // empty
}
validate () {
    OptionFeature f = (OptionFeature) root.getChildByName("feature_name");
    if (f.isTrue()) {
    }
}
```
A well-tuned Web server is very important to get good performance from any Web-based application, including Oracle Configurator. There are several configuration parameters in the Apache Web server and the JServ module, including Configurator-specific parameters, which must be tuned for optimal performance.

Additionally, you can improve performance by load balancing the Apache Web listener (HTTPD) and adjusting heap size during startup.

For instructions in configuring, verifying, and—most importantly troubleshooting—the Apache Web server and the JServ module, see the Oracle Configurator Installation Guide.

See the Section "Structure" on page xviii in the preface to verify the audience for whom this chapter is intended.

### 4.1 Overview

See Section 2.3, "Tuning Key Components of the Oracle Configurator Architecture" on page 2-6 for information about how the Web server participates in a deployed Configurator environment.

Tuning the Web server is an iterative process. Several parameters in various configuration files play a significant role in Oracle Configurator performance. Section 4.2, "Apache Web Server Parameters" presents these parameters, their role in runtime performance, and their default and recommended values. Tuning the Web server consists of tweaking the values appropriately and iteratively.

In addition to setting parameter values in the configuration files, you can load balance the Apache Web listener (HTTP) or install multiple instances of the JServ servlet engine to handle requests from multiple clients. If the Web server is running
on a multi-CPU machine, then it is important to set up the Apache configuration to load balance across the CPUs for optimal performance. For details about how to load balance Apache for Oracle Configurator, see the Oracle Configurator Installation Guide.

4.2 Apache Web Server Parameters

The high performance Apache Web server in the Oracle Internet Application Server (iAS) is part of the Oracle Configurator architecture. Apache Web server parameters are set in the following configuration files:

- The jserv.conf File
- The zone.properties File
- The httpd.conf and httpds.conf Files
- The jserv.properties File

For an overview at a glance of the Apache Web server parameters that affect Oracle Configurator, their recommended values, and the file in which the parameter is set, see Appendix A, "Summary of Web Configuration Parameters".

For details about correct configuration and location of these Web server files for running Oracle Configurator, see the Oracle Configurator Installation Guide.

4.2.1 The jserv.conf File

The jserv.conf file is used to configure Apache and JServ. The following parameters should be checked and set as needed while configuring Apache.

4.2.1.1 ApJServManual

See the Oracle Configurator Installation Guide for basic information about this parameter.

ApJServManual contributes to performance by affecting whether or not you can load balance the Apache Web listener.

Default Value

The ApJServManual parameter is set to off by default. The off value ignores load balancing. When you are using load balancing, this value must be on.
Recommended Value
The recommended value for optimal Oracle Configurator performance is **On**.

4.2.1.2 **ApJServVMTimeout**
See the *Oracle Configurator Installation Guide* for basic information about this parameter, including how to test if the value is appropriate for your server load.

**ApJServVMTimeout** contributes to performance by affecting the amount of time to wait for the JVM to start up or respond. If this value is too low, the JVM shuts down when left idle. Slow or heavily loaded machines benefit from a high value.

Default Value
The **ApJServVMTimeout** parameter is set to 10 seconds by default, which is also the timeout if this parameter is commented out in the **jserv.conf** file.

Recommended Value
The recommended value is **1800** seconds (30 minutes).

4.2.2 **The zone.properties File**
See the *Oracle Configurator Installation Guide* for basic information about this file, including its precedence over and interaction with settings in other configuration files such as **jserv.properties**.

In the **zone.properties** file, there is a **repositories** directive that contains a comma delimited list of servlet repositories controlled by a given servlet zone. Repository is a directory path or Java archive (**.jar** file). Parameters in this file, such as the **session.timeout** parameter apply to the servlets listed.

4.2.2.1 **session.timeout**
The **session.timeout** parameter applies to all sessions or end users on the server and contributes to performance by allowing you to determine the number of milliseconds to wait before invalidating an idle session. An idle session causes some performance loss because the CPU continues to poll the OC Servlet until reaching the **session.timeout** parameter setting.

The **session.timeout** parameter should be set as low (short timeout) as possible on a heavily loaded server. This setting potentially frees up the memory consumed by the configuration session of those end users who became idle, walked away, or killed their browsers. Exercise care to get the right value for the typical maximum end user's length of idle time on the Oracle Configurator window’s pages and any
link flows (such as marketing material) that produce an idle event for the runtime Oracle Configurator.

**Default Value**
The `session.timeout` parameter is set to 180000 milliseconds (30 minutes) by default.

**Recommended Value**
The value of this parameter is generally not changed. There is a small performance loss when restarting an OC Servlet, so 30 minutes is the recommended value.

### 4.2.3 The httpd.conf and httpds.conf Files

See the Oracle Configurator Installation Guide for basic information about these files and how they are used to configure the Apache Web listener in iAS. Both files contain the same information, including the directives: `Port`, `Documented`, `Alias`, and `Includes`.

The difference between these two files is that `httpds.conf` is used when there is a secure socket.

The `httpd.conf` and `httpds.conf` files contain the following parameters that affect performance:

- **Timeout**
- **KeepAlive**
- **MaxKeepAliveRequests**
- **KeepAliveTimeout**
- **MaxSpareServers**
- **MinSpareServers**
- **MaxClients**

#### 4.2.3.1 Timeout

The **Timeout** parameter contributes to performance by allowing you to determine the amount of time in seconds that Apache waits for receipt of TCP packets on a POST request, or the amount of time it takes to receive a GET request. This means that when loading a configuration model, there is no response back to the client until the **Timeout** parameter’s value is reached, or the JServ responds, whichever comes first. With large models loading into the JServ, you will need a
correspondingly long timeout to keep the HTTP listener waiting for a response until
the model is loaded.

**Default Value**
The `Timeout` parameter is set to 300 seconds (5 minutes) by default.

**Recommended Value**
The recommended value is 3600 seconds (60 minutes). The value should be at least
1800 seconds (30 minutes) for adequate Oracle Configurator performance. There is
no performance loss from having the `Timeout` parameter set to a large value,
unless the JServ hangs.

4.2.3.2 **KeepAlive**
The `KeepAlive` parameter provides long-lived HTTP sessions, which allow
multiple requests to be sent over the same TCP connection. Setting this parameter to
`On` can result in up to 50% decrease in latency times for HTML documents
containing many images.

**Recommended Value**
The recommended value is `On`.

4.2.3.3 **MaxKeepAliveRequests**
The `MaxKeepAliveRequests` parameter limits the number of requests allowed
per connection when `KeepAlive` is "On". If `MaxKeepAliveRequests` is set to 0,
unlimited requests are allowed.

**Default Value**
The `MaxKeepAliveRequests` parameter is set to 100 by default.

**Recommended Value**
The recommended value is 0.

4.2.3.4 **KeepAliveTimeout**
The `KeepAliveTimeout` parameter determines the number of seconds Apache
waits for a subsequent request before closing the connection. Once a request has
been received, the timeout value specified by the `Timeout` directive applies.
**Default Value**
The `KeepAliveTimeout` parameter is set to 15 seconds by default.

**Recommended Value**
The recommended value is 15 seconds.

### 4.2.3.5 MaxSpareServers
The `MaxSpareServers` parameter sets the desired maximum number of idle child server processes. An idle process is one that is not handling a request. If more than the number of processes specified by `MaxSpareServers` are idle, then the parent process kills off the excess processes.

Tuning this parameter should only be necessary on very busy sites. Setting this parameter to a large number is almost always inadvisable. This parameter has no effect on a Microsoft Windows platform.

**Recommended Value**
The recommended value is 10 spare child server processes.

### 4.2.3.6 MinSpareServers
The `MinSpareServers` parameter sets the desired minimum number of idle child server processes. An idle process is one that is not handling a request. If fewer than the number of processes specified by `MinSpareServers` are idle, then the parent process creates new child server processes at a maximum rate of one every second.

Tuning this parameter should only be necessary on very busy sites. This parameter has no effect on a Microsoft Windows platform.

**Recommended Value**
The recommended value is 5 spare child server processes.

### 4.2.3.7 MaxClients
The `MaxClients` parameter determines the limit on the total number of servers running, meaning the limit on the number of clients that can simultaneously connect. When this limit is reached clients are locked out, so `MaxClients` should not be set too low. The main purpose of `MaxClients` is to keep a runaway server from taking the system down with it.
The `MaxClients` parameter is often used to prevent an overload of CPU. By limiting the number of users on a given number of JVMs running on a CPU, one can assume that the often fatal 100% CPU utilization is not reached.

**Default Value**
The `MaxClients` parameter is set to 150 by default.

**Recommended Value**
The recommended value is 150.

### 4.2.4 The `jserv.properties` File

The following adjustments and parameters in the `jserv.properties` file are specific to Oracle Configurator and affect runtime performance.

For additional details about the parameters in the `jserv.properties` file and how to set them, see the *Oracle Configurator Installation Guide*.

For better performance of a production system, Oracle recommends the following combination of settings:

- set `cz.uiservlet.dio_share` to `true`
- set `cz.uiservlet.lazyload` to `false`
- set `czuiservlet.pre_load_filename` to preload the configuration model

#### 4.2.4.1 Heap Size

The C and Java code of Oracle Configurator run in the Web server. Therefore, it is worthwhile to consider the heap size allocation during startup.

The initial Java parameters for the Solaris™ Operating Environment should be:

```
-Xms600m; initial heap
-Xmx1500m; max heap
-Xss1m; native stack size
```

These options are non-standard, subject to change, and vary from platform to platform. Users should have a small initial heap and a maximum heap that is smaller than the platform maximum. This is necessary to minimize the chances of running out of memory for the C heap.
4.2.4.2 cz.uiservlet.dio_share

See the Oracle Configurator Installation Guide for basic information about this parameter.

The cz.uiservlet.dio_share parameter contributes to performance by improving the load performance of configuration sessions after the initial load for a given configuration model. The cz.uiservlet.dio_share parameter should be set to true when benchmarking.

The setting of this parameter interacts with the setting of czservlet.pre_load_filename and cz.uiserver.lazyload. For instance, czservlet.pre_load_filename essentially precaches the model, making the initial load no longer than subsequent load times. On the other hand, setting cz.uiserver.lazyload may minimize the advantages of preloading the model.

Setting this parameter to false disables sharing the cached configuration model, which implies that every user experiences the initial load time.

Default Value
The cz.uiservlet.dio_share parameter is set to true by default.

Recommended Value
For better runtime performance, the recommended value is true. Even when this parameter is set to true, testing configuration models from Oracle Configurator Developer always reloads the configuration models so that changes are visible.

4.2.4.3 cz.uiserver.lfalse_is_not_available

See the Oracle Configurator Installation Guide for basic information about this parameter and how it interacts with the Hide Unselectable Controls and Options setting in Oracle Configurator Developer.

The cz.uiserver.lfalse_is_not_available parameter contributes to performance by allowing you to suppress the display of the Logic False icon (such as a red "X") on logic false Options when the parameter is set to false. In complex configurations models, or ones with many Options, displaying such icons may impose some performance cost. However, the default setting (false) causes the OC Servlet to perform additional calculations to determine which options are available.
for selection. This may also affect the runtime performance of configuration models that are complex or have many options.

If you set `cz.uiserver.lfalse_is_not_available` to true, the OC Servlet does not check the logic state of all options in the configuration model to determine which options are available. In this case, all Logic False options appear with the Logic False icon, by default.

Default Value
The `cz.uiserver.lfalse_is_not_available` parameter is set to false by default, meaning Logic False options are displayed as if they have never been selected, meaning as if their logic state were actually UNKNOWN.

Recommended Value
The recommended value of this parameter depends on your configuration model. Less complex models with fewer options likely perform better when the OC Servlet avoids displaying the Logic False icon, even though it does check the state of all options (`cz.uiserver.lfalse_is_not_available = false`). But more complex models with many options likely perform better when the OC Servlet does not have to check availability of all options, but has to display the Logic False icon of logically false options to show what isn’t available (`cz.uiserver.lfalse_is_not_available = true`).

4.2.4.4 `cz.uiserver.lazyload`
See the Oracle Configurator Installation Guide for basic information about this parameter and how to choose its setting based on screen size and usage of your Oracle Configurator.

The `cz.uiserver.lazyload` parameter contributes to performance by determining whether the configuration model is loaded entirely during initialization (slower initial load but faster subsequent screen flips), or lazily and incrementally as controls need to be displayed (faster initial load but slower subsequent screen flips).

Default Value
The `cz.uiserver.lazyload` parameter is set to true by default, meaning the initial load is fast but subsequent screen flips may be slower than if this parameter were set to false.
**Recommended Value**
For better runtime performance, the recommended value of this parameter is \textit{false} if you are preloading the configuration model, meaning initial load may be slower than if this parameter were set to \textit{true}, but subsequent screen flips are fast.

### 4.2.4.5 \texttt{cz.uiservlet.pre_load_filename}
See the \textit{Oracle Configurator Installation Guide} for basic information about this parameter and its interaction with other parameters such as \texttt{cz.uiservlet.dio_share} and \texttt{servlets.startup}.

The \texttt{cz.uiservlet.pre_load_filename} parameter contributes to performance by determining whether the configuration model is cached when the OC Servlet starts up.

\begin{itemize}
  \item \textbf{Note:} If the configuration model is preloaded, the OC Servlet takes longer to start up.
\end{itemize}

**Default Value**
The \texttt{cz.uiservlet.pre_load_filename} parameter is not set by default.

**Recommended Value**
For better runtime performance, the recommended value of this parameter is the absolute path to a file containing the initialization message that preloads the configuration model so the OC Servlet is initialized when the first user starts a configuration session. The parameter \texttt{cz.uiservlet.dio_share} must be set to \textit{true} so the configuration model can be cached.

Do not use the \texttt{Test} button in Oracle Configurator Developer as the source for your initialization message.

### 4.2.4.6 \texttt{cz.activemodel}
See the \textit{Oracle Configurator Installation Guide} for basic information about this parameter, including examples of how settings on the various switches affect pricing and ATP behavior.

The \texttt{cz.activemodel} parameter contributes to performance by allowing you to suppress display of prices, which can be very expensive.

The switches for displaying list prices, discount prices, and ATP should only be turned on if it is essential to display prices on the Oracle Configurator screens.
See also the *Oracle Configurator Implementation Guide* for additional information about pricing in Oracle Configurator.

**Default Value**
The `cz.activemodel` parameter is set to `/nolp | /nodp | /noatp` by default, meaning list pricing, discount pricing, and ATP are *not* displayed.

**Recommended Value**
The recommended values of this parameter are `/nolp | /nodp | /noatp`.

### 4.2.4.7 `cz.uiserver.poll_timeout_applet`
See the *Oracle Configurator Installation Guide* for basic information about this parameter, including how it interacts with `cz.uiserver.check_heartbeat_timeout`.

The `cz.uiserver.poll_timeout_applet` parameter contributes to performance by determining the time interval in milliseconds within which the Java applet client polls the DHTML client to find out from the UI Server session of a DHTML Oracle Configurator window whether the DHTML client browser has been closed (configuration data has been written to the database).

If the timeout is short, then closing the DHTML client browser results in a quicker return to the Forms, because the Java applet’s UI Server checked more often whether to exit to Forms. However, frequent polling slows performance.

If the timeout is long, then performance is better, but it could take a longer time to return to the parent Form after closing the DHTML client browser.

The `cz.uiserver.poll_timeout_applet` parameter setting is only meaningful when you are launching a DHTML Oracle Configurator window from a Forms-based application such as Order Management.

**Default Value**
The `cz.uiserver.poll_timeout_applet` parameter is set to `20000` milliseconds by default.

**Recommended Value**
The recommended range is `30000` to `60000` milliseconds.
4.2.4.8 cz.uiserver.poll_timeout_applet_to_dhtml

See the Oracle Configurator Installation Guide for basic information about this parameter, including how it interacts with cz.uiserver.check_heartbeat_timeout.

The cz.uiserver.poll_timeout_applet_to_dhtml parameter contributes to performance by determining the time interval in milliseconds within which the UI Server session of a DHTML Oracle Configurator window polls the UI Server session of the parent Java applet to find out if the Java applet is still running.

If the timeout is short (frequent polling), then performance is slower but the DHTML session knows sooner if it has been orphaned and no commits to the database will be possible (configuration cannot be saved).

If the timeout is long, then performance is better, but end users could go through long configuration sessions without realizing the DHTML session has been orphaned and they can’t save their configuration.

This is only meaningful when you are launching a DHTML Oracle Configurator window from a Forms-based application such as Order Management.

Default Value

The cz.uiserver.pollTimeoutToDhtml parameter is set to 30000 milliseconds by default.

Recommended Value

The recommended range is 30000 to 60000 milliseconds.

4.2.4.9 cz.uiserver.check_heartbeat_timeout

See the Oracle Configurator Installation Guide for basic information about this parameter, including what heartbeat events are and how the parameter interacts with cz.uiserver.poll_timeout_applet, cz.uiserver.poll_timeout_applet_to_dhtml, and cz.uiserver.heartbeat_interval.

The cz.uiserver.check_heartbeat_timeout parameter contributes to performance by determining the timeout for the UI Server’s checking of heartbeat events. If the UI Server doesn’t receive any heartbeats from the DHTML client browser after this time value, then the UI Server shuts itself down and the guided selling session in the DHTML user interface sends a terminate message back to the Java applet client.
The `cz.uiserver.check_heartbeat_timeout` parameter is meaningful only when you are launching a DHTML Oracle Configurator window from a Forms-based application such as Order Management.

**Default Value**
The `cz.uiserver.check_heartbeat_timeout` parameter is set to 30000 milliseconds by default.

**Recommended Value**
If you are loading a large configuration model in a DHTML Oracle Configurator window, this parameter should be set to a value close to the time it takes to load the configuration model. For example, if the configuration model takes 60 seconds to load, the value of the parameter should be set to approximately 60000 milliseconds.

### 4.2.4.10 `cz.activemodel.lazyloadlistprice`

See the *Oracle Configurator Installation Guide* for basic information about this parameter.

The `cz.activemodel.lazyloadlistprice` parameter contributes to performance by determining whether the initial load or incrementally displaying a screen includes fetching list prices from the database. Setting this parameter to `false` causes eager fetching of list prices, which can impose a significant performance cost at initial load, especially if your configuration model is large.

**Default Value**
The default value of this parameter is `true`.

**Recommended Value**
The recommended value of this parameter is `true`.

### 4.2.5 Pre-Loading the Oracle Configurator Servlet

You can improve performance when initializing the Oracle Configurator Servlet by preloading your Active Models. Initializing means loading an Active Model for the first time during an OC Servlet session. When preloading Models:

- Create a text file containing an initialization message. For details about getting the OC Servlet to process this file, see the information in the *Oracle Configurator*
Installation Guide about the OC Servlet property `cz.uiservlet.pre_load_filename`.

When composing your initialization message:

1. Omit the pricing callback parameters (`pricing_package_name, configuration_session_key, price_multi_item_proc`).
2. Omit the ATP callback parameters (`atp_package_name, configuration_session_key, get_atp_dates_proc, requested_date`).
3. Omit the parameter `config_creation_date`. This will ensure that you always preload the latest published Active Model. The date will default to the system date (SYSDATE).
4. Either of the parameters `alt_database_name` or `database_id` can be used for preloading.

If loading Models for specific languages, then `database_id` and the `icx_session_ticket` parameter are required for the database session in that language.

1. When preloading set the OC Servlet property `cz.uiserver.lazyload=false`. This ensures that all the screens in your Model’s user interface are also cached at startup.

### 4.2.6 Load Testing

To prepare your application for realistic conditions, you should use a testing tool that allows you to set up reasonable tests that emulate what your end users will do. There are various types of testing you should employ, such as:

- Section 4.2.6.1, "Stress Testing"
- Section 4.2.6.2, "User Activity Testing"

#### 4.2.6.1 Stress Testing

In stress testing, the testing tool fires as many commands as possible at the server, to test CPU utilization rate. Try to measure the number of hits per second you can trigger while keeping the load at 80-85% CPU utilization, which is a recommended level.
4.2.6.2 User Activity Testing
In user activity testing, you use the testing tool to build scripts that represent what your real end users will be doing, with the kinds of delays that a user would experience.

After creating scripts that emulate users, run that number of users against the server, at randomly sequenced times, so that you represent a close approximation of what the application will handle.

Examples
- The user might pause and click Next, Back, and Next several times.
- The user might click a number of checkboxes rapidly, because he knows what he wants, since he has previously visited that part of your application.
- The user might pause, thinking, between each click.

The variability of these different scripts, all running within the same time span, gives you a good sense of what your system can actually handle in terms of number of users.

4.2.7 Load Balancing
See the Oracle Configurator Installation Guide for details about the mechanics of load balancing the machines and processes in your deployment.
The goal of collecting data is to determine whether the production architecture supports all possible end users accessing every available configuration model.

Use a benchmarking tool to test the performance of your configuration model(s) and determine software and hardware sizing that is appropriate for the full range of possible end user scenarios.

---

**Note:** Benchmarking tools cannot account for the client-side rendering time, which may be very important.

---

The sections in this chapter are:

- Data Collection and Analysis
- Data Collection Timing and Scope
- Tool: LoadRunner

See the Section "Structure" on page xviii in the preface to verify the audience for whom this chapter is intended.

### 5.1 Data Collection and Analysis

There is a wide range of data that you can collect for analysis to identify modifications that might or have been proven to improve performance.

### 5.1.1 Areas To Examine

Areas that may yield useful data for performance analysis include:

- database or database server size and setup
Another area that may yield important data for analysis is end-user response time. End-user response time is the sum of the server response time, the display time, network latency, and think time. This is the total throughput under load. There is an amount of time that end users will wait for an application to respond before moving on to something else. That end-user walk away time depends on the business and the task. The end-user response time must be less than the end-user walk away time.

For a more specific listing of areas to examine, see Section 2.3, "Tuning Key Components of the Oracle Configurator Architecture" on page 2-6 and Section 2.4, "Isolating Performance Problems and Intervening" on page 2-11.

5.1.2 Benchmarking

Benchmarks allow you to compare performance measurements across changes in the configuration model, machine and server configuration, and other elements affecting performance.

Benchmarks also allow you to size your hardware appropriately to best match your current requirements, as well as handle your future requirements.

The results of benchmarks referred to in this guide are relevant to Oracle Configurator only (see Appendix B, "Case Studies").

Planning and executing the appropriate benchmark tests of any distributed computing system is both critical and complex. Fundamentally, any benchmark you undertake should reflect the actual deployment environment. This means that it should be carried out:

- using the appropriate server-class hardware platform
- over a long duration
- with multiple end users

For example, if your deployment involves dial-up users, your benchmark should test for the slowest expected dial-up speeds.
Conducting a benchmark involves answering the following:

- What is important to measure?
- How are the results to be interpreted?
- What is the minimum necessary test duration?
- How often should the measurements be sampled?
- How can a realistic deployment be constructed for the purposes of measuring a benchmark?

### 5.1.3 Analyzing Data

You analyze data to determine performance and where to tune your system. The data comes from your careful documentation of benchmark test conditions and results, including the data collected by an analysis tool such as LoadRunner. Data analysis often involves making calculations that provide the limits within which your system can perform acceptably, such as the number of concurrent sessions a single CPU can support.

When analyzing your data, keep in mind that approximately 30% CPU utilization needs to be available for running operating system processes. That is equivalent to tuning the runtime Oracle Configurator performance to 70% CPU utilization. Tests with different numbers of end users can help you determine how many end users 70% CPU utilization supports. For examples of CPU utilization data, see Appendix B, “Case Studies”. For details about how components of the Oracle Configurator architecture compete for CPU utilization, see Section 2.3.2, “Server Machine Tuning” on page 2-9 and Section 2.4.1, “System Transactions” on page 2-11.

You can narrow down what runtime events require large numbers of transactions, and thus processing, by comparing the number of transactions logged by your

---

**WARNING:** Do not benchmark using the Test button in Oracle Configurator Developer. The Test button invokes initialization code that sets the internal equivalent of the `cz.ui.servlet.dio_share` parameter to false, which causes every end user to experience the initial load time. Using an HTML launch page or the host application to test Oracle Configurator requires explicitly setting the `cz.ui.servlet.dio_share` parameter.

---

Tools and Collecting Data  5-3
analysis tool. See Table B–4, "Case A Server Performance Times" and Table B–9, "Case B Server Performance Times" for examples.

When determining total client rendering time from your raw data, it is necessary to add the elapsed model update time to the elapsed layout time. See Section 5.2.2, "Client Side Data" on page 5-4 for details about collecting this data.

5.2 Data Collection Timing and Scope

You should conduct tests that include the hosting application.

Collect performance numbers for individual operations within a configuration session, such as screen flips and option clicks.

Collect data when the most useful performance data occurs, which is when 90% of the end users are actively engaged in a configuration session. For this reason, the case studies documentation shows the response time on various operations for the 90th percentile of the full load (see for example, Table B–4, "Case A Server Performance Times").

When an application goes into production, it will likely be operating on a nonstop (24X7) schedule. Any benchmark you undertake should be designed to reflect these requirements, and as such should be run over a period of time. A useful benchmark should run at least 24 hours, longer if possible.

You should conduct tests that include the hosting application, such as Store.

5.2.1 Server Side Data

While collecting server side performance numbers, keep several recommendations in mind:

1. CPU utilization should not exceed 70%
2. If the end-user response time is unacceptable when CPU utilization is below 70%, decrease concurrent users or repeat the tuning process to obtain better response times.

For additional details about collecting server side data, see Section 5.3, "Tool: LoadRunner" on page 5-6.

5.2.2 Client Side Data

You can collect client side rendering time by setting the debug mode to true in the czSource.htm file (set bDebugMode = true).
When `bDebugMode = true`, a window opens on the client machine (see Figure 5–1). For each action taken in the Oracle Configurator window, the debug mode displays the following in the `czSource.htm` file.

**Note:** Since setting `bDebugMode = true` involves modifying the `czSource.htm` file on the middle tier where Oracle Configurator is installed, it is best to collect this client-side data in a test environment.

**Elapsed Time Laying Out Controls and Screens**

Elapsed Layout Time in the `czSource.htm` file is the time taken to paint the UI controls and screens.

**Elapsed Time Updating Model**

Elapsed Model Update Time in the `czSource.htm` file is the time taken to update the UI controls with values.

**Elapsed Time Sending Message to Server**

Elapsed sending message to server time in the `czSource.htm` file is the time taken to send a message from the client browser to the Web server (network time).

The Total Client Rendering Time is the sum of Elapsed Layout Time and the Elapsed Model Update Time.
You will have to record which rtid in the czSource.htm file maps to which end user action in the user interface. The Begin Sending message to server times, both at session initialization and at the submit or done action, are partial times, so are not as directly meaningful as the elapsed times on option clicks and screen flips.

During a LoadRunner benchmark test, set the debug mode to false in the czSource.htm file (set bDebugMode = false). LoadRunner can only simulate end users and does not start an interactive client session, so the client side data cannot be captured during a benchmark test.

### 5.3 Tool: LoadRunner

LoadRunner is one of the many tools used to perform load testing. Load testing involves stressing the product or observing server side performance under specific loads. You can use LoadRunner to collect performance numbers for individual operations within a configuration session.

Use LoadRunner’s Analysis tool to get such data as:

- Response time
- CPU utilization
- Number of transactions per hour
- Throughput
- Hits per second

Figure 5–2  Screen Capture of LoadRunner CPU Utilization Analysis
5.3.1 Settings Specific to Oracle Configurator

See Appendix C, "LoadRunner Settings" for details about setting LoadRunner parameters that are specific to Oracle Configurator and necessary for collecting Oracle Configurator performance data.

5.3.2 General Tips and Guidelines

Load simulation does not require running a wide variety of test cases. Select a group of test cases that represent different usages of the application (such as configuring small, medium, and large systems). There is no need to have multiple test cases representing different input values for one particular use of the application.

Use only test cases that succeed in saving a configuration because each user should go through multiple iterations of the test scripts.

Have written test plans to follow because several recording iterations may be required and repeatability is critical to the correctness of the scripts. During
LoadRunner recording sessions, you see the captured commands inserted in the Vuser scripts in real time as you walk through the test plan.

To execute the Vuser scripts for Oracle Configurator testing, see Appendix C, "LoadRunner Settings".
Tool: LoadRunner
Summary of Web Configuration Parameters

Table A–1 lists all the parameters described in Chapter 4, "Web Server Configuration" in alphabetical order, indicating the recommended value, what file contains the parameter, and where to turn for additional information.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Recommended Value</th>
<th>File</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApJServManual</td>
<td>On</td>
<td>jserv.conf</td>
<td>Section 4.2.1.1 on page 4-2</td>
</tr>
<tr>
<td>ApJServVMTimeout</td>
<td>1800</td>
<td>jserv.conf</td>
<td>Section 4.2.1.2 on page 4-3</td>
</tr>
<tr>
<td>cz.activemodel</td>
<td>/no1p</td>
<td>/nodp</td>
<td>/noatp</td>
</tr>
<tr>
<td>cz.activemodel.lazyloadlistprice</td>
<td>true</td>
<td>jserv.properties</td>
<td>Section 4.2.4.10 on page 4-13</td>
</tr>
<tr>
<td>cz.uiservlet.dio_share</td>
<td>true</td>
<td>jserv.properties</td>
<td>Section 4.2.4.2 on page 4-8</td>
</tr>
<tr>
<td>cz.uiserver.check_heartbeat_timeout</td>
<td># of milliseconds it takes to load the model</td>
<td>jserv.properties</td>
<td>Section 4.2.4.9 on page 4-12</td>
</tr>
<tr>
<td>cz.uiserver.lazyload</td>
<td>false</td>
<td>jserv.properties</td>
<td>Section 4.2.4.4 on page 4-9</td>
</tr>
<tr>
<td>cz.uiserver.false_is_not_available</td>
<td>false</td>
<td>jserv.properties</td>
<td>Section 4.2.4.3 on page 4-8</td>
</tr>
<tr>
<td>cz.uiserver.poll_timeout_applet</td>
<td>20000 to 60000</td>
<td>jserv.properties</td>
<td>Section 4.2.4.7 on page 4-11</td>
</tr>
<tr>
<td>cz.uiserver.poll_timeout_applet_to_dhtml</td>
<td>30000 to 60000</td>
<td>jserv.properties</td>
<td>Section 4.2.4.8 on page 4-12</td>
</tr>
</tbody>
</table>
Table A–1  Web Server Parameters That Affect Oracle Configurator Performance

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Recommended Value</th>
<th>File</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>cz.uiservlet.pre_load_filename</td>
<td>text file with initialization message</td>
<td>jserv.properties</td>
<td>Section 4.2.4.5 on page 4-10</td>
</tr>
<tr>
<td>KeepAlive</td>
<td>On</td>
<td>httpd.conf httpds.conf</td>
<td>Section 4.2.3.2 on page 4-5</td>
</tr>
<tr>
<td>KeepAliveTimeout</td>
<td>15</td>
<td>httpd.conf httpds.conf</td>
<td>Section 4.2.3.4 on page 4-5</td>
</tr>
<tr>
<td>MaxClients</td>
<td>150</td>
<td>httpd.conf httpds.conf</td>
<td>Section 4.2.3.7 on page 4-6</td>
</tr>
<tr>
<td>MaxKeepAliveRequests</td>
<td>0</td>
<td>httpd.conf httpds.conf</td>
<td>Section 4.2.3.3 on page 4-5</td>
</tr>
<tr>
<td>MaxSpareServers</td>
<td>10</td>
<td>httpd.conf httpds.conf</td>
<td>Section 4.2.3.5 on page 4-6</td>
</tr>
<tr>
<td>MinSpareServers</td>
<td>5</td>
<td>httpd.conf httpds.conf</td>
<td>Section 4.2.3.6 on page 4-6</td>
</tr>
<tr>
<td>session.timeout</td>
<td>1800000</td>
<td>zone.properties</td>
<td>Section 4.2.2.1 on page 4-3</td>
</tr>
<tr>
<td>Timeout</td>
<td>3600</td>
<td>httpd.conf httpds.conf</td>
<td>Section 4.2.3.1 on page 4-4</td>
</tr>
</tbody>
</table>
When performance testing new configuration models, the performance data collected on similar, already tuned models can be a guide in setting your success criteria.

This appendix presents several case studies of models that have been tested by Oracle using LoadRunner.

---

**Note:** The models used in these studies were not necessarily optimized for best design or peak performance, and are not necessarily recommended by Oracle as examples of how you should design your model.

---

These studies demonstrate

- what kind of performance you can expect given a similar model and a similar layout (controls per screen)
- what kind of data you should collect to evaluate performance, and how to collect it

In addition, this appendix lists a workable Web server, database server and client machine configuration used by Oracle in internal testing, and intended here as a guide for a benchmarking environment setup.

See [Section 3.2.1, "Size"](#) on page 3-2 for a general definition of small and medium configuration models in numbers of nodes.

See [Chapter 5, "Tools and Collecting Data"](#) for additional information about collecting this kind of data and analyzing it.
See the Section “Structure” on page xviii in the preface to verify the audience for whom this appendix is intended.

B.1 Machine Configuration for Cases A and B

This section lists a workable Web server, database server and client machine configuration used by Oracle in internal testing, and intended here as a guide for your environment setup. These specifications are valid for running and testing Oracle Configurator Release 11i, patchset E.

B.1.1 Web Server

The following is the Web server configuration used in internal testing at Oracle:

- Sun E420 with 4 x 400-Mhz UltraSPARC CPUs
- 4 Gbytes of RAM
- Sun Solaris™ Operating Environment 2.6
- Apache Web server installed on RAID5 file system

B.1.2 Database Server

The following is the database server configuration used in internal testing at Oracle:

- Sun Ultra-60 with 2 x 400-Mhz UltraSPARC CPUs
- 1 Gbyte of RAM
- Sun Solaris™ Operating Environment 2.6
- Database files striped across 3 files systems on software RAID 0+1 with 64k stripe width

B.1.3 LoadRunner Client Machine

The following is the client machine configuration used for running LoadRunner internally at Oracle:

- Dell PowerEdge 2300 with 4 x 500-Mhz CPUs
- 1 Gbyte of RAM
- MS Windows NT 4.0
- Citrix 4.20
For details about the LoadRunner parameters that are specific to Oracle Configurator and necessary for collecting Oracle Configurator performance data, see Appendix C, "LoadRunner Settings".

**B.1.4 Software**

The following is the software used for running LoadRunner tests on Oracle Configurator internally at Oracle:

- Oracle Configurator (Patchset E, Build 15-85)
- Oracle8 Server 8.1.6
- iAS 1.0.2 (Apache Server 1.3.9)
- JDK 1.3 using Java “green” threads
- Mercury LoadRunner 6.5

**B.1.5 Network**

During these LoadRunner tests, the Web server, database server, and LoadRunner client machine all resided in the same building with fiber optic network connection.

**B.2 Case A: Small Configuration Model**

The response time data presented in this case study is specific to the model, UI, and Oracle Configurator release used in the benchmark. See Section B.1.4, "Software" for the Oracle Configurator version used in this benchmark.

Company A is deploying a Web based application that is used to configure a multiplexer. This application is used internally by the company for their sales group to configure the system. The configuration model is presented below by listing the model structure nodes, rules, and User Interface (UI) display elements. The performance data collected on this model is CPU utilization, number of transactions in an hour, and 90th percentile response times (see Table B–4).

Table B–1 describes the product structure in the Oracle Configurator schema as a list of node types and the number of records of each.

<table>
<thead>
<tr>
<th>Table B–1</th>
<th>Case A Model Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Node Type</strong></td>
<td><strong>Records</strong></td>
</tr>
<tr>
<td>BOM Model nodes</td>
<td>6</td>
</tr>
</tbody>
</table>
Table B–1  Case A Model Structure

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOM Option Class nodes</td>
<td>36</td>
</tr>
<tr>
<td>BOM Standard Item nodes</td>
<td>229</td>
</tr>
<tr>
<td>Component nodes</td>
<td>4</td>
</tr>
<tr>
<td>Feature (Boolean)</td>
<td>8</td>
</tr>
<tr>
<td>Feature (Integer)</td>
<td>37</td>
</tr>
<tr>
<td>Feature (Listed Options)</td>
<td>24</td>
</tr>
<tr>
<td>Feature (String)</td>
<td>5</td>
</tr>
<tr>
<td>Option</td>
<td>111</td>
</tr>
<tr>
<td>Product</td>
<td>1</td>
</tr>
<tr>
<td>Resources</td>
<td>9</td>
</tr>
<tr>
<td>Root nodes</td>
<td>1</td>
</tr>
<tr>
<td>Totals</td>
<td>6</td>
</tr>
<tr>
<td>Records in Project</td>
<td>470</td>
</tr>
<tr>
<td>BOM nodes</td>
<td>271</td>
</tr>
<tr>
<td>Non-BOM nodes</td>
<td>199</td>
</tr>
</tbody>
</table>

Table B–2 lists the types and numbers of rules in the model. The Auto-Configuration Functional Companion sets values and adds a new Component that needs to be configured.

Table B–2  Case A Rule Types and Operators

<table>
<thead>
<tr>
<th>Rule Type</th>
<th>Operator</th>
<th>Number of Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison Rule</td>
<td>Implies</td>
<td>5</td>
</tr>
<tr>
<td>Logic Rule</td>
<td>Defaults</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Excludes</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>Implies</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>Negates</td>
<td>1</td>
</tr>
<tr>
<td>Numeric Rule</td>
<td>Requires</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Consumes</td>
<td>74</td>
</tr>
</tbody>
</table>
The following table displays the number and type of UI controls in the application.

**Table B–3  Case A User Interface Controls**

<table>
<thead>
<tr>
<th>Screen</th>
<th>Controls</th>
<th>Clicks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Screen</td>
<td>1 Functional Companion button</td>
<td>1</td>
</tr>
<tr>
<td>First Screen</td>
<td>4 Dropdown elements</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1 Checkbox element</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1 Title bitmap</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 Text element</td>
<td>0</td>
</tr>
<tr>
<td>Second Screen</td>
<td>6 Dropdown elements</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>3 Checkbox elements</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1 Selection list with 22 options</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1 Title bitmap</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 Text element</td>
<td>0</td>
</tr>
<tr>
<td>Third Screen</td>
<td>9 Dropdown elements</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>2 Checkbox elements</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1 Functional Companion Delete button</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>7 Numeric edit control elements</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>7 Resource value displays</td>
<td>0</td>
</tr>
</tbody>
</table>
Case A: Small Configuration Model

Table B–4 and Table B–5 describe the server performance times for the application. Case A performed at 83% CPU utilization with 100 users. In order to determine the total throughput time, you must add the client response time (not provided here) to the server response time.

### Table B–4  Case A Server Performance Times

<table>
<thead>
<tr>
<th>Users</th>
<th>Operations</th>
<th>Arrival Rate (transactions/hour)</th>
<th>90th Percentile Response Times in Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>60 User Scenario</td>
<td>Initialization</td>
<td>918</td>
<td>2.091</td>
</tr>
<tr>
<td></td>
<td>Instantiation</td>
<td>2,450</td>
<td>0.854</td>
</tr>
<tr>
<td></td>
<td>Screen Flip</td>
<td>7,612</td>
<td>0.341</td>
</tr>
<tr>
<td></td>
<td>Option Click</td>
<td>32,906</td>
<td>0.341</td>
</tr>
<tr>
<td></td>
<td>Save Configuration</td>
<td>918</td>
<td>5.59</td>
</tr>
<tr>
<td>80 User Scenario</td>
<td>Initialization</td>
<td>1,254</td>
<td>2.791</td>
</tr>
<tr>
<td></td>
<td>Instantiation</td>
<td>3,774</td>
<td>1.059</td>
</tr>
<tr>
<td></td>
<td>Screen Flip</td>
<td>4,7758</td>
<td>0.414</td>
</tr>
<tr>
<td></td>
<td>Option Click</td>
<td>11,296</td>
<td>0.414</td>
</tr>
<tr>
<td></td>
<td>Save Configuration</td>
<td>1,254</td>
<td>8.31</td>
</tr>
<tr>
<td>100 User Scenario</td>
<td>Initialization</td>
<td>1,494</td>
<td>3.29</td>
</tr>
<tr>
<td></td>
<td>Instantiation</td>
<td>4,570</td>
<td>1.119</td>
</tr>
<tr>
<td></td>
<td>Screen Flip</td>
<td>57,718</td>
<td>0.484</td>
</tr>
<tr>
<td></td>
<td>Option Click</td>
<td>13,680</td>
<td>0.484</td>
</tr>
<tr>
<td></td>
<td>Save Configuration</td>
<td>1,494</td>
<td>11.94</td>
</tr>
</tbody>
</table>

Table B–3  Case A User Interface Controls

<table>
<thead>
<tr>
<th>Screen</th>
<th>Controls</th>
<th>Clicks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 Title bitmap</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>5 Text labels</td>
<td>0</td>
</tr>
</tbody>
</table>
B.3 Case B: Medium Configuration Model

The response time data presented in this case study is specific to the model, UI, and Oracle Configurator release used in the benchmark. See Section B.1.4, "Software" for the Oracle Configurator version used in this benchmark.

Company B is deploying a Telesales application where an end user can call in and order a pizza over the phone. A sales representative takes the call and helps the end user configure a pizza. The configuration model is presented below by listing the model structure nodes, rules, and User Interface (UI) display elements. The performance data collected on this model is CPU utilization, number of transactions in an hour, and 90th percentile response times (see Table B–9).

Table B–6 describes the product structure in the Oracle Configurator schema.

<table>
<thead>
<tr>
<th>Table B–6 Case B Model Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node Type</td>
</tr>
<tr>
<td>BOM Model nodes</td>
</tr>
<tr>
<td>BOM Option Class nodes</td>
</tr>
<tr>
<td>BOM Standard Item nodes</td>
</tr>
<tr>
<td>Component nodes</td>
</tr>
<tr>
<td>Feature (Boolean)</td>
</tr>
<tr>
<td>Feature (Integer)</td>
</tr>
<tr>
<td>Feature (Listed Options)</td>
</tr>
<tr>
<td>Feature (String)</td>
</tr>
<tr>
<td>Option</td>
</tr>
<tr>
<td>Product</td>
</tr>
<tr>
<td>Resources</td>
</tr>
</tbody>
</table>

Table B–5 Case A CPU Utilization

<table>
<thead>
<tr>
<th></th>
<th>60 User Scenario</th>
<th>80 User Scenario</th>
<th>100 User Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>60%</td>
<td>68%</td>
<td>76%</td>
</tr>
<tr>
<td>Min</td>
<td>36%</td>
<td>48%</td>
<td>62%</td>
</tr>
<tr>
<td>Average</td>
<td>42%</td>
<td>53%</td>
<td>54%</td>
</tr>
</tbody>
</table>
Table B-6  Case B Model Structure

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root nodes</td>
<td>1</td>
</tr>
<tr>
<td>Totals</td>
<td>6</td>
</tr>
<tr>
<td>Records in Project</td>
<td>1241</td>
</tr>
<tr>
<td>BOM nodes</td>
<td>619</td>
</tr>
<tr>
<td>Non-BOM nodes</td>
<td>622</td>
</tr>
</tbody>
</table>

Table B-7 lists the types and numbers of rules in the model.

Table B-7  Case B Rule Types and Operators

<table>
<thead>
<tr>
<th>Rule Type</th>
<th>Operator</th>
<th>Number of Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison Rule</td>
<td>Implies</td>
<td>1</td>
</tr>
<tr>
<td>Logic Rule</td>
<td>Defaults</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>Excludes</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>Implies</td>
<td>468</td>
</tr>
<tr>
<td></td>
<td>Negates</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Requires</td>
<td>72</td>
</tr>
<tr>
<td>Numeric Rule</td>
<td>Consumes</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>Contributes</td>
<td>64</td>
</tr>
<tr>
<td>Functional Companions</td>
<td>N/A</td>
<td>0</td>
</tr>
</tbody>
</table>

Table B-8 displays the number and type of UI controls in the application.

Table B-8  Case B User Interface Controls

<table>
<thead>
<tr>
<th>Screen</th>
<th>Controls</th>
<th>Clicks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Screen</td>
<td>4 Buttons</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1 Title bitmap</td>
<td>0</td>
</tr>
<tr>
<td>Pizza1 Screen</td>
<td>1 Title bitmap</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>4 Buttons</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3 Dropdown elements</td>
<td>3</td>
</tr>
</tbody>
</table>
Table B–9 and Table B–10 describe the server performance times for the application. Case B performed at 85% CPU utilization with 60 users. In order to determine the total throughput time, you must add the client response time (not provided here) to the server response time.

Table B–9 Case B Server Performance Times

<table>
<thead>
<tr>
<th>Users</th>
<th>Operations</th>
<th>Arrival Rate (transactions/hour)</th>
<th>90th Percentile Response Times in Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single User Scenario</td>
<td>Initialization</td>
<td>1.03</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>0.23</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Screen Flip</td>
<td>0.39</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Option Click</td>
<td>0.22</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Save Configuration</td>
<td>1.14</td>
<td></td>
</tr>
<tr>
<td>40 User Scenario</td>
<td>Initialization</td>
<td>2,225</td>
<td>1.27</td>
</tr>
<tr>
<td></td>
<td>Instantiation</td>
<td>2,225</td>
<td>0.38</td>
</tr>
<tr>
<td></td>
<td>Screen Flip</td>
<td>13,366</td>
<td>0.48</td>
</tr>
<tr>
<td></td>
<td>Option Click</td>
<td>24,516</td>
<td>0.30</td>
</tr>
</tbody>
</table>
### Table B–9  Case B Server Performance Times

<table>
<thead>
<tr>
<th>Users</th>
<th>Operations</th>
<th>Arrival Rate (transactions/hour)</th>
<th>90th Percentile Response Times in Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>60 User Scenario</td>
<td>Initialization</td>
<td>3,278</td>
<td>1.50</td>
</tr>
<tr>
<td></td>
<td>Instantiation</td>
<td>3,278</td>
<td>0.42</td>
</tr>
<tr>
<td></td>
<td>Screen Flip</td>
<td>19,702</td>
<td>0.53</td>
</tr>
<tr>
<td></td>
<td>Option Click</td>
<td>36,084</td>
<td>0.33</td>
</tr>
<tr>
<td></td>
<td>Save Configuration</td>
<td>3,278</td>
<td>2.27</td>
</tr>
</tbody>
</table>

### Table B–10  Case B CPU Utilization

<table>
<thead>
<tr>
<th></th>
<th>40 User Scenario</th>
<th>60 User Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>64%</td>
<td>85%</td>
</tr>
<tr>
<td>Min</td>
<td>41%</td>
<td>60%</td>
</tr>
<tr>
<td>Average</td>
<td>47%</td>
<td>68%</td>
</tr>
</tbody>
</table>

### B.4 SQL*Plus Queries For Determining Number of Nodes

The following SQL*Plus queries are useful for calculating the total number of nodes in a configuration model, and the number of different types of nodes in a configuration model.

### B.4.1 Number of Nodes by Type in a Configuration Model

Example B–1 shows the query for determining the number of nodes by type in a configuration model. Table B–11 provides the numeric identification of the node types that need to be inserted in the query.

### Table B–11  Numeric Identification of PS_NODE_TYPE

<table>
<thead>
<tr>
<th>PS_NODE_TYPE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>258</td>
<td>Product</td>
</tr>
<tr>
<td>259</td>
<td>Component</td>
</tr>
<tr>
<td>261</td>
<td>Feature</td>
</tr>
</tbody>
</table>
Example B–1  Determine Number of Nodes by Type in a Configuration Model

SQL> SELECT count(*)
    FROM cz_ps_nodes
    WHERE deleted_flag = '0'
    and ps_node_type = ps_node_type
    START WITH ps_node_id = devl_project_id
    CONNECT BY DECODE(PRIOR ps_node_type,263,PRIOR reference_id, PRIOR ps_node_id) =
    DECODE(PRIOR ps_node_type,263,ps_node_id,parent_id)
    /

In Example B–1, insert the PS_NODE_TYPE number shown in Table B–11 for the variable ps_node_type, and insert the DEVL_PROJECT_ID of the configuration model for the variable devl_project_id (see Example B–4).

Example B–2 shows the query for determining the number of BOM nodes in a configuration mode, and Example B–3 shows the query for determining the number of non-BOM nodes in a configuration model.

Example B–2  Determine Number of BOM Nodes in a Configuration Model

SQL> SELECT count(*)
    FROM cz_ps_nodes
    WHERE deleted_flag = '0'
    and ps_node_type in ( 263,436,437,438)
    START WITH ps_node_id = devl_project_id
    CONNECT BY DECODE(PRIOR ps_node_type,263,PRIOR reference_id, PRIOR ps_node_id) =
    DECODE(PRIOR ps_node_type,263,ps_node_id,parent_id)
    /

Table B–11  Numeric Identification of PS_NODE_TYPE

<table>
<thead>
<tr>
<th>PS_NODE_TYPE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>262</td>
<td>Option</td>
</tr>
<tr>
<td>272</td>
<td>Total</td>
</tr>
<tr>
<td>273</td>
<td>Resource</td>
</tr>
<tr>
<td>436</td>
<td>Root node</td>
</tr>
<tr>
<td>263</td>
<td>BOM Model</td>
</tr>
<tr>
<td>437</td>
<td>BOM Option Class</td>
</tr>
<tr>
<td>438</td>
<td>BOM Standard Item</td>
</tr>
</tbody>
</table>

262 Option
272 Total
273 Resource
436 Root node
437 BOM Option Class
438 BOM Standard Item
Example B–3  Determine Number of non-BOM Nodes in a Configuration Model

SQL> SELECT count(*)
    2 FROM cz_ps_nodes
    3 WHERE deleted_flag = '0'
    4 and ps_node_type not in (263,436,437,438)
    5 START WITH ps_node_id = devl_project_id
    6 CONNECT BY DECODE(PRIOR ps_node_type,263,PRIOR reference_id, PRIOR ps_node_id) =
    7 DECODE(PRIOR ps_node_type,263,ps_node_id,parent_id)

Example B–4  Determine DEVL_PROJECT_ID of a Configuration Model

SQL> SELECT devl_project_id, name
    2 FROM cz_devl_projects
    3 WHERE name like '%Sen%'
    4 and deleted_flag='0'

This returns both the ID and the name:

<table>
<thead>
<tr>
<th>DEVL_PROJECT_ID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020</td>
<td>Sentinel Custom Desktop(204 137)</td>
</tr>
</tbody>
</table>

B.4.2  Number of Features by Type in a Configuration Model

Example B–5 shows the query for determining the number of Features by type in a configuration model.

<table>
<thead>
<tr>
<th>FEATURE_TYPE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Option</td>
</tr>
<tr>
<td>1</td>
<td>Integer</td>
</tr>
<tr>
<td>2</td>
<td>Decimal</td>
</tr>
<tr>
<td>3</td>
<td>Boolean</td>
</tr>
<tr>
<td>4</td>
<td>Text</td>
</tr>
</tbody>
</table>

Example B–5  Determine Number of Features by Type in a Configuration Model

SQL> SELECT count(*)
    2 FROM cz_ps_nodes
WHERE deleted_flag = '0'
and ps_node_type = 261
and feature_type = feature_type
START WITH ps_node_id = devl_project_id
CONNECT BY DECODE(PRIOR ps_node_type,263,PRIOR reference_id, PRIOR ps_node_id) =
DECODE(PRIOR ps_node_type,263,ps_node_id,parent_id)
/

In Example B–5, insert the FEATURE_TYPE number shown in Table B–12 for the variable feature_type, and insert the DEVL_PROJECT_ID of the configuration model for the variable devl_project_id (see Example B–4).

### B.4.3 Number of Rules by Type in a Configuration Model

Example B–6 shows the query for determining the number of rules by type in a configuration model.

<table>
<thead>
<tr>
<th>RULE_TYPE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Logic rule</td>
</tr>
<tr>
<td>22</td>
<td>Numeric rule</td>
</tr>
<tr>
<td>23</td>
<td>Compatibility rule</td>
</tr>
<tr>
<td>24</td>
<td>Compatibility Table</td>
</tr>
<tr>
<td>25</td>
<td>Func Preselect</td>
</tr>
<tr>
<td>26</td>
<td>Func Validate</td>
</tr>
<tr>
<td>27</td>
<td>Comparison rule</td>
</tr>
<tr>
<td>29</td>
<td>Functional Companion</td>
</tr>
<tr>
<td>30</td>
<td>Design Chart</td>
</tr>
</tbody>
</table>

Example B–6  Determine Number of Rules by Type in a Configuration Model

SQL> SELECT count(*)
FROM cz_rules
WHERE rule_type = rule_type
and devl_project_id in (436,263)
SELECT distinct ps_node_id
FROM cz_ps_nodes
WHERE deleted_flag = '0'
and ps_node_type in(436,263)
START WITH ps_node_id = devl_project_id
CONNECT BY DECODE(PRIOR ps_node_type,263,PRIOR reference_id, PRIOR ps_node_id) =
DECODE(PRIOR ps_node_type,263,ps_node_id,parent_id))
/

In Example B–6, insert the RULE_TYPE number shown in Table B–13 for
the variable rule_type, and insert the DEVL_PROJECT_ID of the configuration model
for the variable devl_project_id (see Example B–4).

B.4.4 Number of Rules by Relation in a Configuration Model

Example B–7 shows the query for determining the number of rules by relation type
in a configuration model.

<table>
<thead>
<tr>
<th>EXPR_RULE_TYPE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Requires</td>
</tr>
<tr>
<td>2</td>
<td>Implies</td>
</tr>
<tr>
<td>3</td>
<td>Excludes</td>
</tr>
<tr>
<td>4</td>
<td>Negates</td>
</tr>
<tr>
<td>5</td>
<td>Defaults</td>
</tr>
<tr>
<td>6</td>
<td>NumSelections</td>
</tr>
<tr>
<td>8</td>
<td>Contributes</td>
</tr>
<tr>
<td>10</td>
<td>Consumes</td>
</tr>
</tbody>
</table>

Example B–7  Determine Number of Rules by Relation in a Configuration Model

SQL> SELECT count(*)
FROM cz_rules
WHERE rule_type = rule_type
and expr_rule_type = expr_rule_type
and devl_project_id in {
SELECT distinct ps_node_id
FROM cz_ps_nodes
WHERE deleted_flag = '0'
and ps_node_type in(436,263)
START WITH ps_node_id = devl_project_id
CONNECT BY DECODE(PRIOR ps_node_type,263,PRIOR reference_id, PRIOR ps_node_id) =
DECODE(PRIOR ps_node_type,263,ps_node_id,parent_id))
In Example B–7, insert the RULE_TYPE number shown in Table B–13 for the variable rule_type, insert the EXPR_RULE_TYPE number shown in Table B–14 for the variable expr_rule_type, and insert the DEVL_PROJECT_ID of the configuration model for the variable devl_project_id (see Example B–4).
This appendix describes some LoadRunner parameters that are specific to Oracle Configurator and necessary for collecting Oracle Configurator performance data. The sections are:

- **Virtual User (Vuser) Generator**
- **Vuser Parameterization**
- **Modifying Vuser Scripts**
- **Debugging Vuser Scripts**

This information is based on LoadRunner 6.5. LoadRunner places a load, so must be installed and run on a separate machine from the Web server being tested for load performance.

For information about LoadRunner settings generally, see the LoadRunner documentation listed in the Section “Related Documents” in the preface of this guide.

For additional tips and guidelines on running tests, see Section 5.3.2, "General Tips and Guidelines”.

See the Section "Structure" on page xviii in the preface to verify the audience for whom this chapter is intended.

### C.1 Virtual User (Vuser) Generator

Before recording the Vuser scripts, log in to LoadRunner selecting the Vuser type **WebHTTP/HTML**, and set parameters described in this section. Selecting **WebHTTP/HTML** loads the appropriate libraries for recording Vuser scripts emulating Oracle Configurator users.
C.1.1 Tools Menu: Recording Options

From the menu toolbar, select Tools: Recording Options. Make the following section in the tabs of the Recording Options window.

**Browser Tab**
Specify the path to Netscape or Internet Explorer browser.

**Recording Proxy Tab**
Select Obtain the proxy settings from the recording browser. Make sure your browser proxy is set up correctly, in particular if you need to traverse a firewall.

**Recording Tab**
Select HTTP. Then select Record all HTTP requests as custom requests. No other selections should be checked in this tab.

**Correlation and Advanced Tabs**
No options should be selected in these tabs.

C.1.2 Tools Menu: General Options

From the menu toolbar, select Tools: General Options. Make the following selection in the tabs of the General Options window.

**Display Tab**
Show browser during replay should not be selected. The runtime viewer does not support JavaScript, so it does not correctly display the screens for Oracle Configurator during replay.

C.1.3 Vuser: Runtime Settings

**Runtime Settings** are used to specify the runtime settings for the Vuser scripts. These settings are associated with Vuser scripts. You can modify the runtime settings in this section or from the LoadRunner Controller tool.
From the menu toolbar, select Vuser: Runtime Settings. Make the following selection in the tabs of the Runtime Settings window.

**Iterations Tab**
When debugging or during single user scenarios, initially set **Iteration Count** to 1. When simulating multi-user loads, set **Iteration Count** to a large number such as 100. Set **Iteration Pace** to **As soon as possible**. If iterations can’t be started at a specified time, it may not be effective or useful to select **Start new iteration**.

**Log Tab**
Select **Standard log** initially or during debugging. Disabling logging allows the script to run faster. **Extended log** is not necessary unless you suspect a bug in LoadRunner or need more information for debugging.

**Think Time Tab**
Set **Use a percentage of recorded think time** to the range of 50% to 200% for random think time. During debugging, it is appropriate to select **Ignore think time**.

**Network Tab**
In the section **Timeouts**, set both **Connect timeout** and **Receive timeout** to 300 seconds.

**HTTP Tab**
In the section **HTTP Request Profile**, set **HTTP Version** to **Http 1.0**
Select the appropriate browser software version from the list of values. If the list of values does not contain your browser software version, select **Custom user agent** and enter the appropriate identification. For example, for Netscape 4.73, enter:

Mozilla/4.73 [en] (WinNT; I)

**Http 1.1** is not compatible with the **Advanced Options** used. The custom user agent information is passed to the Apache server so that the right template is returned to the client (such as Internet Explorer or Netscape). This data can be found in the Oracle Configurator session log file during an interactive session (search for **user-agent** in the log).

**Advanced Options Tab**
Select (enable) the following options:

**Analog Mode**
Enable HTTP Keep Alive
Enable loading of web resources
Allow DNS caching
Reset context/state between iterations
Do not select Use WinInet reply engine. This parameter does not work for Oracle Configurator.

Proxy Tab
Select Obtain the proxy settings from the default browser.

General Tab
In the Multithreading section, select Run Vuser as a thread. No other options should be selected.

C.2 Vuser Parameterization
To make the Vuser scripts more portable for tests running on different machines, parameterize the host and port information. See the LoadRunner documentation listed in the Section "Related Documents" in the preface of this guide for details.

C.3 Modifying Vuser Scripts
The following modifications of recorded Vuser scripts are specifically useful to LoadRunner testing of Oracle Configurator.

C.3.1 Parameterize the Session ID
The session ID is created at the Proxy page. Parameterize the session ID by inserting the following line of code right before the line containing the OC Servlet URL proxy, such as oracle.cz.servlet.Proxy:

```java
web_create_html_param("Session_Id","sessionId\" value="","";
```

Refer to the LoadRunner for more help (see "Related Documents" on page xx). For debugging, check if the ID is being populated correctly by inserting the following line after the line containing the OC Servlet URL proxy:

```java
lr_output_message("*********** Session_Id==> \"%s\"");
lr_eval_string ("{Session_Id} " ) ;
```
Example C–1 shows an example of parameterizing the session ID in a LoadRunner Vuser script.

**Example C–1  Obtaining Parameterized Session ID in LoadRunner Vuser Script**

```plaintext
web_create_html_param("Session_Id","sessionId" value="","" );
web_custom_request("oracle.apps.cz.servlet.Proxy",
    "URL=http://{Host}:{Port}/servlet/oracle.apps.cz.servlet.Proxy",
    "Method=GET",
    "TargetFrame=",
    "Resource=0",
    "RecContentType=text/html",
    "SupportFrames=0",
    LAST);
lr_output_message("*********** Session_Id==> "%s"",lr_eval_string("{Session_Id}"));
```

Search for the strings `sessionId` and `session-id` and replace the original value assigned to them (typically DHTML# such as DHTML3) with the parameter `{Session_Id}`. The session ID is used numerous times throughout the script, and each occurrence needs to be changed to reference the parameter.

### C.3.2 Think Times

LoadRunner records all pauses and many `lr_think_time()` commands are inserted into a Vuser script during recording. You only need one `lr_think_time()` command at the beginning of the script and between each transaction, so clean out all the excess think time commands. This better simulates a real user pausing between mouse clicks.

### C.3.3 Use Host and Port Parameter

Replace all occurrences of the hostname and port number with the user-defined Host and Port parameter.

Using a global find and replace, replace `mytestcomputer.mycompany.com` with `{Host}` and `myport` with `{Port}`.

### C.3.4 Parameterize User Name and Password

If multiple database accounts are used to test different schema versions, parameterize the user name and password as well.
C.3.5 Leverage Cookies

Another helpful debugging tool is to print out the cookie, which contains the JServ name. This is helpful when multiple JServs are running. Place the following command before the proxy page.

```c
/* get the cookie which contains the jserv name */
web_create_html_param("jserv_cookie", "JServSessionIdroot=", ";")
```

Example C–2 shows the command to use for printing out the cookie name.

**Example C–2 Code for Printing Out Cookie Names**

```c
/* print the cookie */
lr_output_message("+++++ jserv cookie ==> "%s\", lr_eval_string("{jserv_cookie}"));
```

C.3.6 Optimize Vuser Scripts

The start of each test script’s `Action()` section should contain the following commands:

```c
web_cleanup_cookies();
web_set_max_retries("0");
web_set_timeout(RECEIVE, "300");
```

The `web_cleanup_cookies()` command makes sure the previous sessionID is cleared before starting a new configuration session (before the `Init` transaction).

The `web_set_max_retries("0")` command prevents the Vuser script from trying to connect to the server again if the first connection request failed.

The `web_set_timeout()` command sets the timeout limit (also a runtime setting).

Example C–3 shows how to segment scripts into transactions to get the performance response times for specific actions, such as selecting an Option.

**Example C–3 Transactions in Test Scripts**

```c
lr_think_time( 3 );
lr_start_transaction("OptionClick");
web_add_header("Content-Type", "application/x-www-form-urlencoded");
```
The only way to check correctness of a script’s execution is to put validation checks into the test script itself. For a benchmark, have all configuration sessions end in a saved and valid configuration. Place checks at the end of the script for complete and saved configuration status.

1. Declare the following variables at the beginning of the `Action()` section:
   
   ```
   LPCSTR vuser_group;
   int Vuser;
   ```

2. Before the `Save` transaction, insert the following:
   ```
   /*
   * Determine test success through these variables:
   * config_hdr_id, is_valid, is_complete, and exit_status.
   */
   web_create_html_param("config_hdr_id", "<b>Configuration Header ID:</b>","</p>");
   web_create_html_param("is_valid", "<b>Configuration Valid:</b>","</p>");
   web_create_html_param("is_complete", "<p><b>Configuration Complete:</b>"","</p>");
   web_create_html_param("exit_status", "<b>Exit Status:</b>"","</p>");
   ```

3. After the `Save` transaction, insert the following:
   ```
   /*
   * Test completed successfully only when is_valid is true, is_complete is true, and
   */
   ```
*exit_status is save. Otherwise, the test failed.
* No other verifications are done in the script to cause slow execution.
*/
/* Get vuser number and vuser group name. */
lr_whoami(&vuser, &vuser_group, NULL);
if ( (strcmp(lr_eval_string("{is_valid}"), lr_eval_string("true")) == 0) &&
     (strcmp(lr_eval_string("{is_complete}"), lr_eval_string("true")) == 0) &&
     (strcmp(lr_eval_string("{exit_status}"), lr_eval_string("save")) == 0) )
{
lr_log_message("+++++ Config Hdr ID %s done by VUser %d in group %s.",
               lr_eval_string("{config_hdr_id}"), vuser, vuser_group);
}
else
{
lr_output_message("Verification Failure: VUser %d in group %s failed; %s in
                  %s.",
               vuser, vuser_group, lr_eval_string("{Session_Id}"),
               lr_eval_string("{jserv_cookie}"));
}

Even if the test succeeds, a validation check error message is written to the
LoadRunner Controller window and log file. If there is a way to directly assert a test
failure, place it inside the else statement after the Save transaction in the Vuser
script.
Glossary of Terms and Acronyms

This glossary contains definitions that you may need while working with Oracle Configurator.

**Active Model**
The compiled structure and rules of a *configuration model* that is loaded into memory on the Web server at *configuration session* initialization and used by the *Oracle Configurator engine* to validate runtime selections. The Active Model must be generated either in *Oracle Configurator Developer* or programmatically in order to access the configuration model at runtime.

**API**
Application Programming Interface

**applet**
A Java application running inside a Web browser. *See also* Java and *servlet*.

**application architecture**
The software structure of an application at runtime. Architecture affects how an application is used, maintained, extended, and changed.

**architecture**
*See* application architecture.

**ATO**
Assemble to Order
**ATP**
Available to Promise

**attribute**
The defining characteristic of something. Models have attributes such as Effectivity Sets and Usage. Components, Features, and Options have attributes such as Name, Description, and Effectivity.

**benchmark**
Represents performance data collected during runtime tests under various conditions that emulate expected and extreme use of the product.

**beta**
An external release, delivered as an installable application, and subject to acceptance, validation, and integration testing. Specially selected and prepared end users may participate in beta testing.

**bill of material**
A list of Items associated with a parent Item, such as an assembly, and information about how each Item relates to that parent Item.

**Bills of Material**
The application in Oracle Applications in which you define a bill of material.

**BOM**
See bill of material.

**BOM item**
The node imported into Oracle Configurator Developer that corresponds to an Oracle Bills of Material item. Can be a BOM Model, BOM Option Class node, or BOM Standard Item node.

**BOM Model**
A model that you import from Oracle Bills of Material into Oracle Configurator Developer. When you import a BOM Model, effective dates, ATO rules, and other data are also imported into Configurator Developer. In Configurator Developer, you can extend the structure of the BOM Model, but you cannot modify the BOM Model itself or any of its attributes.
BOM Model node
The imported node in Oracle Configurator Developer that corresponds to a BOM Model created in Oracle Bills of Material.

BOM Option Class node
The imported node in Oracle Configurator Developer that corresponds to a BOM Option Class created in Oracle Bills of Material.

BOM Standard Item node
The imported node in Oracle Configurator Developer that corresponds to a BOM Standard Item created in Oracle Bills of Material.

Boolean Feature
An element of a component in the Model that has two options: true or false.

bug
See defect.

build
A specific instance of an application during its construction. A build must have an install program early in the project so that application implementers can unit test their latest work in the context of the entire available application.

CIO
See Oracle Configuration Interface Object (CIO).

client
A runtime program using a server to access functionality shared with other clients.

Comparison rule
An Oracle Configurator Developer rule type that establishes a relationship to determine the selection state of a logical Item (Option, Boolean Feature, or List-of-Options Feature) based on a comparison of two numeric values (numeric Features, Totals, Resources, Option counts, or numeric constants). The numeric values being compared can be computed or they can be discrete intervals in a continuous numeric input.
Compatibility rule
An Oracle Configurator Developer rule type that establishes a relationship among Features in the Model to control the allowable combinations of Options. See also, Property-based Compatibility rule.

Compatibility Table
A kind of Explicit Compatibility rule. For example, a type of compatibility relationship where the allowable combination of Options are explicitly enumerated.

component
A piece of something or a configurable element in a model such as a BOM Model, Model, or Component.

Component
An element of the model structure, typically containing Features, that is configurable and instantiable. An Oracle Configurator Developer node type that represents a configurable element of a Model. Corresponds to one UI screen of selections in a runtime Oracle Configurator.

Component Set
An element of the Model that contains a number of instantiated Components of the same type, where each Component of the set is independently configured.

concurrent manager
A process manager that coordinates the concurrent processes generated by users’ concurrent requests. An Oracle Applications product group can have several concurrent managers.

concurrent process
A task that can be scheduled and is run by a concurrent manager. A concurrent process runs simultaneously with interactive functions and other concurrent processes.

concurrent processing facility
An Oracle Applications facility that runs time-consuming, non-interactive tasks in the background.

concurrent program
Executable code (usually written in SQL*Plus or Pro*C) that performs the function(s) of a requested task. Concurrent programs are stored procedures that
perform actions such as generating reports and copying data to and from a database.

**concurrent request**
A user-initiated request issued to the concurrent processing facility to submit a non-interactive task, such as running a report.

**configuration**
A specific set of specifications for a product, resulting from selections made in a runtime configurator.

**configuration attribute**
A characteristic of an item that is defined in the host application (outside of its inventory of items), in the Model, or captured during a configuration session. Configuration attributes are inputs from or outputs to the host application at initialization and termination of the configuration session, respectively.

**Configuration Interface Object**
See Oracle Configuration Interface Object (CIO).

**configuration model**
Represents all possible configurations of the available options, and consists of model structure and rules. It also commonly includes User Interface definitions and Functional Companions. A configuration model is usually accessed in a runtime Oracle Configurator window. See also model.

**configuration rules**
The Oracle Configurator Developer Logic rules, Compatibility rules, Comparison rules, Numeric rules, and Design Charts available for defining configurations. See also rules.

**configuration session**
The time from launching or invoking to exiting Oracle Configurator, during which end users make selections to configure an orderable product. A configuration session is limited to one configuration model that is loaded when the session is initialized.

**configurator**
The part of an application that provides custom configuration capabilities. Commonly, a window that can be launched from a hosting application so end users
can make selections resulting in valid configurations. Compare Oracle Configurator.

connectivity
The connection between client and database server that allows data communication.

The connection across components of a model that allows modeling such products as networks and material processing systems.

Connector
The node in the model structure that enables an end user at runtime to connect the Connector node's parent to a referenced Model.

Container Model
A type of BOM Model that you import from Oracle Bills of Material into Oracle Configurator Developer to create configuration models containing connectivity and trackable components. Configurations created from Container Models can be tracked and updated in Oracle Install Base.

count
The number or quantity of something, such as selected options. Compare instance.

CRM
See Customer Relationship Management
CTO
Configure to Order

customer
The person for whom products are configured by end users of the Oracle Configurator or other ERP and CRM applications. Also the end users themselves directly accessing Oracle Configurator in a Web store or kiosk.

customer-centric extensions
See customer-centric views.

customer-centric views
Optional extensions to core functionality that supplement configuration activities with rules for preselection, validation, and intelligent views. View capabilities include generative geometry, drawings, sketches and schematics, charts, performance analyses, and ROI calculations.

Customer Relationship Management
The aspect of the enterprise that involves contact with customers, from lead generation to support services.

customer requirements
The needs of the customer that serve as the basis for determining the configuration of products, systems, and services. Also called needs assessment. See guided buying or selling.

CZ
The product shortname for Oracle Configurator in Oracle Applications.

data import
Populating the Oracle Configurator schema with enterprise data from ERP or legacy systems via import tables.

Data Integration Object
Also known as the DIO, the Data Integration Object is a server in the runtime application that creates and manages the interface between the client (usually a user interface) and the Oracle Configurator schema.
data maintenance environment
The environment in which the runtime Oracle Configurator data is maintained.

data source
A programmatic reference to a database. Referred to by a data source name (DSN).

DBMS
Database Management System

default
A predefined value. In a configuration, the automatic selection of an option based on the preselection rules or the selection of another option.

Defaults rule
An Oracle Configurator Developer Logic rule that determines the logic state of Features or Options in a default relation to other Features and Options. For example, if A Defaults B, and you select A, B becomes Logic True (selected) if it is available (not Logic False).

defect
A failure in a product to satisfy the users' requirements. Defects are prioritized as critical, major, or minor, and fixes range from corrections or workarounds to enhancements. Also known as a bug.

defect tracking
A system of identifying defects for managing additional tests, testing, and approval for release to users.

deliverable
A work product that is specified for review and delivery.

demonstration
A presentation of the tested application, showing a particular usage scenario.

Design Chart
An Oracle Configurator Developer rule type for defining advanced Explicit Compatibilities interactively in a table view.
design review
A technical review that focuses on application or system design.

developer
The person who uses Oracle Configurator Developer to create a configurator. See also implementer and user.

Developer
The tool (Oracle Configurator Developer) used to create configuration models.

DHTML
Dynamic Hypertext Markup Language

DIO
See Data Integration Object.

distributed computing
Running various components of a system on separate machines in one network, such as the database on a database server machine and the application software on a Web server machine.

DLL
Dynamically Linked Library

DSN
See data source.

element
Any entity within a model, such as Options, Totals, Resources, UI controls, and components.

end user
The ultimate user of the runtime Oracle Configurator. The types of end users vary by project but may include salespeople or distributors, administrative office staff, marketing personnel, order entry personnel, product engineers, or customers directly accessing the application via a Web browser or kiosk. Compare user.

enterprise
The systems and resources of a business.
environment
The arena in which software tools are used, such as operating system, applications, and server processes.

ERP
Enterprise Resource Planning. A software system and process that provides automation for the customer’s back-room operations, including order processing.

Excludes rule
An Oracle Configurator Developer Logic rule determines the logic state of Features or Options in an excluding relation to other Features and Options. For example, if A Excludes B, and if you select A, B becomes Logic False, since it is not allowed when A is true (either User or Logic True). If you deselect A (set to User False), there is no effect on B, meaning it could be User or Logic True, User or Logic False, or Unknown. See Negates rule.

extended functionality
A release after delivery of core functionality that extends that core functionality with customer-centric views, more complex proposal generation, discounting, quoting, and expanded integration with ERP, CRM, and third-party software.

feature
A characteristic of something, or a configurable element of a component at runtime.

Feature
An element of the model structure. Features can either have a value (numeric or Boolean) or enumerated Options.

Functional Companion
An extension to the configuration model beyond what can be implemented in Configurator Developer.

An object associated with a Component that supplies methods that can be used to initialize, validate, and generate customer-centric views and outputs for the configuration.

functional specification
Document describing the functionality of the application based on user requirements.
**guided buying or selling**
Needs assessment questions in the runtime UI to guide and facilitate the configuration process. Also, the model structure that defines these questions. Typically, guided selling questions trigger configuration rules that automatically select some product options and exclude others based on the end user's responses.

**host application**
An application within which Oracle Configurator is embedded as integrated functionality, such as Order Management or iStore.

**HTML**
Hypertext Markup Language

**ICX**
Inter-Cartridge Exchange

**implementation**
The stage in a project between defining the problem by selecting a configuration technology vendor, such as Oracle, and deploying the completed configuration application. The implementation stage includes gathering requirements, defining test cases, designing the application, constructing and testing the application, and delivering it to end users. See also developer and user.

**implementer**
The person who uses Oracle Configurator Developer to build the model structure, rules, and UI customizations that make up a runtime Oracle Configurator. Commonly also responsible for enabling the integration of Oracle Configurator in a host application.

**Implies rule**
An Oracle Configurator Developer Logic rule that determines the logic state of Features or Options in an implied relation to other Features and Options. For example, if A Implies B, and you select A, B becomes Logic True. If you deselect A (set to User False), there is no effect on B, meaning it could be User or Logic True, User or Logic False, or Unknown. See Requires rule.

**import server**
A database instance that serves as a source of data for Oracle Configurator's Populate, Refresh, and Synchronization concurrent processes. The import server is sometimes referred to as the remote server.
import tables
Tables mirroring the Oracle Configurator schema Item Master structure, but without integrity constraints. Import tables allow batch population of the Oracle Configurator schema’s Item Master. Import tables also store extractions from Oracle Applications or legacy data that create, update, or delete records in the Oracle Configurator schema Item Master.

incremental construction
The process of organizing the construction of the application into builds, where each build is designed to meet a specified portion of the overall requirements and is unit tested.

initialization message
The XML message sent from a host application to the Oracle Configurator Servlet, containing data needed to initialize the runtime Oracle Configurator. See also termination message.

install program
Software that sets up the local machine and installs the application for testing and use.

Instance
An Oracle Configurator Developer attribute of a component’s node that specifies a minimum and maximum value. See also instance.

instance
A runtime occurrence of a component in a configuration. See also instantiate. Compare count.

Also, the memory and processes of a database.

instantiate
To create an instance of something. Commonly, to create an instance of a component in the runtime user interface of a configuration model.

integration
The process of combining multiple software components and making them work together.
integration testing
Testing the interaction among software programs that have been integrated into an application or system. Compare unit test.

intelligent views
Configuration output, such as reports, graphs, schematics, and diagrams, that help to illustrate the value proposition of what is being sold.

IS
Information Services

item
A product or part of a product that is in inventory and can be delivered to customers.

Item
A Model or part of a Model that is defined in the Item Master. Also data defined in Oracle Inventory.

Item Master
Data stored to structure the Model. Data in the Item Master is either entered manually in Oracle Configurator Developer or imported from Oracle Applications or a legacy system.

Item Type
Data used to classify the Items in the Item Master. Item Catalogs imported from Oracle Inventory are Item Types in Oracle Configurator Developer.

Java
An object-oriented programming language commonly used in internet applications, where Java applications run inside Web browsers and servers. See also applet and servlet.

LAN
Local Area Network

LCE
Logical Configuration Engine. Compare Active Model.
legacy data
Data that cannot be imported without creating custom extraction programs.

load
Storing the configuration model data in the Oracle Configurator Servlet on the Web server. Also, the time it takes to initialize and display a configuration model if it is not preloaded.

The burden of transactions on a system, commonly caused by the ratio of user connections to CPUs or available memory.

log file
A file containing errors, warnings, and other information that is output by the running application.

Logic rules
Logic rules directly or indirectly set the logical state (User or Logic True, User or Logic False, or Unknown) of Features and Options in the Model.

There are four primary Logic rule relations: Implies, Requires, Excludes, and Negates. Each of these rules takes a list of Features or Options as operands. See also Implies rule, Requires rule, Excludes rule, and Negates rule.

maintainability
The characteristic of a product or process to allow straightforward maintenance, alteration, and extension. Maintainability must be built into the product or process from inception.

maintenance
The effort of keeping a system running once it has been deployed, through defect fixes, procedure changes, infrastructure adjustments, data replication schedules, and so on.

Metalink
Oracle’s technical support Web site at:
http://www.oracle.com/support/metalink/

Model
The entire hierarchical “tree” view of all the data required for configurations, including model structure, variables such as Resources and Totals, and elements in
support of intermediary rules. Includes both imported BOM Models and Models created in Configurator Developer. May consist of BOM Option Classes and BOM Standard Items.

**model**
A generic term for data representing products. A model contains elements that correspond to items. Elements may be components of other objects used to define products. A configuration model is a specific kind of model whose elements can be configured by accessing an Oracle Configurator window.

**model-driven UI**
The graphical views of the model structure and rules generated by Oracle Configurator Developer to present end users with interactive product selection based on configuration models.

**model structure**
Hierarchical "tree" view of data composed of elements (Models, Components, Features, Options, BOM Models, BOM Option Class nodes, BOM Standard Item nodes, Resources, and Totals). May include reusable components (References).

**MS**
Microsoft Corporation

**Negates rule**
A type of Oracle Configurator Developer Logic rule that determines the logic state of Features or Options in a negating relation to other Features and Options. For example, if one option in the relationship is selected, the other option must be Logic False (not selected). Similarly, if you deselect one option in the relationship, the other option must be Logic True (selected). See Excludes rule.

**node**
The icon or location in a Model tree in Oracle Configurator Developer that represents a Component, Feature, Option or variable (Total or Resource), Connector, Reference, BOM Model, BOM Option Class node, or BOM Standard Item node.

**Numeric rule**
An Oracle Configurator Developer rule type that express constraint among model elements in terms of numeric relationships. See also, Contributes to and Consumes from.
OC
See Oracle Configurator.

ODBC
Open Database Connectivity. A database access method that uses drivers to translate an application’s data queries into DBMS commands.

OCD
See Oracle Configurator Developer.

opportunity
The workspace in Oracle Sales Online in which products, systems, and services are configured, quotes and proposals are generated, and orders are submitted.

option
A logical selection made by the end user when configuring a component.

Option
An element of the Model. A choice for the value of an enumerated Feature.

Oracle Configuration Interface Object (CIO)
A server in the runtime application that creates and manages the interface between the client (usually a user interface) and the underlying representation of model structure and rules in the Active Model.

The CIO is the API that supports creating and navigating the Model, querying and modifying selection states, and saving and restoring configurations.

Oracle Configurator
The product consisting of development tools and runtime applications such as the Oracle Configurator schema, Oracle Configurator Developer, and runtime Oracle Configurator. Also the runtime Oracle Configurator variously packaged for use in networked or Web deployments.

Oracle Configurator architecture
The three-tier runtime architecture consists of the User Interface, the Active Model, and the Oracle Configurator schema. The application development architecture consists of Oracle Configurator Developer and the Oracle Configurator schema, with test instances of a runtime Oracle Configurator.
Oracle Configurator Developer
The suite of tools in the Oracle Configurator product for constructing and maintaining configurators.

Oracle Configurator engine
Also LCE. Compare Active Model.

Oracle Configurator schema
The implementation version of the standard runtime Oracle Configurator data-warehousing schema that manages data for the configuration model. The implementation schema includes all the data required for the runtime system, as well as specific tables used during the construction of the configurator.

Oracle Configurator Servlet
Vehicle for Oracle Configurator containing the UI Server.

Oracle Configurator window
The user interface that is launched by accessing a configuration model and used by end users to make the selections of a configuration.

Oracle SellingPoint Application
No longer available or supported.

output
The output generated by a configurator, such as quotes, proposals, and customer-centric views.

performance
The operation of a product, measured in throughput and other data.

Populator
An entity in Oracle Configurator Developer that creates Component, Feature, and Option nodes from information in the Item Master.

preselection
The default state in a configurator that defines an initial selection of Components, Features, and Options for configuration.

A process that is implemented to select the initial element(s) of the configuration.
**product**
Whatever is ordered and delivered to customers, such as the output of having configured something based on a model. Products include intangible entities such as services or contracts.

**project manager**
A member of the project team who is responsible for directing the project during implementation.

**project plan**
A document that outlines the logistics of successfully implementing the project, including the schedule.

**Property**
A named value associated with a node in the Model or the Item Master. A set of Properties may be associated with an Item Type. After importing a BOM Model, Oracle Inventory Catalog Descriptive Elements are Properties in Oracle Configurator Developer.

**Property-based Compatibility rule**
A kind of compatibility relationship where the allowable combinations of Options are specified implicitly by relationships among Property values of the Options.

**prototype**
A construction technique in which a preliminary version of the application, or part of the application, is built to facilitate user feedback, prove feasibility, or examine other implementation issues.

**PTO**
Pick to Order

**publication**
A unique deployment of a configuration model (and optionally a user interface) that enables a developer to control its availability from hosting applications such as Oracle Order Management or /Store. Multiple publications can exist for the same configuration model, but each publication corresponds to only one Model and User Interface.
publishing
The process of creating a publication record in Oracle Configurator Developer, which includes specifying applicability parameters to control runtime availability and running an Oracle Applications concurrent process to copy data to a specific database.

QA
Quality Assurance

RAD
Rapid Application Development

RDBMS
Relational Database Management System

reference
The ability to reuse an existing Model or Component within the structure of another Model (for example, as a subassembly).

Reference
An Oracle Configurator Developer node type that denotes a reference to another Model.

regression test
An automated test that ensures the newest build still meets previously tested requirements and functionality. See also incremental construction.

Requires rule
An Oracle Configurator Developer Logic rule that determines the logic state of Features or Options in a requirement relation to other Features and Options. For example, if A Requires B, and if you select A, B is set to Logic True (selected). Similarly, if you deselect A, B is set to Logic False (deselected). See Implies rule.

resource
Staff or equipment available or needed within an enterprise.

Resource
A variable in the Model used to keep track of a quantity or supply, such as the amount of memory in a computer. The value of a Resource can be positive or zero,
and can have an Initial Value setting. An error message appears at runtime when the value of a Resource becomes negative, which indicates it has been over-consumed. Use Numeric rules to contribute to and consume from a Resource.

Also a specific node type in Oracle Configurator Developer. See also node.

**reusable component**

See reference and model structure.

**reusability**

The extent to and ease with which parts of a system can be put to use in other systems.

**RFQ**

Request for Quote

**ROI**

Return on Investment

**rules**

Also called business rules or configuration rules. Constraints applied among elements of the product to ensure that defined relationships are preserved during configuration. Elements of the product are Components, Features, and Options. Rules express logic, numeric parameters, implicit compatibility, or explicit compatibility. Rules provide preselection and validation capability in Oracle Configurator.

See also Comparison rule, Compatibility rule, Design Chart, Logic rules and Numeric rule.

**runtime**

The environment and context in which applications are run, tested, or used, rather than developed.

The environment in which an implementer (tester), end user, or customer configures a product whose model was developed in Oracle Configurator Developer. See also configuration session.

**sales configuration**

A part of the sales process to which configuration technology has been applied in order to increase sales effectiveness and decrease order errors. Commonly identifies customer requirements and product configuration.
schema
The tables and objects of a data model that serve a particular product or business process. See Oracle Configurator schema.

SCM
Supply Chain Management

server
Centrally located software processes or hardware, shared by clients.

servlet
A Java application running inside a Web server. See also Java, applet, and Oracle Configurator Servlet.

SFA
Sales Force Automation

solution
The deployed system as a response to a problem or problems.

SQA
Software Quality Assurance

SQL
Structured Query Language

system
The hardware and software components and infrastructure integrated to satisfy functional and performance requirements.

termination message
The XML message sent from the Oracle Configurator Servlet to a host application after a configuration session, containing configuration outputs. See also initialization message.

test case
A description of inputs, execution instructions, and expected results that are created to determine whether a specific software feature works correctly or a specific requirement has been met.
**Total**
A variable in the **Model** used to accumulate a numeric total, such as total price or total weight.

Also a specific node type in **Oracle Configurator Developer**. See also **node**.

**UI**
See **User Interface**.

**Unknown**
The logic state that is neither true nor false, but unknown at the time a configuration session begins or when a Logic rule is executed. This logic state is also referred to as **Available**, especially when considered from the point of view of the runtime Oracle Configurator end user.

**unit test**
Execution of individual routines and modules by the application implementer or by an independent test consultant to find and resolve defects in the application. Compare integration testing.

**update**
Moving to a new version of something, independent of software release. For instance, moving a production configurator to a new version of a configuration model, or changing a configuration independent of a model update.

**upgrade**
Moving to a new release of **Oracle Configurator** or **Oracle Configurator Developer**.

**user**
The person using a product or system. Used to describe the person using **Oracle Configurator Developer** tools and methods to build a runtime Oracle Configurator. Compare end user.

**User Interface**
The part of **Oracle Configurator architecture runtime** architecture that is generated from the model structure and provides the graphical views necessary to create configurations interactively. Interacts with the Active Model and data to give end users access to customer requirements gathering, product selection, and customer-centric views.
**user interface**
The visible part of the application, including menus, dialog boxes, and other on-screen elements. The part of a system where the user interacts with the software. Not necessarily generated in Oracle Configurator Developer.

**user requirements**
A description of what the configurator is expected to do from the end user's perspective.

**user's guide**
Documentation on using the application or configurator to solve the intended problem.

**validation**
Tests that ensure that configured components will meet specific criteria set by an enterprise, such as that the components can be ordered or manufactured.

**Validation**
A type of Functional Companion that is implemented to ensure that the configured components will meet specific criteria.

**VAR**
Value-Added Reseller

**variable**
Parts of the Model that are represented by Totals, Resources, or numeric Features.

**VB**
Microsoft Visual Basic. Programming language in which portions of Oracle Configurator Developer are written.

**verification**
Tests that check whether the result agrees with the specification.

**WAN**
Wide Area Network

**Web**
The portion of the Internet that is the World Wide Web.
WIP
Work In Progress

XML
Extensible Markup Language, a highly flexible markup language for transferring data between Web applications. Used for the initialization message and termination message of the Oracle Configurator Servlet.
Index

A

Active Model
communication with user interface, 2-7
complexity, 3-7
definition, 1-2
See also configuration models
Alias
directive in httpd.conf file, 4-4
Apache Web listener
load balance, 4-1, 4-2
timeout, 4-5
Apache Web server
See Web server
contribution to performance, 4-2
default value, 4-2
recommended value, 4-3
ApJServVMTimeout
contribution to performance, 4-3
default value, 4-3
recommended value, 4-3
architecture
effect on performance, 2-6
ATP (Available To Promise)
effect on performance, 2-12
switches in jserv.properties file, 4-10
Available To Promise
See ATP

B

benchmark
cz.uiervlet.dio_share setting, 4-8
definition, 1-3, 5-2
document
test conditions, 5-3
documenting
baselines, 2-2
steps and results, 2-3
tuning process, 2-3, 2-6
duration, 5-4
planning and executing, 5-2
scope, 5-4
Test button in OC Developer, 5-3
See also testing
browser
effect on UI performance, 3-5
recommended settings, 2-10
resource limitations and UI performance, 3-6
settings on client, 2-10
transmission by, 2-11
buttons
Done, 2-14

caching
for sharing configuration models, 4-8
results of CIO methods, 3-12
case analysis
of rule propagation, 3-9
CIO (Configuration Interface Object)
code design, 2-7
custom UI code, 2-7
in Configurator architecture, 2-7
optimizing calls by custom UI, 3-7
tuning, 2-7
use by Functional Companions, 3-10 to 3-18
client
as source of performance problem, 5-2
browser settings, 2-10
collecting data
  bDebugMode parameter in
czSource.htm, 5-4
disk de-fragmentation, 2-10
limiting access to Web server, 4-6
memory, 2-10
swap space, 2-10
throughput, 5-4
tuning, 2-10
sequence, 2-5
Compatibility rules
  engine processing, 3-10
Components
  mandatory versus optional, 3-3
ComponentSet
  performance effects, 3-13
Configuration Interface Object
See CIO
configuration models
  based on production-ready BOM, 3-1
caching, 4-8, 4-10
  based on Browser settings, 2-10
  on OC Servlet startup, 4-10
complexity, 2-6, 3-2
  suppress Logic False icon, 4-8
definition, 1-2
design
  and performance, 2-7
  optimal for Configurator engine, 3-1
tuning sequence, 2-4
Functional Companions, 2-7, 3-10
initialization, 2-12
interaction with other software, 3-10
large
  load and timeout, 4-4
  load time matched to heartbeat, 4-13
load, 2-12
  initial, 2-5
  lazy, 4-9
  preloading, 3-6
  recommendation, 3-6
tradeoffs, 4-9
performance
  key areas of influence, 3-1
  reasons for poor, 3-1
preloading, 3-2, 4-10
  and cz.uiserver.lazyload setting, 4-8
  and database tuning, 2-5
  schedule, 3-6
propagation of selection changes, 2-6
routing access, 2-9
size, 2-6
  in number of nodes, 3-2
  large, 3-3
  load time, 3-2
  measuring, 3-2
  medium, 3-3
  rare, 3-3
  small, 3-3
  typical, 3-3
  UI display time affected by, 3-2
tuning, 2-7
update UI with values, 5-4, 5-5
See also Model
configuration session
  cz_session log file, 2-9
  number per CPU, 5-3
performance
  analysis, 5-2
  effect of Model structure, 3-2
  save before, 2-12
  session.timeout parameter, 4-3
  validation when idle, 4-3
configurations
  cannot be saved, 4-12
  definition, 1-3
  saving
    implicit, 2-12
    performance, 2-5
Configurator
See Oracle Configurator
configuring
  Option Classes, 3-6
  test cases, 5-8
cookies
  printing out name, C-6
CPU
  increase number, 2-9
  processing, 2-9
  scheduling, 2-12
  utilization, 5-3
    competition for processes, 2-12
    examples of, B-6, B-9
    JVM, 2-5
    MaxClients parameter, 4-7
custom user interface
  optimizing code, 2-7
CZ schema
  See Oracle Configurator schema
cz_session log file, 2-9
cz.activemodel
  contribution to performance, 4-10
  default value, 4-11
  recommended value, 4-11
cz.activemodel.lazyloadlistprice
  contribution to performance, 4-13
  default value, 4-13
  recommended value, 4-13
czSource.htm
  compared to cz_session log, 2-9
  debug mode
    during benchmark, 5-6
    for client data, 5-4
  elapsed times
    message to server, 5-5
    model update, 5-5
    network, 2-9, 5-5
  UI controls and screens layout, 5-5
cz.uiserver.check_heartbeat_timeout
  contribution to performance, 4-12
  default value, 4-13
  interaction with cz.uiserver.poll_timeout_applet, 4-11
  interaction with cz.uiserver.poll_timeout_applet_to_dhtml, 4-12
  recommended value, 4-13
cz.uiserver.lazyload
  contribution to performance, 4-9
  default value, 4-9
  recommended value, 4-10
cz.uiserver.lfalse_is_not_available
  contribution to performance, 4-8
  default value, 4-9
  recommended value, 4-9
cz.uiserver.poll_timeout_applet
  contribution to performance, 4-11
  default value, 4-11
  recommended value, 4-11
cz.uiserver.poll_timeout_applet_to_dhtml
  contribution to performance, 4-12
  default value, 4-12
  recommended value, 4-12
cz.uiservlet.dio_share
  benchmark setting, 4-8
  contribution to performance, 4-8
  default value, 4-8
  recommended value, 4-8
cz.uiservlet.pre_load_filename
  contribution to performance, 4-10
  create text file, 4-14
  default value, 4-10
D
  data
    analysis, 2-4, 5-1, 5-3
    ATP, 2-12
    collection, 5-1
    database, 5-1
    pricing, 2-12
    processed in memory, 2-8, 2-12
    that affects performance, 5-1
    tool for performance analysis, 5-3
  database
    as source of performance problem, 5-1
    commits, 2-12
    competition for CPU, 2-9
    server, 2-8
    transactions, 2-9, 2-12
    tuning, 2-5
    See also Oracle Configurator schema
  defaults
    performance effects, 3-13
  deltas (from baseline)
    minimizing despite Unknown Options, 3-5
    designing
CIO code, 2-7
configuration models, 2-7, 3-1
custom UI, 2-7
Functional Companions, 3-10
rules, 2-11, 3-7
User Interface, 3-5
versus tuning, 2-1
DHTML
Configurator
decision to use, 3-7
heartbeat, 4-12
orphaned session, 4-12
control
processing in OC Servlet, 3-7
effect on performance, 3-7
disk
de-fragmented, 2-10
distributed computing
definition, 1-3
importance of, 2-12
testing, 5-2
Documented
directive in httpd.conf file, 4-4
Done button
performance in Configurator, 2-14
Dropdown List
render time, 3-6
server processing of, 3-6

E
end users
client tuning recommendations, 2-10
demands, 2-6
idle time and timeout settings, 4-3
multiple and CPU utilization, 5-3
routing model access, 2-9
samples of, 2-3
spread across CPUs, 2-9
testing for multiple, 5-2
transmission of action, 2-11
engine
See Oracle Configurator
See also Active Model
errors
validation check, C-8
events
analyzing runtime, 5-3
Functional Companion sequence, 2-7, 3-13
timeout transactions, 2-11
UI processing, 2-11
examples
changing default selections at runtime, 3-10
collecting data to evaluate performance, B-1
LoadRunner report, 5-8
of user activity testing, 4-15
performance impact of upgrades, 2-14
typographical conventions in, xxi

F
Features
Boolean
defaulting behavior, 3-10
initial values, 3-10
firewalls
network performance considerations, 2-9
Functional Companions
definition, 1-2
design, 3-10
interaction with configuration model, 3-10
sequence of events, 2-7, 3-13
tuning, 2-7
Validation, 3-18

garbage collection, 2-9
getAvailableNodes()
performance consideration, 3-12
guided buying or selling
See also User Interface

H
heap size, 4-7
options, 4-7
hierarchy
Model structure depth, 3-3
host application
as source of performance problem, 5-2
definition, 1-2
save before configuration session, 2-12
session, 5-2
HTML
effect on performance, 3-7
HTML templates
effect on UI performance, 3-7
modified to improve performance, 3-7
HTTPD
See Apache Web listener
httpd.conf file
directives
Alias, 4-4
Documented, 4-4
Includes, 4-4
Port, 4-4
parameters
KeepAlive, 4-5
KeepAliveTimeout, 4-5
MaxClients, 4-6
MaxKeepAliveRequests, 4-5
MaxSpareServers, 4-6
MinSpareServers, 4-6
Timeout, 4-4
httpds.conf file
See httpd.conf file
I
iAS (Oracle Internet Application Server)
element in architecture, 2-8
Includes
directive in httpd.conf file, 4-4
initialization
message
for preloading, 4-13
input
case analysis of rule propagation, 3-9
internet
temporary files, 2-10
Internet Application Server
See iAS
J
Java applet
client poll of DHTML, 4-11
comparative performance, 3-6
Java Virtual Machine
See JVM
JavaScript
decision to use, 3-7
effect on performance, 3-7
processing in OC Servlet, 3-7
JServ
model load and timeout, 4-4
module in Web server, 4-1
multiple instances, 4-1
jserv.conf file
parameters
ApJServManual, 4-2
ApJServVMTimeout, 4-3
jserv.properties file
heap size, 4-7
parameters
cz.activemodel, 4-10
cz.activemodel.lazyloadlistprice, 4-13
cz.uiuserserver.check_heartbeat_timeout, 4-12
cz.uiuserserver.lazyload, 4-9
cz.uiuserserver.ifalse_is_not_available, 4-8
cz.uiuserserver.poll_timeout_applet, 4-11
cz.uiuserserver.poll_timeout_applet_to_dhtml, 4-12
cz.uiuserserver.dio_share, 4-8
cz.uiuserserver.pre_load_filename, 4-10
recommended combination of settings, 4-7
JVM (Java Virtual Machine)
limit number per CPU, 2-5
limit users on, 2-5, 4-7
startup time, 4-3
K
KeepAlive
ccontribution to performance, 4-5
recommended value, 4-5
KeepAliveTimeout
ccontribution to performance, 4-5
Index-6

default value, 4-6
recommended value, 4-6

L

limitations
browser resources and UI controls, 3-6
load balancing
effect of ApJServManual, 4-2
load time
definition, 1-2
with implicit save, 2-12
LoadRunner, 5-3
client side data, 5-6
Controller tool, C-2
Controller window, C-8
debug mode setting in czSource.htm, 5-6
log file, C-8
parameters for Oracle Configurator, C-1
version, C-1
Vuser script runtime settings, C-2
Vuser type, C-1
log files
configuration session, 2-9, C-3
cz_session, 2-9
LoadRunner, C-8
logic
cycles, 3-9
state
checking, 3-11
Logic False, 3-5
Logic True, 3-10
of Options, 3-2
reset, 3-10
Logic False
display as logic Unknown, 4-9
displayed as Available, 3-4
suppress icon display, 4-8

M

machines
client setup, 5-2
client size, 5-2
maintenance
effect on performance, 2-14
mandatory
Components, 3-3
MaxClients
contribution to performance, 4-6
default value, 4-7
JVM load, 4-7
recommended value, 4-7
maximum heap size
performance, 4-7
MaxKeepAliveRequests
contribution to performance, 4-5
default value, 4-5
recommended value, 4-5
MaxSpareServers
collection to performance, 4-6
recommended value, 4-6
memory
client machine, 2-10
increasing, 2-9
messages
violation, 3-9
Microsoft Internet Explorer
settings, 2-10
MinSpareServers
collection to performance, 4-6
recommended value, 4-6
Model
caching, 2-10
design
and performance, 2-7
Model structure
designing
References, 3-3
effectivity and performance, 3-3
hierarchical depth, 3-3
load time, 3-3
nesting, 3-3
nodes
Feature, 3-10
intermediate, 3-3, 3-8
Option, 3-6
Option Class, 3-6
Properties, 3-3
ratio of BOM to non-BOM, 3-3
Reference, 3-3
Resource, 3-3
Total, 3-3
number of Reference nodes, 3-3
Usages and performance, 3-3
See also configuration model
models
See configuration models

N
nesting
of Reference nodes, 3-3
Netscape Navigator
identification in LoadRunner, C-3
network
between Web server and database, 2-9
elapsed times
message to Web server, 5-5
firewall considerations, 2-9
latency, 2-9, 4-5
tuning, 2-6, 2-9
nodes
Properties, 3-3
Reference, 3-3
See also Model structure
numeric
cycles, 3-9
Numeric Feature
assigning constant values in, 3-3

O
OC Servlet
competition for CPU, 2-9
creating UI changes, 2-11
DHTML control processing, 3-7
JavaScript processing, 3-7
properties
cz.activemodel, 4-10
cz.activemodel.lazyloadlistprice, 4-13
cz.uiserver.check_heartbeat_timeout, 4-12
cz.uiserver.lazyload, 4-9
cz.uiserver.ifalse_is_not_available, 4-8, 4-9
cz.uiserver.poll_timeout_applet, 4-11
cz.uiserver.poll_timeout_applet_to_dhtml, 4-12
cz.uiserver.dio_share, 4-8
cz.uiserver.pre_load_filename, 4-10, 4-14
interaction with cz.uiserver.check_heartbeat_timeout, 4-12
startup model caching, 4-10
transactions, 2-8, 2-9, 2-12
tuning, 2-8
Option Classes
with many Items, 3-6
Options
assigning constant values in, 3-3
availability and performance, 3-6
hiding unavailable, 3-6
in Selection List, 3-6
Oracle Applications
debug tool, 2-13
Oracle Configurator
architecture, 2-6
competition for CPU processes, 2-12
elements of, 2-6
engine
logic processing, 2-11
rule propagation behavior, 3-8, 3-10
runtime
See runtime Oracle Configurator
TAR template, xxii
tier of most processing, 2-6
window
See runtime Oracle Configurator
Oracle Configurator Developer
definition, 1-2
designing rules, 3-2
product support, xxii
Oracle Configurator schema
purging
for performance, 2-8
Oracle Configurator Servlet
See OC Servlet
Oracle Internet Application Server
See iAS
output
case analysis of rule propagation, 3-9
parameters
  in httpd.conf, 4-4
  in jserv.conf, 4-2
  in jserv.properties, 4-7
  in zone.properties, 4-3
performance
  analyzing data, 5-3
  ATP, 2-12
  client, 2-10
  client machine, 5-2
  collecting data, 5-1
  compare measurements, 5-2
  configuration session, 5-2
  Configurator Done button, 2-14
  CPU utilization, 2-12
  data that affects, 5-1
  definition, 1-3
  determining goals, 2-3
  diagnosis, 2-3
distributed computing advantage, 2-12
effect of
  adding ComponentSets, 3-13
  defaults, 3-13
  deleting ComponentSets, 3-13
  requests, 3-13
  rule types, 3-10
  UI design, 3-5
  visibility settings, 3-6
  elements involved, 2-6
  expectations, 2-3, B-1
host application, 5-2
HTML versus DHTML, 3-7
identifying modifications, 5-1
improvements
  preloading models, 4-13
  incremental numbers, 2-4
  key areas of influence, 3-1, 3-7
  Functional Companions, 3-10
limits, 5-3
main areas of gain, 1-1, 2-2
network, 2-11
pricing, 2-12
reasons for poor, 3-1
samples of, B-1
server, 5-2
success criteria, 2-3
UI, 2-7, 2-11
usage levels, 2-3
what is good enough, 2-3
Port
  directive in httpd.conf file, 4-4
preloading
  configuration models
    and cz.uiserver.lazyload setting, 4-8
    duration, 3-2
    schedule, 3-6
pricing
  eager fetching, 4-13
  lazy loading, 4-13
  performance problems with, 2-12
  suppress display of, 4-10, 4-13
  switches in jserv.properties file, 4-10
processing
  network, 2-11
  UI, 2-11
Product Support, xxii
Properties
  assigning constant values in, 3-3
  optimizing performance with, 3-3
publications
  initialization message, 4-14
purging
  the CZ schema, 2-8
References
  nested, 3-3
  node
    removing single-use, 3-3
    planning, 3-3
    remove unused, 3-3
requests
  performance effects, 3-13
  retracting
    all, 3-14
    before adding instances, 3-14
round robin CPU scheduling, 2-12
router, 2-9
rules
  case analysis, 3-9
  commonly used subexpressions, 3-8
  Compatibility rules, 3-10
  complexity, 3-8
  contradictions, 3-9
  debugging, 3-8, 3-10
  Defaults rules, 3-10
  designing, 2-11, 3-7
  disabling, 3-8, 3-10
  folder, 3-10
  error messages, 3-9
  folders
    disabling, 3-10
    intermediate, 3-8
  logic or numeric cycles, 3-9
  number of, 3-8
  overlapping conditions, 3-10
  propagation in engine, 3-8
  propagation paths, 3-9
  redundancy, 3-9
  violation messages, 3-9
runtime Oracle Configurator
  definition, 1-2
  link flows, 4-4

S

saving
  before configuration session, 2-12
  conditions for failure, 4-12
  encrypted pages to disk, 2-10
  explicit, 2-12
  implicit, 2-12
scalability
  in benchmarks, 1-1, 5-2
scheduling
  CPU, 2-12
screen
  flips, 4-9
Secure Sockets Layer (SSL)
  httpds.conf file, 4-4
security
  httpds.conf file, 4-4
  secure socket, 4-4
Selection List
  number of Options in, 3-6
  render time, 3-6
  server processing of, 3-6
sequence
  tuning, 2-5
server
  architecture
    effect on UI performance, 3-5
  as source of performance problem, 5-2
database, 5-1
display time, 2-12
machine, 2-6
machine choice, 2-9
processing, 2-11
  Dropdown List, 3-6
  Selection List, 3-6
response time, 2-12
size and setup, 5-2
tuning, 2-9
See also Web server
session.timeout
  contribution to performance, 4-3
default value, 4-4
recommended value, 4-4
sizing
  hardware, 5-2
state
  assertions by custom code, 3-11
logic
    checking, 3-11
    false, 3-5
  of Options, 3-2
reset, 3-10
setting, 3-14
true, 3-10
setting, 3-14
Support, xxii
swap space, 2-10

T

TAR (Technical Assistance Request)
for Oracle Configurator, xxii

template, xxii

TCP
connections, 4-5
packets, 4-4

See also network

Technical Assistance Request (TAR)
See TAR

testing
collecting data, 5-1
documenting conditions and results, 5-3
failure assertion, C-8
success criteria
defining, 2-3
setting, B-1
usability, 2-3
validation, C-7, C-8

throughput
benchmarking, 2-1
definition, 1-3
tiers
See architecture

Timeout
collection to performance, 4-4
default value, 4-5
recommended value, 4-5
timeouts
and polling, 4-11
in configuration session, 4-3
in Web Server, 2-13
parameters in zone.properties file, 4-3
settings of firewall, 2-9

transactions
CIO, 2-7
collecting data on, 5-3
database, 2-8, 2-9, 2-12
frequency
calculating throughput, B-6, B-9
pricing data, 2-12
OC Servlet, 2-8, 2-9, 2-12
runtime events, 2-11
system, 2-11

tuning
areas of intervention, 2-2
components, 2-7
database, 2-5
document results, 2-4
hot spots, 2-13
methodology, 2-2
network, 2-6
OC Servlet, 2-8
principles, 2-2
process, 2-4
sequence, 2-4, 2-5
first time, 2-4
model design, 2-4
Web server, 2-5
success criteria, 2-3
Web server, 2-5

U

UI Server
polling and timeout, 4-12
unknown values
displayed as Available, 3-4
upgrading
from a previous 11i release, 2-14
usability, 2-3

User Interface
communication with Active Model, 2-7
create, 2-7
custom, 2-7, 3-5, 3-7
customized, 3-5
definition, 1-2
design, 3-5
designing
DHTML versus Java applet, 3-6
recommendations, 3-6
dynamic visibility, 3-6
generated default, 3-6

graphics
number, 3-7
size, 3-7
type, 3-7
guided selling with hidden items, 3-6
memory usage, 3-6
performance
design considerations, 3-5
due to browser, 3-6
processing of events, 2-11
rendering
  and network tuning, 2-6
  and system transactions, 2-11
elapsed layout time, 5-4, 5-5
guided buying or selling, 3-6
screen
  number of graphics per, 3-6
  number of UI controls per, 3-6
  throughput of, 3-6
time to paint, 2-6, 5-4, 5-5
types of, 3-6
types of, 3-5
UI controls
  Dropdown List, 3-6
  Option, 3-6
  Selection List, 3-6
time to paint, 2-6, 5-4, 5-5
visibility settings, 3-6

V
validation
  checks in test scripts, C-7, C-8
tests, 3-18
Validation Functional Companion type, 3-18
violation messages
  customization for intermediary rules, 3-9

W
Web server
  Apache, 2-8
configuration files
  parameters, 4-1
  summary of parameters, A-1
containing a runaway, 4-6
CPU allocation, 2-9
garbage collection, 2-9
GET request, 4-4
idle process, 4-6
limit clients, 4-6
machine, 2-6
memory, 2-9
multi-CPU machine
  load balance Apache, 4-2
  multiple instances of jServ, 4-1
  POST request, 4-4
  processing, 2-8, 4-6
  processing data, 2-12
  requests from multiple clients, 4-1
  router, 2-9
  single versus multiple users, 2-5
  tuning, 2-5
degraded performance, 2-5
iterative process, 4-1
multiple users, 2-5
See also server

Z
zone.properties file
  description, 4-3
  parameters
    session.timeout, 4-3