IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

ORACLE CORPORATION and ORACLE U.S.A. INC.,	)	
Plaintiffs/Counterclaim Defendants,	)	C.A. No. 06-414-SLR
v.	)	**JURY TRIAL DEMANDED**
EPICREALM LICENSING, LP,	)	**PUBLIC VERSION**
Defendant/Counterclaim Plaintiff.	)	

APPENDIX IN SUPPORT OF EPICREALM'S MOTION FOR PARTIAL SUMMARY JUDGMENT THAT THE ORACLE REFERENCES DO NOT ANTICIPATE

VOLUME II of IV

EXHIBITS 5 to 8

Richard L. Horwitz (#2246)
David E. Moore (#3983)
POTTER ANDERSON & CORROON LLP
Hercules Plaza, 6th Floor
1313 N. Market Street
Wilmington, DE 19899
Tel: (302) 984-6000
rhorwitz@potteranderson.com
dmoore@potteranderson.com

Attorneys for Defendant/Counterclaim Plaintiff epicRealm Licensing, LP

OF COUNSEL:

Harry J. Roper
George S. Bosy
Aaron A. Barlow
Patrick L. Patras
David R. Bennett
Paul D. Margolis
Benjamin J. Bradford
Emily C. Johnson
JENNER & BLOCK
330 N. Wabash Avenue
Chicago, IL 60611-7603
Tel: (312) 923-8305

Dated: July 31, 2008
Public Version Dated: August 7, 2008
876824 / 31393 / Oracle

TABLE OF CONTENTS

Exh.	Document
1.	U.S. Patent 5,894,554, Lowery et al.
2.	U.S. Patent 6,415,335, Lowery et al.
3.	Report of Plaintiffs' Expert Michael I. Shamos Concerning Invalidity, May 2, 2008
4.	Exhibit 3 from Shamos Expert Report
5.	Oracle WebServer User's Guide, Release 1.0 A34986-2, ORCL00941713-00941960, Ref. 153
6.	The Oracle WebServer: A Technical Discussion, ORCL01042968-01042974, Ref. 503
7.	Oracle WebServer Release 2.0, Feature List, 10/30/95, ORCL00937853-00937862
8.	Oracle WebServer 2.0 Technical Note March 1996, ORCL000761-000781, Ref 605
9.	Oracle WebServer User's Guide, Release 2.0 A23646-2, ORCL000782-1035, Ref. 604
10.	Parties' Proposed Element-by-Element Claim Construction Chart, June 30, 2008
11.	Excerpt from Oracle Internet Directory Application Developer's Guide, Release 2.1.1, September 2000, A86082-01, ORCL00106573-00106712
12.	U.S. Patent 5,761,673 Bookman, et al.
13.	The Rush to the Net, May 1996, ORCL01606514-01606526
14.	Oracle World, ORCL01050220-01050280
15.	InfoSpinner Status, 11/15/95, EPIC000146-000148
16.	Intentionally Left Blank
17.	WebSpinner product summary, EPIC000168-000173
18.	Distribution Agreement between Software AG of North America and InfoSpinner, 2/28/96, EPIC000006-000032
19.	Excerpt from Understanding SQL*Net Release 2.2, ORCL01806620-01806818
20.	Excerpt from Functional Specification for Runtime Load Balancing, RDBMS, 10gR2, ORCL01060148-01060193

Exh.	Document
21.	Excerpt from Understanding SQL Net Release 2.3, A42484-1, ORCL01808597-01808820
22.	USSN 09/234,048, US 6,415,335, Paper 9, Response to Office Action, EPIC000489-501
23.	Excerpt from Deposition transcript of Paul C, Clark, July 11, 2008
24.	Excerpt from Deposition transcript of Keith Lowery, April 25, 2008
25.	Excerpt from Deposition transcript of Katrina Montinola, February 12, 2008
26.	Excerpt from Deposition transcript of Matthew Sarboraria, May 15, 2008
27.	Excerpt from Deposition transcript of Michael Shamos, June 30, 2008
28.	Excerpt from Deposition transcript of Douglas Surber, February 28, 2008
29.	Oracle WebServer User's Guide Release 2.0 Production, Part No. A23646-1 "WebServer 2.0 Guide", ORCL01816630-01816882

IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

CERTIFICATE OF SERVICE

I, David E. Moore, hereby certify that on August 7, 2008, the attached document was electronically filed with the Clerk of the Court using CM/ECF which will send notification to the registered attorney(s) of record that the document has been filed and is available for viewing and downloading.

I further certify that on August 7, 2008, the attached document was Electronically Mailed to the following person(s):

Mary B. Graham
Morris, Nichols, Arsht & Tunnell LLP
1201 North Market Street
P.O. Box 1347
Wilmington, DE 19899
MGraham@MNAT.com

James G. Gilliland
Igor Shoiket
Townsend and Townsend and Crew LLP
Two Embarcadero Center
Eighth Floor
San Francisco, CA 94111-3834
OracleEpicrealm@townsend.com

Theodore T. Herhold
Robert J. Artuz
Eric M. Hutchins
Eric A. Mercer
Joseph A. Greco
Nitin Gupta
Townsend and Townsend and Crew LLP
379 Lytton Avenue
Palo Alto, CA 94301
OracleEpicrealm@townsend.com

Chad E. King
Townsend and Townsend and Crew LLP
1200 Seventeenth Street
Suite 2700
Denver, CO 80202
OracleEpicrealm@townsend.com

/s/ David E. Moore
Richard L. Horwitz
David E. Moore
Potter Anderson & Corroon LLP
Hercules Plaza – Sixth Floor
1313 North Market Street
P.O. Box 951
Wilmington, DE 19899-0951
(302) 984-6000
rhorwitz@potteranderson.com
dmoore@potteranderson.com

Exhibit 5

Oracle® WebServer User's Guide

Release 1.0
Part No. A34986–2

ORACLE®

MAC003037

Oracle WebServer User's Guide, Release 1.0

Part No. A34986–2

Copyright © 1995 Oracle Corporation

All rights reserved. Printed in the U.S.A.

Contributing Authors: Mala Anand, Matthew Bookman, Martin Gruber, Magnus Lonnroth,Katrina Montinola, Mary Owen, and Joe Pistritto

Contributors: Arthur Gleckler, Walter Hudson, Kelly Ireland, Aileen Jaitin, Max Pollak, Sandy Venning, and Marcel Weekes

This software was not developed for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It is the customer's responsibility to take all appropriate measures to ensure the safe use of such applications if the programs are used for such purposes.

This software/ documentation contains proprietary information of Oracle Corporation; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited.

If this software/ documentation is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

Restricted Rights Legend Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of DFARS 252.227–7013, Rights in Technical Data and Computer Software (October 1988).

Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

If this software/ documentation is delivered to a U.S. Government Agency not within the Department of Defense, then it is delivered with " Restricted Rights", as defined in FAR 52.227–14, Rights in Data – General, including Alternate III (June 1987).

The information in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error free.

Oracle, SQL*Forms, SQL*DBA, SQL*Loader, SQL*Net and SQL*Plus are registered trademarks of Oracle Corporation.

PL/ SQL and Oracle7 are trademarks of Oracle Corporation.

All other products or company names are used for identification purposes only, and may be trademarks of their respective owners.

Preface

Purpose

The Oracle WebServer User's Guide describes the Oracle WebServer and explains how to use and administer its various components.

Audience

This manual is designed for two distinct audiences: the programmer writing applications that publish Oracle data over the World Wide Web, and the administrator managing a commercial Web site that has deployed the Oracle WebServer.

How This Book is Organized

The following is a list of chapters, and the contents of each chapter:

Chapter 1 Overview

This chapter introduces the components of the Oracle WebServer, and explains how they work together.

Chapter 2 Introduction to HTML

This chapter provides basic information about HTML. The chapter also provides step by step instructions for creating an HTML file to be viewed on a Web browser.

MAC003039

ORCL00941715

Chapter 3 The Oracle Web Listener

This chapter explains how to use and administer the Oracle Web Listener. The chapter shows how the Oracle Web Listener handles requests for static and dynamic documents, and how it returns documents to the client.

Chapter 4 Introduction to the Oracle7 Server and to PL/SQL

This chapter provides a brief overview of how to use the Oracle7 Server, and how to use PL/ SQL, a language for developing applications for an Oracle database.

Chapter 5 Oracle Web Agent

This chapter describes the Oracle Web Agent, the main differentiating factor between the Oracle WebServer and other Web servers with database backends. The chapter also describes how to set up and manage the Oracle Web Agent.

Chapter 6 Oracle WebServer Developer's Toolkit

This chapter explains how to use special utilities that generate HTML tags within PL/ SQL procedures, thereby easing the development of dynamic HTML pages with PL/ SQL.

Chapter 7 The Oracle WebServer Administration Utility

This chapter explains how to use the Administration Utility, which is a set of Web pages that make it easy for you to create and configure Oracle Web Listeners and Oracle Web Agent services.

Appendix A Oracle WebServer Messages

This appendix describes the messages generated by the components ofthe Oracle WebServer. Foreach message that is caused by an error, it offers a cause and a suggested action.

> Note: If you are reading this manual on–line on the Web, you will notice a number of hypertext links to help you navigate *through the document. However, be aware that HTML does not* support a return to the origin of a link within a document. If you use a link to move to a different section of the document, you do not return to the anchor, but to the previous target that you reached through a link.

Conventions Used in This Manual

The notational conventions used in this document are described in the following table:

MAC003040

ORCL00941716

Feature	Explanation
monospace text	Text to be enterd by the user. Type text exactly as shown. Text typed on a PC is not case–sensitive, unless noted otherwise; text typed on a UNIX system is case sensitive. Monospace is also used when referring to text in syntax statements or example code.
italics	Italics represent a variable. Substitute an appropriate value. Italics are also used for emphasis, to draw attention to the first use of a technical term, and for names of documents.
[ ]	Brackets enclose optional items or indicate a function key. Do not type the brackets.
|	A vertical bar represents an "or" option among several options. You must enter one, and only one, of the options separated by the vertical bar. Do not enter the vertical bars themselves.
punctuation	Punctuation other than brackets and vertical bars must be entered in commands exactly as shown.
UPPERCASE	UPPERCASE characters within the text represent command names and HTML tags and attributes.

How to Use this Manual

Every reader of this manual should read Chapter 1, "Overview". This chapter gives a brief description of the Oracle WebServer and how it works.

If you are unfamiliar with HyperText Markup Language, see Chapter 2, "Introduction to HTML".

Administrators should read Chapter 3, "Oracle Web Listener," Chapter 5, "Oracle Web Agent," and Chapter 7 "The Oracle WebServer Administration Utility" to understand how these components work together, and how to administer them.

Programmers will be particularly interested in Chapter 5 "Oracle Web Agent," as well as Chapter 6, "Oracle WebServer Developer's Toolkit," which explains how to generate dynamic HTML documents in PL/ SQL. It lists all the functions and procedures provided by the Oracle

WebServer Developer's Toolkit to simplify the task of generating dynamic documents.

If you are unfamiliar with the Oracle7 Server and PL/ SQL, read the introduction to these two subjects in Chapter 4, "Introduction to the Oracle7 Server and to PL/ SQL".

Related Publications

The following books contain more information on the Oracle7 Server and PL/ SQL programming language:

- Oracle7 Server Concepts Manual

  This manual provides general information about the Oracle7 Server and how it works.

- Oracle7 Server Administrator's Guide

  This manual tells you how to administer the Oracle7 Server.

- Oracle7 Server Application Developer's Guide

  This manual tells you how to develop applications for the Oracle7 Server.

- Oracle7 Server Utilities User's Guide

  This manual tells you how to use SQL*DBA, Export, Import, and SQL*Loader, the utilities bundled with the Oracle7 Server.

- PL/SQL User's Guide and Reference Manual

  This manual gives you information about PL/ SQL, the Oracle7 procedural language extension to SQL.

- Oracle7 Server SQL Reference

  This manual gives you reference information about Oracle SQL commands and functions.

Your Comments Are Welcome

We value and appreciate your comments as an Oracle user and reader of the manuals. As we write, revise, and evaluate our documentation, your opinions are the most important input we receive. At the back of our printed manuals is a Reader's Comment Form, which we encourage you to use to tell us what you like and dislike about this manual or other Oracle manuals. If the form is not available, please use the following address or FAX number.

MAC003042

ORCL00941718

Oracle Internet Products Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood City, CA 94065
U.S.A.
FAX: 415–506–7200

Contents

Chapter 1

Overview 1–1
What is the Oracle WebServer? 1–2
Components of the Oracle WebServer 1–2
The Oracle Web Listener 1–3
The Oracle Web Agent 1–4
The Oracle WebServer Developer's Toolkit 1–4
Oracle7 Server 1–5
Summary 1–5

Chapter 2

Introduction To HTML 2–1
What is HTML? 2–2
How Are HTML Documents Created? 2–2
Getting Started 2–3
Document Structure 2–4
Body Tags 2–4
Summary of Basic HTML Tags 2–7
List Tags 2–8
Ordered Lists 2–8
Unordered Lists 2–9
Nested Lists 2–9
Definition Lists 2–10
Hypertext Linking 2–11
What is a URL? 2–11
Graphics within HTML Documents 2–14
Linking to Sections on Pages 2–15
Reviewing Changes to Your HTML Document 2–16

MAC003044

ORCL00941720

Adding Style to Your HTML Document . . . . . 2 – 16
Special HTML Tags . . . . . 2 – 17
Tables . . . . . 2 – 18
Basic Table Tags . . . . . 2 – 18
Basic Table Attributes . . . . . 2 – 19
Example Table . . . . . 2 – 20
Forms . . . . . 2 – 20
Forms Syntax . . . . . 2 – 21
Form Tags . . . . . 2 – 22
Form Selection Menus . . . . . 2 – 24
Creating Your Own HTML Document . . . . . 2 – 26
More Information about HTML . . . . . 2 – 27

Chapter 3
Oracle Web Listener . . . . . 3 – 1
Oracle Web Listener Features . . . . . 3 – 2
High Performance Architecture . . . . . 3 – 2
File Caching in Memory . . . . . 3 – 2
Memory Mapping of Files . . . . . 3 – 2
Directory Mapping . . . . . 3 – 3
Language Extensions . . . . . 3 – 3
HTTP Protocol Negotiation Features . . . . . 3 – 4
Common Gateway Interface (CGI) . . . . . 3 – 4
Imagemap Support . . . . . 3 – 7
Domain Name Server (DNS) Resolution . . . . . 3 – 7
Client to Web Listener Security . . . . . 3 – 8
Logging . . . . . 3 – 9
Starting Up and Shutting Down the Oracle Web Listener . . 3 – 9
Oracle Web Listener Configuration Parameters . . . . . 3 – 10
Logging Configuration Parameters . . . . . 3 – 12
Directory Mapping Parameters . . . . . 3 – 13
File Cache Definitions . . . . . 3 – 13
Encoding Definitions . . . . . 3 – 14
MIME Types Definitions . . . . . 3 – 15
Language Extensions . . . . . 3 – 15
Basic and Digest Authentication Configuration . . . . . 3 – 17
Oracle Web Listener Miscellaneous Configuration Parameters . . . . . 3 – 19

Chapter 4
Introduction to the Oracle7 Server and to PL/SQL . . . . . 4 – 1
What is the Oracle7 Server? . . . . . 4 – 1
Database Tables . . . . . 4 – 2

MAC003045

ORCL00941721

Users, Connections, Privileges, and Roles . . . . . 4 – 4
Distributed Databases . . . . . 4 – 5
What is SQL? . . . . . 4 – 5
Retrieving Data . . . . . 4 – 6
Creating Tables . . . . . 4 – 7
Inserting and Manipulating the Data . . . . . 4 – 8
Querying Multiple Tables Through Joins . . . . . 4 – 10
Where to Look for More Information . . . . . 4 – 11
What is PL/ SQL? . . . . . 4 – 12
Basic Structure and Syntax . . . . . 4 – 13
The DECLARE Section . . . . . 4 – 14
The EXECUTABLE Section . . . . . 4 – 21
Assignments . . . . . 4 – 21
Flow Control . . . . . 4 – 21
The EXCEPTION Section . . . . . 4 – 25
Declaring Exceptions . . . . . 4 – 26
Handling Exceptions . . . . . 4 – 27
Storing Procedures and Functions in the Database . . . . . 4 – 28
Privileges Required . . . . . 4 – 29
Packages . . . . . 4 – 30
Database Triggers . . . . . 4 – 34
Creating Triggers . . . . . 4 – 35

Chapter 5 Oracle Web Agent . . . . . 5 – 1
Components of the Oracle Web Agent . . . . . 5 – 2
Who Uses the Oracle Web Agent . . . . . 5 – 2
How the Oracle Web Agent Works . . . . . 5 – 3
Web Agent Service . . . . . 5 – 5
How the Web Agent Uses CGI Environment Variables . . . . . 5 – 8
Passing parameters to PL/ SQL . . . . . 5 – 9
Overloading Procedures . . . . . 5 – 15
Accessing CGI Environment Variables . . . . . 5 – 16
Oracle Web Agent Error Handling . . . . . 5 – 17
Creating Error Pages . . . . . 5 – 18
Error Log File . . . . . 5 – 19
Oracle Web Agent Version Number . . . . . 5 – 19

Chapter 6 Oracle WebServer Developer's Toolkit . . . . . 6 – 1
Installing the Oracle WebServer Developer's Toolkit . . . . . 6 – 2
Optimizing Multiple–Service Installations . . . . . 6 – 3
Security Note . . . . . 6 – 4

MAC003046

ORCL00941722

Procedure and Function Reference . . . . . 6 – 4
Parameters Passed into Procedures and Functions . . . . . 6 – 5
Print Procedures . . . . . 6 – 5
htp.print . . . . . 6 – 6
htp.p . . . . . 6 – 6
htp.prn . . . . . 6 – 6
htp.prints . . . . . 6 – 6
htp.ps . . . . . 6 – 6
Structure Tags . . . . . 6 – 7
htp.htmlOpen . . . . . 6 – 7
htp.htmlClose . . . . . 6 – 7
htp.headOpen . . . . . 6 – 7
htp.headClose . . . . . 6 – 7
htp.bodyOpen . . . . . 6 – 8
htp.bodyClose . . . . . 6 – 8
Head Related Tags . . . . . 6 – 8
htp.title . . . . . 6 – 8
htp.htitle . . . . . 6 – 9
htp.base . . . . . 6 – 9
htp.isindex . . . . . 6 – 10
htp.linkRel . . . . . 6 – 10
htp.linkRev . . . . . 6 – 10
htp.meta . . . . . 6 – 10
Body Tags . . . . . 6 – 11
htp.line . . . . . 6 – 11
htp.hr . . . . . 6 – 11
htp.nl . . . . . 6 – 11
htp.br . . . . . 6 – 12
htp.header . . . . . 6 – 12
htp.anchor . . . . . 6 – 12
htp.mailto . . . . . 6 – 13
htp.img . . . . . 6 – 13
htp.para . . . . . 6 – 13
htp.paragraph . . . . . 6 – 14
htp.address . . . . . 6 – 14
htp.comment . . . . . 6 – 14
htp.preOpen . . . . . 6 – 14
htp.preClose . . . . . 6 – 15
htp.blockquoteOpen . . . . . 6 – 15
htp.blockquoteClose . . . . . 6 – 15
List Tags . . . . . 6 – 15
htp.listHeader . . . . . 6 – 16
htp.listItem . . . . . 6 – 16

MAC003047

ORCL00941723

htp.ulistOpen . . . . . 6 – 16
htp.ulistClose . . . . . 6 – 17
htp.olistOpen . . . . . 6 – 17
htp.olistClose . . . . . 6 – 17
htp.dlistOpen . . . . . 6 – 17
htp.dlistClose . . . . . 6 – 17
htp.dlistDef . . . . . 6 – 18
htp.dlistTerm . . . . . 6 – 18
htp.menulistOpen . . . . . 6 – 18
htp.menulistClose . . . . . 6 – 18
htp.dirlistOpen . . . . . 6 – 19
htp.dirlistClose . . . . . 6 – 19
Character Format Tags . . . . . 6 – 19
htp.cite . . . . . 6 – 20
htp.code . . . . . 6 – 20
htp.emphasis . . . . . 6 – 20
htp.em . . . . . 6 – 20
htp.keyboard . . . . . 6 – 20
htp.kbd . . . . . 6 – 21
htp.sample . . . . . 6 – 21
htp.strong . . . . . 6 – 21
htp.variable . . . . . 6 – 21
Physical Format Tags . . . . . 6 – 22
htp.bold . . . . . 6 – 22
htp.italic . . . . . 6 – 22
htp.teletype . . . . . 6 – 22
Form Tags . . . . . 6 – 23
htp.formOpen . . . . . 6 – 23
htp.formClose . . . . . 6 – 23
htp.formCheckbox . . . . . 6 – 24
htp.formHidden . . . . . 6 – 24
htp.formImage . . . . . 6 – 24
htp.formPassword . . . . . 6 – 25
htp.formRadio . . . . . 6 – 25
htp.formReset . . . . . 6 – 25
htp.formSubmit . . . . . 6 – 26
htp.formText . . . . . 6 – 26
htp.formSelectOpen . . . . . 6 – 26
htp.formSelectOption . . . . . 6 – 27
htp.formSelectClose . . . . . 6 – 28
htp.formTextarea . . . . . 6 – 28
htp.formTextareaOpen . . . . . 6 – 28
htp.formTextareaClose . . . . . 6 – 29

Table Tags . . . . . 6 – 29
htp.tableOpen . . . . . 6 – 29
htp.tableClose . . . . . 6 – 29
htp.tableCaption . . . . . 6 – 30
htp.tableRowOpen . . . . . 6 – 30
htp.tableRowClose . . . . . 6 – 30
htp.tableHeader . . . . . 6 – 30
htp.tableData . . . . . 6 – 31
OWA_UTIL Package . . . . . 6 – 31
owa_util.signature . . . . . 6 – 31
owa_util.signature (cname) . . . . . 6 – 32
owa_util.showsource (cname) . . . . . 6 – 32
owa_util.showpage . . . . . 6 – 32
owa_util.get_cgi_env(function) . . . . . 6 – 33
owa_util.print_cgi_env . . . . . 6 – 33
owa_util.mime_header . . . . . 6 – 34
owa_util.get_owa_service_path (Function) . . . . . 6 – 34
owa_util.tableprint . . . . . 6 – 34
Customized Extensions to HTP Packages . . . . . 6 – 37

Chapter 7
The Oracle WebServer Administration Utility . . . . . 7 – 1
Database Administration . . . . . 7 – 2
Startup Modes . . . . . 7 – 3
Shutdown Modes . . . . . 7 – 3
Web Agent Administration . . . . . 7 – 4
Creating or Modifying Web Agent Services . . . . . 7 – 4
The Oracle Web Agent Administration Service Creation Form . . . . . 7 – 5
The Oracle Web Agent Administration Service Modification Form . . . . . 7 – 9
Securing a Web Agent Service . . . . . 7 – 10
Web Listener Administration . . . . . 7 – 11
The Oracle Web Listener Creation Form . . . . . 7 – 12
The Configure Extended Parameters Form . . . . . 7 – 16
Configure Security Form . . . . . 7 – 25

Appendix A
Oracle WebServer Messages . . . . . A – 1
00001 – 00600 Generic Oracle WebServer Configuration Messages . . . . . A – 2
05000 – 05499 Oracle Web Agent Messages . . . . . A – 3
05500 – 05599 Oracle Web Agent Administration Messages . . . . A – 5
05600 – 05699 Oracle Web Database Administration Messages . A – 7

MAC003049

ORCL00941725

05700 – 05799 Oracle Web Listener Configuration Messages . . . A – 8
05900 – 05999 Oracle Web Server Configuration Messages . . . . . A – 9

Index

MAC003050

ORCL00941726

CHAPTER

1 Overview

This chapter explains the concepts used in the Oracle WebServer to enable the creation of Web applications using the full power of the Oracle7 Server. The Oracle WebServer provides an integrated product that facilitates the creation of Web sites that are robust, interesting, easy to use, and provides a way for Web–based users to interact securely with data stored in the Oracle7 Server, the world's most popular relational database.

MAC003051

ORCL00941727

What is the Oracle WebServer?

The Oracle WebServer is an HTTP server with a tightly integrated Oracle7 Server that enables the creation of dynamic HTML documents from data stored in an Oracle database. When the data changes, these HTML documents are updated automatically, with no further effort on the part of the site administrator. This approach supplements the presentation of static, or unchanging, data which is found on most Web sites today, with the dynamic real–time data present in business systems based on the Oracle7 Server

Business data is stored on the Oracle7 Server. It is formatted into Web documents within the server and then transmitted to Web clients. All data is stored only once, eliminating the need to "snapshot" data periodically for use on the Web.

The components that make up the Oracle WebServer work together to make the delivery of both static and dynamic pages possible.

Components of the Oracle WebServer

The components of the Oracle WebServer are as follows:

- Oracle Web Listener. The Oracle Web Listener receives requests from users using any Web browser. Requests for static (file–based) pages are handled internally by the Listener process, which functions as an HTTP server.
- Oracle Web Agent. The Oracle Web Agent handles requests from users for dynamic (program–created) pages. It handles the connection to an Oracle7 Server, invocation of the requested procedure, and transmission of the resulting HTML document back to the Web browser.
- Oracle WebServer Developer's Toolkit. The Oracle WebServer Developer's Toolkit is a set of procedures that help the customer create programs to generate dynamic documents that are easily presented in HTML.
- Oracle7 Server. The Oracle7 Server provides the storage for all dynamic data in relational tables, and all the program logic used to create dynamic HTML pages.

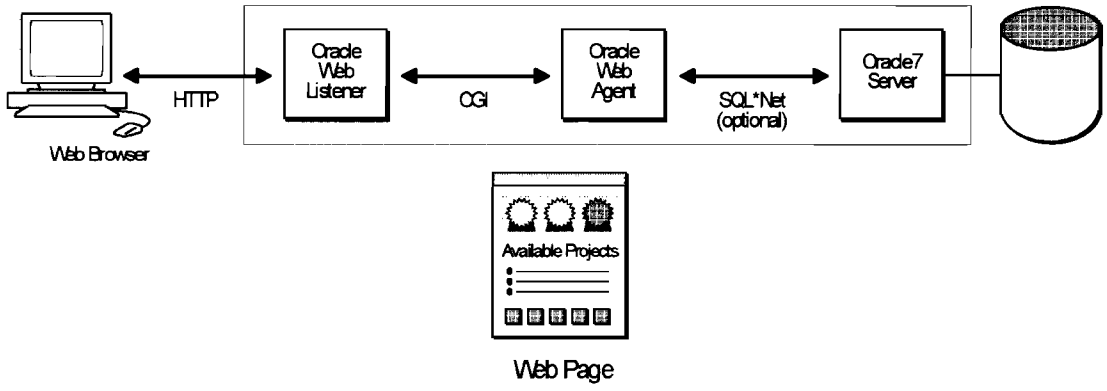
Each of these components and their interaction are described in more detail in the sections that follow.

The following picture illustrates how the Oracle WebServer integrates its components, bringing the power of Oracle7 to the World Wide Web.

MAC003052

ORCL00941728

http://www.oracle.com/invservice/owa/oracle/home



The Oracle Web Listener

The Oracle Web Listener is a commercial quality HTTP server that services document requests from any Web browser. Clients can be located anywhere on a network using the standard HTTP protocol. Under this protocol, the type of the document is sent to the client along with the document itself. The Web Listener supports many different document types are supported. Clients either interpret and display the document appropriately, or pass the document to a program that specifically handles that document type. For example, an MPEG viewer on the client machine may receive a video clip drawn from a newscast.

The Oracle Web Listener can handle a large number of simultaneous requests, and has advanced features which use system resources more efficiently than other HTTP servers available on the market. These features are described in more detail in Chapter 3, "Oracle Web Listener".

When the Oracle Web Listener receives a request from a client, it first determines whether that request is for a static document or a dynamic document. If the request is for a static document, the Web Listener sends the file and the associated type information directly to the client. If the request is for a dynamic document, it is created "on the fly" by a program invoked by the Web Listener, in compliance with the Common Gateway Interface (CGI). CGI is an interface that enables HTTP servers to run a program and use the output of that program in a document that is sent to the user. Most HTTP servers support this interface, and the Oracle Web Listener uses it to invoke the Oracle Web Agent when a

database procedure is requested. In addition, any other executable program may be invoked through CGI.

For further information about the Web Listener, see Chapter 3.

The Oracle Web Agent

The Oracle Web Agent is a program that is invoked by the Oracle Web Listener when a request for a database procedure is received. It handles the details of making a connection to the Oracle7 Server. A Web Agent will connect to a single Oracle7 Server, using a specific database username and password which are specified as part of a Web Agent service. To connect to different servers, or different schemas on the same server, multiple Web Agent services may be configured on a single Oracle WebServer. This allows for great flexibility in the creation of applications that unify data from several different servers, while controlling exactly what information a Web client can access.

For more information on the Oracle Web Agent, please see Chapter 5, "Oracle Web Agent".

The Oracle WebServer Developer's Toolkit

The Oracle WebServer Developer's Toolkit is a set of PL/ SQL routines and sample applications that help the user quickly create a Web application using the Oracle WebServer. The samples can serve as templates for user applications. With some simple modifications to include text and graphics specific to the user's site, these samples can be quickly and easily customized for a particular site.

When the Oracle Web Agent logs into the Oracle7 Server, it starts a PL/ SQL procedure that has been created by the user to generate an HTML page as its output. The procedures and functions in the Developer's Toolkit generate the actual HTML tags, so that the user can be focused on the content of the application, rather than the specifics of HTML.

For detailed information on the routines included in the Oracle WebServer Developer's Toolkit, see Chapter 6 "Oracle WebServer Developer's Toolkit".

MAC003054

ORCL00941730

Oracle7 Server

The Oracle7 Server is a high performance, fault–tolerant relational database management system, especially designed for on–line transaction processing and large database applications. The Oracle7 Server reliably manages a large amount of data in a multi–user environment. The Oracle7 Server delivers high performance even while many users concurrently access the data. It is also secure from unauthorized access and provides efficient solutions for failure recovery.

For a discussion of the Oracle7 Server, see Chapter 4 in this manual and the documentation listed in the Preface of this book.

Summary

The integrated package provided by the Oracle WebServer allows the creation of dynamic applications that take advantage of the power of Oracle7 database technology and the interactive appeal of the World Wide Web. In the following chapters, each component of the product is described in more detail.

MAC003055

ORCL00941731

MAC003056

ORCL00941732

CHAPTER

2 Introduction To HTML

This chapter discusses the basic concepts of HyperText Markup Language, or HTML. This chapter is an introduction to HTML and provides important information on how to set up, format, and define HTML documents.

The chapter covers the following topics:

- Basic concepts
- Getting started
- HTML tags
- Hypertext Linking
- Graphics
- Tables
- Forms
- Creating your own document
- On–line information about HTML

Note: This chapter contains examples of formatted elements, such as underlined text. Some of these examples may not appear correctly on all on–line viewing systems.

MAC003057

ORCL00941733

What is HTML?

Hypertext Markup Language (HTML) is the standard language used for creating hypermedia documents on the Web. HTML documents can be viewed by many different Web browsers, of varying abilities, in a simple, portable way. When a document is coded in HTML, a browser. can interpret the HTML to identify the elements of the document and to render it. The use of HTML allows documents to be formatted for presentation using fonts and line justification appropriate for the system on which it is displayed.

Most documents have common elements, such as a title, paragraphs, or lists. Using HTML tags you can label these elements as you are writing. HTML tags provide the browser with a minimum of presentation information, while keeping the integrity of information in the document. All the reader needs is a formatting tool, a Web browser, which interprets the HTML tags and produces an on–screen display that approximates the intent of the document creator.

With most methods of documentation, the writer of a document has strict control over the look and feel of a document. With HTML, the reader (subject to the capabilities of the Web browser) has control over the look and feel of a document. HTML allows you to mark titles or paragraphs with HTML instructions or tags, and then leaves the interpretation of these tags up to the browser. For example, one browser may indent the beginning of each paragraph, and another may leave only a blank line. The user of a particular browser may also have some control over the specific fonts used.

HTML tags can be divided into two main categories:

- tags that define how the body of the document is to be displayed by the browser
- tags that define information about the document such as the title

Remember, the power of HTML is that your document can be viewed on a variety of browsers, on most platforms, and formatted to suit any reader.

How Are HTML Documents Created?

HTML documents can be created using any text editor (emacs, textedit, or vi on UNIX machines; DOS and Macintosh machines have a variety of simple text editors or HTML–specific programs). Choose the editor you are most comfortable with to write your HTML document.

MAC003058

ORCL00941734

For example, HTML editors such as SoftQuad's "HoTMetaL" allow the creation of HTML documents graphically in "what you see is what you get" (WYSIWYG) mode. In addition, many traditional word processing packages have add–ons or integrated HTML output capabilities.

To create dynamic pages that retrieve information from an Oracle7 Server, you can generate HTML with the PL/ SQL utility packages provided by the Oracle WebServer Developer's Toolkit. See Chapter 5, "Oracle Web Agent," and Chapter 6, "Oracle WebServer Developer's Toolkit," for more information.

Getting Started

All HTML tags begin with a < (left angle bracket), and end with a > (right angle bracket). There is usually a beginning tag and an ending tag.

An example is the title tag which surrounds the text that is designated as the document's title:

```
<TITLE>All the Hockey Greats</TITLE>
```

Tags are usually paired as follows:

```
<TITLE> and </TITLE>
```

The ending tag looks like the beginning tag except that a forward slash precedes the text within the bracket. In this example, the tag <TITLE> tells the Web browser to use a title format, and the </ TITLE> tells the browser that the title heading is complete.

A few tags, such as <P>, which is a paragraph delimiter, do not need an end tag, but most do.

HTML is not case sensitive; therefore, the previous tags could look like this:

```
<title>All the Hockey Greats</title>
```

The convention used in this document is to capitalize all HTML format tags.

Note: Extra spaces, tabs, or returns that you have added by hand are highly discouraged, and will be lost. HTML only interprets tabs, extra spaces, and returns enclosed by the <PRE> </ PRE> tags. See "Preformatted Tag" for more information on this HTML option.

Document Structure

When a browser receives a document, it determines how it should be interpreted. The very first tag you need in your HTML document is the <HTML> structure tag. This declares that the content of your document is written with HTML. A minimal HTML document would look like this:

```
<HTML>. . . the content of the document . . .
</HTML>
```

Head Tag

The head tag can be used right after the HTML declaration, or not at all in your document. This tag represents the prologue to the rest of the file. Avoid putting any text into the document <HEAD> tag. This tag is placed immediately before and after the <TITLE> tag, as shown in the following example:

```
<HTML>
<HEAD>
<TITLE>All the Hockey Greats</TITLE>
</HEAD>
```

Note: Technically, the start and end tags for <HTML>, <HEAD>, and <BODY> are not needed. However, they are recommended because the head and body structure tags allow a browser to determine certain properties of a document, such as the <TITLE>, without having to parse, or go through the whole document.

Title Tag

Most browsers display the contents of the <TITLE> tag in the title bar of the window containing the document, and in the bookmark file of the browser if it supports one. The title, surrounded by <TITLE> and </TITLE> tags, is placed between the Head tags, as shown above. The title of a document does not appear in the contents of the document window, however. You must separately indicate it as a heading inside the body of the document if you want it to appear there.

Body Tags

The body tags specifically identify the body components in an HTML document. The body of an HTML document can contain links, text, and formatting information inside of the <BODY> and </BODY> tags.

MAC003060

ORCL00941736

Body Tag

The body of the document should be marked off with the <BODY> and </ BODY> tags. This is the part of the document that is displayed as the page of text and graphics on your Web browser.

Heading Levels

When writing an HTML document, organize the text by heading levels to reflect its structure and organization. The first heading would be level 1, the next sub heading level 2, and so on. Most browsers recognize up to six heading levels, with six distinct styles. Heading levels above 6 are *indistinguishable from one another.*

The largest heading is a level 1 heading. The syntax of the head 1 is as follows:

```
<H1>Hockey Greats on Offense</H1>
```

Other headings can be created as follows:

```
<Hx>Text here</Hx>
```

where x is a number between 1 and 6 specifying the heading level. For example, if your next heading level is a level 3, the syntax would look like the following:

```
<H3>Hockey Defense </H3>
```

Paragraph Tag

Unlike most word processors, HTML usually ignores carriage returns. Word wrapping can occur at any point in your source file. Therefore, paragraphs must be separated with the <P> tag. If you do not separate your paragraphs with the <P> tag, your document will look like one long paragraph.

Preformatted Tag

The preformatted tag, <PRE>, allows you to present text formatted specifically to a screen. The preformatted text ends at the closing </ PRE> tag. Within the preformatted text:

- Line breaks move to the next line
- The <P> tag is moved to the next line
- Horizontal tabs move in multiples of eight
- A fixed width font is used for all characters after the <PRE> tag.

Avoid using tags that define paragraph formatting, such as headings or address, within the <PRE> tags. They will have no effect.

Let's incorporate some of the previous examples to show what the document looks like with a title, a couple of heading level tags, and a few paragraphs:

```
<HTML>
<HEAD>
<TITLE>All the Hockey Greats</TITLE>
</HEAD>
<BODY>
<H1>All the Hockey Greats Before 1970</H1>
<H2>The Original Six Teams</H2>
This section deals with all of the hockey legends before
the expansion.<P>
The game was very different for these players.<P>
There is no way New Jersey would have won the Stanley Cup.
New Jersey just would not have had the talent to do it.<P>
Chicago would still be on top.<P>
All would be well.<P>
</BODY>
</HTML>
```

The result would display something like this:

All The Hockey Greats Before 1970

The Original Six Teams

This section deals with all of the hockey legends before the expansion.

The game was very different for these players.

There is no way New Jersey would have won the Stanley Cup. New Jersey just would not have had the talent to do it.

Chicago would still be on top.

All would be well.

Note: The title, "All the Hockey Greats," would not show within the document itself. On most browsers it would be displayed in the title bar.

Forced Line Breaks

The <BR> tag forces a line to break . The best example of the use of this tag is for formatting addresses, or some other sequence of lines where you don't want the browser to add extra spacing. For example:

```
Sandy's Super Sundaes<BR>
123 Main Street<BR>
Anytown, USA<BR>
```

BlockQuote

The <BLOCKQUOTE> tag is used to contain text quoted from another source. The quote will be indented approximately 8 spaces. For example:

```
My favorite hockey saying is<P>
<BLOCKQUOTE>
Today is a great day for hockey.
</BLOCKQUOTE>
But I'm not sure if Bob Johnson really said it like
that.</P>
```

This would appear something like the following:

My favorite hockey saying is

> Today is a great day for hockey.

But I'm not sure if Bob Johnson really said it like that.

Summary of Basic HTML Tags

The following table lists the basic HTML tags and their corresponding values:

Opening	Closing	Definition
<HTML>	</HTML>	An entire HTML document
<HEAD>	</HEAD>	The prologue of the document
<TITLE>	</TITLE>	Title of the document
<BODY>	</BODY>	Content of the document
<H1>	</H2>	First level heading
<H2>	</H2>	Second level heading
<H3>	</H3>	Third level heading
<H4>	</H4>	Fourth level heading
<H5>	</H5>	Fifth level heading
<H6>	</H6>	Sixth level heading

MAC003063

ORCL00941739

Opening	Closing	Definition
<P>		Paragraph
<PRE>	</ PRE>	Preformatted text
 		Forced line break
<BLOCKQUOTE>	</ BLOCKQUOTE>	Text quoted from another source

The previous section provides all you need to know to get started with HTML. At this point you can write a simple document in HTML. However, the following sections show you how to enhance your HTML pages to present information in many different fashions.

List Tags

There are three basic lists in HTML:

ordered These lists have numbered items.

unordered These lists have bullets to mark each item.

definition These lists alternate a term with its definition

You can create nested lists with indents using the ordered or unordered tags. Simply place a second list (complete with its own start and end tags) within the first lists enclosing tags. Whether the nested list uses the same markers (numbers or bullets,) depends on the browser; some track the number of nests you use and change the markers of each successive nesting to blocks or other symbols. See the following example in the section "Nested Lists".

Ordered Lists

In an ordered list, the browser automatically inserts numbers. Therefore, if you insert or delete an item in your ordered list, the numbers will reflect the change automatically.

An ordered list begins with <OL> and ends with </ OL>. The individual list items are started with the <LI> tag. The following is an example of an ordered list:

```
<OL>
<LI>Gordie Howe
<LI>Rocket Richard
```


MAC003064

ORCL00941740

```
<LI>Howie Morenz
</OL>
```

Unordered Lists

In an unordered list the browser typically uses bullets or dashes to indicate the items in your list. (Each browser has its own way of indicating an unordered list)

An unordered list begins with <UL> and ends with </UL>. The following is an example of an unordered list:

```
<UL>
<LI>Gordie Howe
<LI>Rocket Richard
<LI>Howie Morenz
</UL>
```

Nested Lists

Here is an example of how to nest a list within another:

```
<HTML>
<HEAD>
<TITLE>All the Hockey Greats</TITLE>
</HEAD>
<BODY>
<H1>Hockey Greats before Expansion</H1>
<H2>The Original Six Teams</H2>
This section deals with all of the hockey legends before
the expansion. <P>
<UL>
<LI>Gordie Howe
<LI>Rocket Richard
<LI>Howie Morenz
<UL>
<LI>great player
<LI>good stickhandler
</UL>
<LI>Bobby Orr
</UL>
</BODY>
</HTML>
```

Here's what the example would look like:

MAC003065

ORCL00941741

Hockey Greats before Expansion

The Original Six Teams

This section deals with all of the hockey legends before the expansion.

- Gordie Howe
- Rocket Richard
- Howie Morenz
  - great player
  - good stickhandler
- Bobby Orr

Note: When you create nested lists using HTML, you do not need to indent the nested HTML components. You may wish to do so for clarity, however.

Definition Lists

The definition list tags <DL>...<D/ DL> enclose both the defined term (identified with the <DT> tag), and the definition of that term (identified with the <DD> tag). Most browsers format the definition on a separate line from the term. The following is an example of a definition list:

```
<DL>
<DT>Slapshot:
<DD>A shot used to drill the goalie at speeds up to 100
mph.
<DT>Wrist shot:
<DD>A shot used to scare the goalie after the slapshot.
</DL>
```

The output looks like this:

Slapshot:
A shot used to drill the goalie at speeds up to 100 mph.

Wristshot:
A shot used to scare the goalie after the slapshot.

MAC003066

ORCL00941742

Hypertext Linking

Hypertext linking is the key characteristic that makes the Web appealing to users. By adding hypertext links, called anchors in your HTML document, you can create a highly intuitive information flow and guide the user directly to the information he or she needs.

Anchors have a standard format that allows any Web browser to interpret a link and perform the proper function (called a method) for that type of link. Links can refer to other documents, specific locations within the same document, or can perform operations, such as retrieving a file using FTP for display by the browser. URLs can refer to a specific location by an absolute pathname, or can be relative to the current document, which is more convenient when managing a large site.

> Note: You can use hypertext links to navigate through a document or to move from document to document. However, HTML does not support returning you to the anchor point of a link within a document. If you use a hypertext link within a document, and then use the Back button, you do not return to the anchor, but to the previous point that you reached through a link.

What is a URL?

HTML uses what are called Uniform Resource Locators (URLs) to represent hypermedia links and links to network services within documents. The first part of the URL before the colon specifies the access method. The part of the URL after the colon is interpreted specifically according to the access method. In general two forward slashes after the colon indicate a machine name.

The general format of a URL is:

method://machine-name/*path*/*foo.html*

The following example would fetch the document index.html from the server www.acme.com using the HTTP protocol.

http://www.acme.com/index.html

A Uniform Resource Locator (URL) has the following format:

method://*servername*:*port*/*pathname*#*anchor*

The components of the URL are as follows:

method — is the name of the operation that is performed to interpret this URL. The most common methods are:

file	Read a file from the local disk. The filename is interpreted on the user's machine, so this can be used to display any file that resides on the user's disk. For example: `file:/home/jjones/jjones.html` displays the file `jjones.html` from the directory `/home/jjones` on the local machine.
http	Access a page over the network by way of the HTTP protocol. (This is the most common method, usually used to get an HTML document.) For example: `http://www.acme.com/` accesses Acme's home page.
mailto	Activate a mail session to the specified username and host. For example: `mailto:jjones@us.acme.com` mails a message to jjones if the browser supports mail creation. Note that the `mailto` method does not require double forward slashes after the colon.
ftp	Retrieve a file using anonymous FTP from a server. For example, `ftp://`*`hostname/directory/filename`
servername	is optional and indicates the full hostname of a machine connected to the network. For example, `www.oracle.com` is the fully qualified hostname of Oracle's web server. If a servername is not specified, the URL is a relative link, and it is assumed that the file is on the same server that was used to display the current page. An IP address may be used instead of a hostname, although it is not recommended to build content with embedded IP addresses.
port	is the TCP port number that the web server is running on. The default is 80 if `:port` is not specified. This parameter is not used in most URLs.
pathname	is the relative or absolute pathname of the document being accessed by this URL. Web servers can be configured to interpret certain pathnames differently. For instance, CGI applications work by configuring the HTTP server to recognize that files within certain directories should be executed instead of being returned to the browser. For example: `http://www.acme.com/index.html`

MAC003068

ORCL00941744

In this example, an HTTP connection is made to index.html, which is the name of the file to be accessed on the server www.acme.comusing port 80 (the default). The file could have a full UNIX–style pathname to indicate a document contained in a lower level directory. If there is no directory component in the pathname, the document must be located in the server's document root directory which is configured by the server administrator. If the pathname part of the URL is missing, many servers provide a directory listing of the document root directory or access a specific 'top level' file (usually index.html).

#anchor — The named anchor points to a specific location within an HTML page. In addition to specifying a document name, specifying #anchor will cause most browsers to move the top of the display to the point referred to by the anchor. These anchor names are inserted into documents with the NAME tag as explained under "Linking to Sections on Pages" later in this chapter.

Structure of an Anchor Link within a Document

So far, we've discussed what a URL looks like. To cause a link to be displayed for the user to access, an anchor link must be embedded in the document text. The HTML syntax which allows this is:

<A HREF="*URL*">*text_to_be_displayed_highlighted*</A>

The <A HREF="URL"> tag opens the anchor link and the </A> tag closes it. All the text between those tags is displayed highlighted in some way by a web browser. A common technique is to display it underlined and in blue or some other user–selected contrasting color.

The URL part of the tag refers to the text of the URL reference as defined in the previous section. The URL text does not appear on the user's screen; it is only used when the user activates the link, usually by clicking on it with the mouse.

An example HTML segment:

```
For interesting products see
<A HREF="http://www.acme.com">Acme's home page.</A>
```

This line would produce on the user's screen:

For interesting products see Acme's home page.

MAC003069

ORCL00941745

Graphics within HTML Documents

One of the most compelling features of the Web is the ability to embed references to graphics and other data types within a document using the <IMG...ISMAP> tags. This adds a very lively character to your pages and makes them visually interesting.

There are two ways to use graphics from within an HTML document. The first is by embedding them within the document itself, so the user's screen will display the graphics within the context of the other elements of your document (such as explanatory text). This is the most common technique used by HTML designers and is called an "inline image". The syntax for specifying this is:

```
<IMG SRC="URL" ALT="text"
    ALIGN=["top" | "middle" | "bottom" | "textop" ] ISMAP>
```

The elements in this syntax statement are as follows:

URL	is the same syntax as any other URL, as explained above. This is the way the browser accesses the actual image data file, which should be in a format supported by the browser. Currently GIF and JPEG formats are supported by most browsers. Specifying the URL is required.
ALT="text"	will cause the string text to be displayed if the browser is incapable of displaying images, or if image display is turned off. This is a way of 'labelling' the image the user would be seeing if image display was turned on. ALT is an optional keyword; if it is missing, no text will be displayed if images are turned off on the browser. Most browsers put some sort of icon on the screen to indicate an image would normally be there. Using the ALT tag is recommended so that your page is compatible with text–only browsers such as Lynx.
ALIGN	is used with one of the keywords to tell the browser where to place the next block of text. This allows a certain amount of creativity in the layout of your page. If this is not specified, most browsers put the image on the left side of the screen and fill in following text to the right of it.
ISMAP	tells the server that this image is a bitmap and allows the user to click on a location on the image to cause a URL to be accessed directly. Image maps are an advanced HTML feature and require server side configuration to function.

MAC003070

ORCL00941746

For example, the following line will cause the file logo.gif to be fetched from server www.oracle.com and the text Oracle Logo to be displayed if the user has graphic display turned off.

```
<IMG SRC="http://www.oracle.com/logo.gif" ALT="Oracle Logo">
```

Linking to Sections on Pages

You can link to a different area or section of your document by using a hidden reference marker to that specific section. This provides a quick way to move through sections of the document without having to scroll up or down. Once you click on that link, the hidden reference places you in the section, and the Browser presents the hidden marker line as the first line on the screen.

To create a link, follow these steps:

1. Create a named anchor marker in each section title that you might want to jump to. The HTML syntax is as follows:

   ```
   <A NAME="named_anchor">Text_to_link_to</A>
   ```

   In the following example, a hidden reference marker is placed in a Head2 level:

   ```
   <H2><A NAME="intro">Introduction to Hockey</A></H2>
   ```

2. Create the link by entering:

   ```
   <A HREF="#named_anchor">Text</A>
   ```

   For example, the link might appear as:

   ```
   <A HREF="#intro">Gordie Howe</A>
   ```

The "#" symbol instructs your web browser to look through the HTML document for a named anchor called "intro".

When the user clicks on "Gordie Howe," the browser displays the heading "Introduction to Hockey" at the top of the screen.

Note: As specified earlier in the URL syntax, a link to a section can appear in the same document or in another document. The example here describes a link to a specific section of the same document.

Reviewing Changes to Your HTML Document

At some point, you will want to view the changes you have made to your HTML document. The following steps show you how to view your changes:

1. After you have edited your HTML file in your favorite text editor, save the file.
2. If you are currently displaying the document in a browser, you will need to reload it to see the changes. Select File —>Reload on the browser window. (In some browsers, reload will be under another menu item.)

The browser will read in the new file information and display the file with the changes you have made.

Adding Style to Your HTML Document

HTML tags offer several text styles so that you can emphasize any text in your document. The following is a short list of the most often used tag styles :

- bold
- italics
- underline
- mono spaced (typewriter style)

You can also use combinations of styles (for example, bold and italics)

Style	Element or Tag	Result
Bold	<B>I want this text bold</ B>	I want this text bold
Italics	<I>I want this text italics</ I>	I want this text italics
Underlined	<U>I want this under-lined</ U>	I want this underlined
Mono spaced(typewriter)	<TT>I want this text typed</ TT>	I want this text typed

You can add style to text that appears anywhere in the HTML document. Combinations of styles can also be used, provided you assign all the closing tags needed.

<I >Hockey</ I > i s <B>Li f e</ B>. <P>

becomes a paragraph that looks like this:

Hockey is Life.

The style tags surround the words they affect, in conjunction with other tags such as headings.

Be careful how many style tags you use on one page. If you use too many, the text may become difficult to read.

Special HTML Tags

The following tags or characters are options provided to make your HTML document more robust.

Address Tag

The <ADDRESS> tag is used to specify the author of a particular HTML document, and a way to contact the author (such as an email address). The syntax is as follows:

```
<ADDRESS>
address_of_author
</ ADDRESS>
```

Escape Sequences

The following ASCII characters have special meaning within HTML and cannot be used in regular text:

- left angle bracket <
- right angle bracket >
- ampersand &
- double quote "

To use these characters you must use these escape sequences:

For	Use
<	<
>	>
&	&
"	"

There are several more escape sequences to support accented characters, such as the umlaut or the tilde. A full list of supported characters can be

found in any number of reference manuals on HTML syntax. See also the list of on–line HTML references at the end of this chapter.

Tables

Tables in HTML organize data by row and column. Tables can contain a wide range of content, such as headers, lists, paragraphs, or figures. They can include any element or tag in HTML.

Cells can be merged across rows or columns.

Basic Table Tags

This section describes the basic table tags and their meanings.

Table: <TABLE>. . . </ TABLE>

This is the main wrapper for all the other table tags. Other table tags can be ignored if they aren't wrapped inside of the <TABLE></ TABLE> tags. By default, tables have no borders. Borders are added if the BORDER attribute is specified. See the next section "Basic Table Attributes".

Table Row: <TR> </ TR>

The number of rows in a table is specified by how many <TR> tags are contained within it. <TR> can have both the ALIGN and VALIGN attributes, which if specified become the default alignments for all cells in this row. See the next section, "Basic Table Attributes".

Table Data: <TD> </ TD>

This specifies a standard table data cell. Table data cells must only appear within table rows. Each row need not have the same number of cells specified, because short rows will be padded with blank cells on the right. A cell can contain any of the HTML tags normally present in the body of an HTML document. The default alignment of table data is ALIGN=left and VALIGN=middle. These alignments can be overridden by any alignments specified in the containing <TR> tag.

Note: Row alignments are overridden by any attributes assigned to a cell.

By default, lines inside of table cells can be broken up to fit within the overall cell width. Use the NOWRAP attribute described in the section, "Basic Table Attributes," to prevent line breaking for that cell.

Table Header: <TH> </ TH>

MAC003074

ORCL00941750

The table header cells are identical to data cells in all respects, except that header cells are in a bold font and have a default ALIGN=center.

Caption: <CAPTION>. . . </CAPTION>

This optional tag represents the caption for a table. The <CAPTION> tags should appear inside the <TABLE></TABLE> tags but not inside table rows or cells. The default alignment for the <CAPTION> tag is ALIGN=top but can be explicitly set to ALIGN=bottom. Like table cells, any document body HTML tags can appear in a caption. Captions are always horizontally centered with respect to the table, and they may have their lines broken to fit within the width of the table.

Basic Table Attributes

This section lists the basic table attributes and their meanings.

BORDER: This attribute appears in the Table tag. If present, borders are drawn around all table cells. If absent, there are no borders. By default space is left for borders, so a table has the same width with or without the border attribute.

ALIGN: If the ALIGN attribute appears inside a <CAPTION> </CAPTION> tag, it controls whether the caption appears above or below the table. It can have the values top or bottom. The default attribute is ALIGN=top.

When appearing inside a <TR>, <TH>, or <TD> tag, ALIGN controls whether text inside the table cell(s) is aligned to the left side of the cell, the right side of the cell, or centered within the cell. Values are left, center, and right.

VALIGN: The VALIGN attribute appears inside a <TR>, <TH>, or <TD> tag. This attribute controls whether text inside the table cell(s) is aligned to the top of the cell, the bottom of the cell, or vertically centered within the cell. It can also specify that all the cells in the row should be vertically aligned to the same baseline. Values are top, middle, bottom, and baseline.

NOWRAP: If the NOWRAP attribute appears in any table cell, the lines within this cell cannot be broken to fit the width of the cell. Be cautious in use of this attribute as it can result in excessively wide cells.

COLSPAN: The COLSPAN attribute can appear in any table cell and specifies how many columns of the table a specified cell should span. The default COLSPAN for any cell is 1.

ROWSPAN: The ROWSPAN attribute can appear in any table cell and specifies how many rows of the table this cell should span. The default

ROWSPAN for any cell is 1. A span that extends into rows that were not specified with a <TR> tag will be truncated.

COLSPEC: The COLSPEC attribute can be used when needed to exert control over column widths, either by setting explicit widths or by specifying relative widths. Specify the table width explicitly or as a fraction of the current margins.

Example Table

The following is a table using some of the table tags and attributes we have previously discussed:

```
<TABLE BORDER>
<CAPTION ALIGN=bottom> Table #1 </CAPTION>
<TR><TD ROWSPAN=2> </TD> <TH COLSPAN=2>Average</TH></TR>
<TR><TH>Height </TH><TH>Weight </TH></TR>
<TR><TD>Males</TD><TD ALIGN=center> 69 </TD>
<TD ALIGN=center>150 </TD></TR>
<TR><TD>Femal es</TD><TD ALIGN=center> 64 </TD>
<TD ALIGN=center>130 </TD></TR>
</TABLE>
```

The table will look like this:

	Average	
	Height	Weight
Males	69	150
Females	64	130

Table #1

Forms

HTML pages can be formatted in any fashion, but remain read–only. The HTML form feature brings the added advantage of being interactive. An HTML form lets the Web user enter comments and specify database search criteria.

When a form is interpreted by a Web browser, a special graphical user interface (GUI) screen is created with text entry fields, buttons, checkboxes, pull–down menus, and scrolling lists. When the Web user fills out the form and presses a button indicating the form should be

MAC003076

ORCL00941752

submitted, the information on the form is sent to an HTTP server for processing by a CGI (Common Gateway Interface) program. For more information on CGI, see Chapter 3, "The Oracle Web Listener".

When you write a form, each of your input items has an <INPUT> tag. When the user places data in these items in the form, that information is encoded into the form data and is known as the "value".

All form elements have Name and Value attributes. Data are sent as NAME=VALUE pairs, separated by ampersands (&), where name is given in the NAME attribute, and value is given in the VALUE attribute, or replaced by the user.

This section illustrates the basic use of HTML forms. Forms can be used for simple table searches or complex queries to relational databases.

Forms Syntax

All forms begin with <FORM> and end with </ FORM>.

The syntax is as follows:

<FORM METHOD="get | post" ACTION="*URL*">*Form_elements_and_other_HTML*
</ FORM>

METHOD

The request method supplies the data to the program. There are two request methods that can be used to access your forms. Depending on which request method you use, you will receive the encoded results of the form in a different way.

- GET: Information from a form is appended onto the end of the URL being requested. Your CGI program receives the encoded form input in the environment variable QUERY_STRING. Use of the GET method is discouraged.
- POST: This request method transmits all form input information immediately after the requested URL. Your CGI program will receive the encoded form input on standard input. The server will not send you an EOF on the end of the data; instead use the environment variable CONTENT_LENGTH to determine how much data you should read from standard input. This is the preferred method.

ACTION

ACTION specifies the URL being requested from the form. This URL will almost always point to a CGI script to decode the form results. If you are referring to a script on the same server as the form, you can use a relative URL.

MAC003077

ORCL00941753

TEXTAREA

The <TEXTAREA> tag is used to allow a user to enter more than one line of text in a form. The following is an example of the TEXTAREA tag:

```
<TEXTAREA NAME="address" ROWS=14 COLS=60>
Chicago Blackhawks
1800 Madison Ave
Chicago, Il 60612
</TEXTAREA>
```

The attributes included within the <TEXTAREA> tag are used to initialize the field's value. The </TEXTAREA> tag is always required even if the field is initially blank.

The following are attributes of <TEXTAREA> and determine the visible dimensions of the field in characters:

- NAME–user defined name
- ROWS–height in characters of TEXTAREA
- COLS–width in characters of TEXTAREA

If you want text to appear within the text area, enter it between the start and end <TEXTAREA>tags.

INPUT

The <INPUT> tag allows you to input a single word or line of text, with a default width of 20 characters. It is usually preceded with some descriptive text.

The following are attributes of <INPUT>:

- CHECKED–When present indicates that a checkbox or radio button is selected.
- MAXLENGTH–The maximum number of characters that will be accepted as input. This limits the number of characters a user can type into the field. The form will give an error beep if the user tries to enter too many characters. This can be greater that specified by SIZE, in which case the field will scroll appropriately. The default is unlimited.
- NAME–Symbolic field name used when transferring the form's contents. This attribute is always needed and should uniquely identify this field.
- SIZE–Specifies how large an area to allocate, in characters, on the screen.

- SRC–A URL specifying an image for use only with TYPE=IMAGE.
- TYPE–Defines the type of data the field accepts. Defaults to free text. Several types of fields can be defined with the TYPE attribute:

  CHECKBOX

  Used for simple Boolean attributes, or for attributes that can take multiple values at the same time. The latter is represented by a number of checkbox fields each of which has the same NAME. Each occurrence of a checkbox is either ON or OFF. For a multi–valued attribute, there is a checkbox for each attribute, indicating whether the attribute applies. The various attributes are associated with one another by the fact that each attribute checkbox uses the same name. Each selected checkbox generates a separate NAME=VALUE pair in the submitted data, even if this results in duplicate names. The default value for CHECKBOX is ON.

  HIDDEN

  No field is presented to the user, but the content of the field is sent with the submitted form. This value may be used to transmit state information about client–server interaction, or for password information.

  IMAGE

  An image field upon which you can click with a pointing device, causing the form to be immediately submitted. The coordinates of the selected point are measured in pixel units from the upper left corner of the image, and are returned (along with the other contents of the form) in two NAME=VALUE pairs. The x–coordinate is submitted under the name of the field with .x appended, and the y–coordinate is submitted under the name of *the field with .y appended. Any value attribute is ignored. The* image itself is specified by the SRC attribute, exactly as for the <IMG> tag.

  PASSWORD

  Same as TEXT attribute, except that password text is not displayed as it is entered.

  RADIO

  For attributes which can take a single value from a set of alternatives. Each radio button field in the group should be given the same NAME. Only the selected radio button in the group generates a NAME=VALUE pair in the submitted data. Radio buttons require an explicit VALUE attribute.

RESET

This is a button that when pressed resets the form's fields to their specified initial values.

SUBMIT

This button, when pressed, submits the form. You can use the VALUE attribute to provide a non–editable label to be displayed on the button. The default label is application–specific. The NAME attribute, if used, passes a name=value pair along with the submitted form.

TEXT

Single line text entry fields. Use in conjunction with the SIZE and MAXLENGTH attributes. Use the TEXTAREA tag for text fields that can accept multiple lines.

- VALUE–Assigns an initial default value for the field, or the value when checked for checkboxes and radio buttons. This attribute is required for radio buttons.

Form Selection Menus

There are three types of selection menu tag for forms:

- Select: user selects from a fixed set of values represented by the option tag. This tag is usually displayed with a pull down menu.
- Select single: same as Select, but display is presented with a window with three items displayed at once . If there are more than three options, the window will have a scroll bar.
- Select multiple: allows multiple items to be selected from the menu.

SELECT

The SELECT tag allows the user to select a value from a fixed list. This is usually presented as a pull down menu.

The SELECT tag has one or more options between the start<SELECT> and end</ SELECT> tag. By default the first option is displayed in the menu. The following is an example of a <SELECT> tag:

```
<FORM>
<SELECT NAME=group>
<OPTION>  Gretzky
<OPTION>  Messier
```


MAC003080

ORCL00941756

```
<OPTION>  Coffey
</SELECT>
</FORM>
```

SELECT SINGLE

The SELECT SINGLE tag is the same as the SELECT tag, but options are displayed in a window with three items shown at once. If there are more than three options, the window will have a scroll bar. The SIZE tag within the SELECT tag specifies how many options will be shown in the window. The following is an example of a <SELECT SINGLE>tag:

```
<FORM>
<SELECT SINGLE NAME=group      SIZE=3>
<OPTION>                Gretzky
<OPTION>                Messier
<OPTION>                Coffey
<OPTION>                Kurri
</SELECT>
</FORM>
```

In this example, the first three names would appear in the window, and a scroll bar would scroll to the last name.

SELECT MULTIPLE

The SELECT MULTIPLE tag is the same as the SELECT SINGLE tag, but the user can select more than one option in the window. The SIZE tag specifies how many lines appear in the window, and the MULTIPLE tag specifies how many options can be selected.

The following is an example of SELECT MULTIPLE:

```
<FORM>
<SELECT MULTIPLE NAME=group SIZE=3    MULTIPLE=2>
<OPTION>                Gretzky
<OPTION>                Messier
<OPTION>                Coffey
<OPTION>                Kurri
</SELECT>
</FORM>
```

Note: On some browsers, it may be necessary to hold down the CONTROL or SHIFT key to select multiple items.

If multiple items are selected, they each get passed to the server with the same name. The decoding script has to be able to recognize multiple values associated with the same name.

Creating Your Own HTML Document

Now that most of HTML has been demystified, you can create your own HTML documents. Use any browser to view the HTML document you have created.

The following is an example of an HTML document that you can type word for word, or modify as you wish:

```
<HTML>
<HEAD>
<TITLE>Chicago Blackhawks: A Love Story</TITLE>
</HEAD>
<BODY>Chicago Blackhawks: A Love Story
<P>This is a story about my beloved Blackhawks. Year after
painful year they amass teams that could potentially win a
Stanley Cup. But year after insidiously painful year, they
manage to lose in the first or second round of the
playoffs. If you have any suggestions as to how to
reconcile this evil, many fans would be grateful. Still we
live on to love the Blackhawks.
<H2>Some Players We Have Loved</H2>
<UL>
<LI><A HREF="Hull.html">Bobby Hull</A>
<LI><A HREF="Makita.html">Stan Makita</A>
<LI><A HREF="Espo.html">Tony Esposito</A>
<LI><A HREF="Jr.html">Jeremy Roenick</A>
</UL>
<H2>Some Players We Love to Hate</H2>
<OL>
<LI><A HREF="Clark.html">Wendall Clark</A>
<LI><A HREF="Gilmour.html">Doug Gilmour</A>
<LI><A HREF="Domi.html">Tie Domi</A>
<LI><A HREF="cheapshot.html">Ulf Samuellson</A>
</OL>
<H2>List of Love Letters to Our Team</H2>
<UL>
<LI><A HREF="http://www.love2hawks.com/">Love Letters</A>
<LI><A HREF="http://www.fromNHL.com/">Letters from NHL</A>
```


MAC003082

ORCL00941758

```
</UL>
<HR><A HREF="Write">Write</A>to me.
Click below to send me comments.
<BR> <A HREF="mailto:oldstadium@madison.com">
<I>crazed4hawks, oldstadium@madison.com</I>
</A>
</BODY>
</HTML>
```

More Information about HTML

For online information on HTML, the following URL sites provide a wealth of information:

```
http://www.ncsa.uiuc.edu/demoweb/html-primer.html
http://union.ncsa.uiuc.edu/HyperNews/get/www/html.html
http://fire.clarkson.edu/doc/html/htut.html
http://ugweb.cs.ualberta.ca/~gerald/guild/html.html
```


MAC003083

ORCL00941759

MAC003084

ORCL00941760

CHAPTER

3 Oracle Web Listener

The Oracle Web Listener is a high–performance scalable and portable HTTP server, supporting high–traffic Web applications where quick response is crucial. The Oracle Web Listener is designed to enable global distributed information systems over a corporate network or the global Internet.

From a user's perspective, the Web is a collection of documents or pages which contain text, images, and hypertext links to other pages. By pointing, clicking, and traversing the links, the user has instant access to a distributed collection of information. The Oracle Web Listener merges the techniques of information retrieval and hypertext to create a powerful global information system.

The Oracle Web Listener works on a simple client–server model. Clients send requests to the Oracle Web Listener. The listener interprets a request by reading the URL, finds (or generates) the information requested, and returns that information to the client in the form requested.

You can run several Oracle Web Listeners on the same machine by giving each a dedicated port. Having multiple Web Listeners is a good way to balance the load among various applications.

The language that the Oracle Web Listener uses to communicate with its clients is HyperText Transfer Protocol (HTTP). The Web Listener implements version 1.0 of the HTTP protocol as described in the IETF HTTP Working Group (http:/ / www.ics.uci.edu/ pub/ ietf/ http/). All

MAC003085

ORCL00941761

Web clients must be able to speak the HTTP protocol in order to send and receive hypermedia documents.

HTTP is an application level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia information systems. It is a generic, stateless, object–oriented protocol which can be used for many systems, through extension of its request methods (commands).

Oracle Web Listener Features

The Oracle Web Listener includes a number of features to enhance performance and extend its usability in a variety of environments.

High Performance Architecture

For maximum performance the Web Listener is designed to run as an asynchronous engine via a single process with a single thread, in comparison to most HTTP servers, which start a new thread or process whenever a new connection is made. This considerably decreases the time required to service a request and reduces Listener machine resource utilization, providing high performance under a heavy load.

File Caching in Memory

The Oracle Web Listener allows a configurable set of commonly accessed files to be cached in memory. This provides very good performance when these files are accessed by a client.

In the current version of the Oracle Web Listener, the list of files to be cached in memory is determined by the WebServer Administrator and configured manually. The administrator should periodically analyze the Listener logs to determine which files should be cached and to change the configuration accordingly.

Memory Mapping of Files

On operating systems which support this feature, the Oracle Web Listener will automatically map files into memory addresses when they are accessed (unless it is otherwise specified that they are to be cached permanently into memory),. In this case the file will appear to be loaded in memory, allowing more than one connection accessing the same file to avoid duplicate disk reads. In addition, the operating system will

MAC003086

ORCL00941762

usually pre–fetch the next segment of a memory–mapped file while the previous one is being transmitted, further increasing performance, even in the single connection case.

Directory Mapping

The Oracle Web Listener supports mapping pathnames that appear externally in URLs on the Web to locations internally on the Web Listener machine. This allows a consistent appearance to the clients, even if the Web Listener's disk is reorganized.

The specific mapping of externally visible (virtual) directories to local physical directories in the file system is configurable by the WebServer Administrator.

For example, the URL

```
http://www.acme.com/products/info.html
```

can be mapped into the directory /disk1/products of system www.oracle.comas:

```
/disk1/products/info.html
```

If the products directory is moved to /disk3 in a reorganization, the Administrator reconfigures the virtual directory products to point to the new location /disk3/products in the Web Listener configuration file (owl.cfg), and the Web client is completely unaware of the change.

For a full description of the use of directory mappings to control the use of "virtual" parent directories and subdirectories, see the section "Directory Mappings" in Chapter 7, "The Oracle WebServer Administration Utility".

> Note: The Web Listener needs to be reloaded to recognize the change. For information on reloading the Web Listener, see "Web Listener Control Utility" later in this chapter.

Language Extensions

The Oracle Web Listener supports files stored in different character sets appropriate to the language being used. The Web Listener can interpret the file extension to provide both the type of the data in the file (such as HTML data) and the national language in which the file is written. The language mapping feature is controlled by the Administrator, and allows files written in different character sets appropriate to national languages to be accessed by clients of the Web Listener.

MAC003087

ORCL00941763

The mapping of specific file extensions to language and data types is controlled by the Web Listener configuration file.

For more information about the Web Listener configuration file, see "Configuration Parameters," later in this chapter.

HTTP Protocol Negotiation Features

One of the features of HTTP is the ability of clients and HTTP servers to exchange information about the types of data they can transfer and interpret. These negotiation features allow the construction of systems composed of different data types and languages.

When a Web client requests an object, it can specify what types of data it can receive and express a preference for specific types. The Web Listener can use the information to send the appropriately formatted data to the client if more than one format is stored on the Web Listener. This process is called HTTP type negotiation and is fully supported by the Oracle Web Listener. For example, a document might contain an inline image, which could be transferred in either GIF or JPEG format to a Web client.

The Oracle Web Listener also supports a similar concept for negotiation of the appropriate national language to display a document for a specific user. The document could be stored in multiple national languages (for instance English and Canadian French), and the appropriate copy of the document would be sent if the client expressed a preference. This makes the maintenance of a multilingual Web site a great deal easier, since there is no need to maintain duplicate URLs and separate directory trees for different languages.

Common Gateway Interface (CGI)

Common Gateway Interface (CGI) version 1.1 is the standard technique used by an HTTP server to execute a program that generates HTML output. This technique provides dynamic content rather than static content from files on disk. The Oracle Web Listener is fully compliant with CGI 1.1.

When a request comes in that the Web Listener recognizes as a request to execute a CGI application, a separate process is created to perform the operation. The Web Listener maintains communication with the process to transmit input from the Web client to it, and to retrieve the HTML output it may generate for display on the client's screen.

The Web Listener Administrator can configure specific directories to contain CGI applications to be run instead of files to be transmitted. Either of these can be done using the Web Listener configuration file. For

MAC003088

ORCL00941764

more information on the Web Listener configuration file, see "Configuration Parameters," later in this chapter.

CGI Application Execution

When a client requests a URL from the Oracle Web Listener, the URL's pathname component is analyzed to determine if it represents a directory containing files (the usual case). These files are then sent immediately to the client or to a directory containing CGI applications, sometimes referred to as scripts, which must be executed as separate programs to generate output which is then sent to the client.

The technique of CGI application execution is one of the key features of the Web. It allows the integration of many information sources. The Oracle Web Agent is one CGI application that can be accessed using this technique.

Once the Listener has determined that a URL represents a CGI application, it interprets the URL to extract path information and arguments to be passed to the CGI application on startup.

URLs to CGI programs are split into three different parts:

- virtual path
- extra path information
- query string

The syntax is as follows:

virtual_path extra_path_information?query_string

virtual path	similar to a path you would use to access a regular document or image. That is, it points the server to the file that contains the CGI program you want executed.
extra path information	additional information you can embed in the URL after the program name. Extra path information is optional. It can be used to convey constant information to your scripts independent of the client's intervention. It can also be used to access the server's virtual–to–physical path translation mechanism.
query string	another optional part of the URL. It can either be explicitly given in the hypertext anchor, it can come from a user typing into a search dialog box for an HTML document with the ISINDEX tag, or it can come from HTML forms.

MAC003089

ORCL00941765

User input from URLs and other resources

When data is typed into a search dialog or text entry form, the data is encoded using URL encoding. In this encoding the following rules apply:

- Spaces are changed into + signs
- Any of the characters can be "escaped" by changing them into a sequence of the form %xx, where x is a hexadecimal digit. The character is identified by translating the two hexadecimal digits into a number from 0–255, which represents a character.

If the data is coming from a search dialog resulting from an ISINDEX tag, the above translations can be applied directly. Further, your CGI program can receive this information fully translated on the command line if you want to avoid performing the translation yourself.

If the data is coming from an HTML form, then the location of this data varies depending on the method attribute specified with the FORM tag in your HTML document. If the GET method is used, this information comes from a QUERY_STRING variable. If the POST method is used, this information is sent to your program using standard input.

The data provided will be in the form:

name1=value1&name2=value2. . . . . &nameN=valueN

If there are any equal signs (=) or ampersands (&) in the encoded data, they are encoded using the above encoding rules. To decode the data appropriately, the NAME=VALUE pairs should be split (eliminating the ampersands), then each pair should be split into a name and a value, and then the URL decoding should be applied to each portion of the pair.

When a form is submitted you can often use the order in which the form items appear to determine the order in which your CGI program receives the NAME=VALUE pairs. However, you should not depend on this behavior. The various form elements have their own ways to determine what value will be associated with the name they are given. All of the textual input areas use the user's typed input as the value.

- Radio buttons use the value of the enabled button.
- Checkboxes are unchecked. Hence, they will use either an empty string, or their name will not appear in the encoded form data at all.
- Hidden form elements can be used to send constant or state information to your script.

MAC003090

ORCL00941766

Imagemap Support

The imagemap feature of the Oracle Web Listener allows a displayed image sent to the client to have multiple link areas that provide different hypermedia links (URLs) based on where the user clicks on the image. An example is a map of the world that allows the user to select a country by clicking on it. Use of imagemaps is extremely common in sophisticated Web sites.

The Oracle Web Listener includes native processing of imagemaps within the Web Listener. Many other HTTP servers require a separate process to be created to interpret an imagemap. By doing this internally, the Oracle Web Listener improves performance and lowers system resource utilization when this common feature of HTML is being used.

Domain Name Server (DNS) Resolution

When a client requests a URL from an HTTP server, the client's network address (IP address) is logged by the HTTP server and can be used to control access to documents on the HTTP server. The Domain Name System (DNS) converts the client's network address to the client's hostname as an ASCII string.

> Note: The DNS process can be time consuming, since another machine where the naming system is already running may be contacted.

The Oracle Web Listener provides three options for controlling DNS resolution of network addresses:

- Always resolve addresses on connections.
- Do "lazy" resolutions where the name is not resolved until an external application, a security module, or a CGI script needs it.
- Never resolve addresses even if a client asks for it.

You can use the Oracle WebServer Administration Utility or edit the configuration file to control which type of DNS resolution is used. The default is never to resolve using DNS, which provides the best performance.

When DNS resolution does occur, the results are cached. Since there are often several connections from the same client in close succession, this helps to minimize the performance penalty if DNS resolution is required.

MAC003091

ORCL00941767

Client to Web Listener Security

The security framework for the Oracle Web Listener includes support for Basic Authentication and Digest Authentication. Both of these schemes provide for username and password authentication on documents. Access may also be controlled based on the client's network address. Use the Oracle WebServer Administration Utility, described in Chapter 7, to configure the security features.

Basic Authentication

This scheme uses usernames and passwords which are sent in the clear across the network. This technique has largely been replaced in practice by Digest Authentication and is included for compatibility with older Web browsers that don't implement Digest authentication.

Digest Authentication

This scheme is similar to Basic Authentication, except that it uses usernames and passwords which are sent encrypted across the network. A cryptographic checksum (digest) is used to encode the password, preventing an attacker from gaining access to the original username/ password combination. This technique is significantly more secure than Basic Authentication and is recommended for all new applications, although older browsers, some of which are still quite popular, do not support Digest Authentication.

Internet Protocol–Based Restriction

The network address of the client may be used to restrict access to information on the Oracle Web Listener. Addresses may be individually restricted or allowed access, or entire IP networks may be specified to allow users within certain locations access while denying all others.

When a client requests a file protected by IP–based security, its address is compared against the specified addresses in the order given, and as soon as a match is found, the client is allowed or denied access as specified. Files protected by IP–based restriction may not be accessed by clients other than those listed in the configuration parameters.

Note that Digest Authentication is a safer technique to determine who a user really is, since a credential (the password) must be presented to gain access. Network address–based restriction techniques are very convenient in that they avoid password management problems, but are also less secure, since a clever attacker may falsify network addresses.

Domain Name–Based Restriction

Domain name–based restriction is similar to IP–based restriction. A list of groups and hostnames specifies which hosts should or should not have access to a set of documents.

This technique uses symbolic hostnames rather than network addresses. Hence, it is easier to administer since network addresses may change if the network architecture is changed. This technique still suffers from the

MAC003092

ORCL00941768

problem of clever attackers falsifying their network addresses and assuming the identity of a host other than their own.

Security on Specific Files

Specific files can be made secure by associating them with Basic Authentication, Digest Authentication, IP–based restriction, domain–name based restriction, or any combination of one authentication and one restriction method.

Other Security Features

In addition to the authentication features, the Web Listener's user and group ID (on UNIX implementations) may be specified in the configuration file so that the Web Listener itself can be restricted to accessing only files available to that user and group. This allows the Web Listener to be started as a privileged user as required on some implementations, but still operates as if it were an unprivileged program. This provides additional security against an attack directly on the Web Listener process. For more information on the use of the Group ID for security purposes, see "Basic Configuration Parameters" in Chapter 7 of this document.

Logging

When a request is made of the Web Listener, the request information may be logged to a specified log file. The Oracle Web Listener records connections in the industry standard Common Log Format. This format includes the first line of the client's request, which specifies the action, URL, and protocol version. It also includes such information as the client's host address, whether the request was successful, and how many bytes were transmitted. This information may be later used to analyze what types of accesses are performed by which users, and to check for errors. Use the WebServer Administration Utility to specify the log file directory and the names of the standard and error log files in the Web Listener configuration file.

> Note: Be sure that the user ID under which the Web Listener runs has privileges to read and write to the log file directory.

Starting Up and Shutting Down the Oracle Web Listener

The easiest way to start up, shut down, or reload an Oracle Web Listener is to use the Administration Utility described in Chapter 7 of this document. If you wish to perform these operations from the UNIX operating system, use the WLCTL Utility as described in the next section.

MAC003093

ORCL00941769

The WLCTL (Web Listener Control) Utility

This utility resides in the ORACLE_HOME bin directory and allows you to easily startup, stop, and reload your Oracle Web Listener.

The syntax you use to invoke WLCTL to perform various operations is as follows:

```
> wlctl [start|stop|reload] [port_number] ORACLE_HOME
```

If ORACLE_HOME is set in your environment and is the ORACLE_HOME where the Oracle WebServer is installed, it need not be included on the command line.

The following examples assume that the Oracle WebServer is located in /u01/oracle:

- With ORACLE_HOME set:

  ```
  >wlctl start 8888
  ```

- Specifying ORACLE_HOME on the command line:

  ```
  >wlctl start 8888 /u01/oracle
  ```

- With ORACLE_HOME set:

  ```
  wlctl stop 8888
  ```

- Specifying ORACLE_HOME on the command line:

  ```
  wlctl stop 8888 /u01/oracle
  ```

- With ORACLE_HOME set:

  ```
  >wlctl reload 8888
  ```

- Specifying ORACLE_HOME on the command line:

  ```
  >wlctl reload 8888 /u01/oracle
  ```

The stop and reload functions require that the Listener PID file for the Web Listener you are stopping or reloading resides in the location pointed to by the Web Listener configuration file.

Oracle Web Listener Configuration Parameters

The configuration parameters for the Oracle Web Listener are stored in owa.cfg, the Web Listener configuration file. It is read when the Web Listener is initially started and on receipt of a signal '1' on UNIX implementations. The Web Listener configuration file is specified with the –c option on the command line when the Web Listener is started,

MAC003094

ORCL00941770

allowing multiple Web Listeners with different configuration files to be started on the same Web Listener machine.

Oracle WebServer provides an HTML–based Administration Utility which may be accessed with any forms–capable Web browser. This utility eliminates the need for the WebServer administrator to edit the Web Listener configuration file manually in most cases, and includes explanatory help text on the individual parameters. The Administration Utility is described in Chapter 7, "The Oracle WebServer Administration Utility". This section documents the parameters in the configuration file for completeness, in the event an administrator wishes to edit the file manually.

The Web Listener configuration file is divided into sections, which start with a section name in brackets—for example, [NetInfo]. Individual configuration parameters are set by name = value pairs, with the configuration parameter on the left of the equal sign and the value on the right as specified in the table below.

For example, consider the following portion of a configuration file:

```
;
;  www.acme.com configuration file
;
[NetInfo]
HostName  =  www.acme.com
HostAddress  =  ANY
PortNumber  =  80
```

This portion of the file sets HostName to www.acme.com, HostAddress to ANY and PortNumber to 80.

Table 3 – 1 contains the parameters, grouped by logical function.

Section	Parameter Name	Default Value	Description
NetInfo	HostName	none	hostname Web Listener is on, using the fully qualified host name, e.g.. www.acme.com The Listener uses the primary Internet address if no hostname is specified.
	HostAddress	ANY	Internet address on which the Web Listener on a multi–homed host accepts connection. ANY indicates connections accepted on all interfaces (IP addresses). Leave this set to ANY unless you have a specific need to change it.

MAC003095

ORCL00941771

PortNumber	80	TCP/ IP port number on which the Web Listener accepts requests. Note that port numbers less than 1024 require the Web Listener to run as root on many UNIX systems. 80 is the usual default for Web HTTP servers.
MaxConnectCount	50	Maximum number of simultaneous connections from users. The Web Listener ignores requests over this limit. Note there is a compile–time per–platform limit which may not be exceeded, no matter what value is specified here.
DNSResolution	NEVER	Controls DNS address to name resolution. If set to ALWAYS, the Web Listener always translates address to names. If set to LAZY, the Web Listener resolves names on demand only. If set to LAZY_WITH_CGI, the Web Listener also resolves on demand for CGI applications. If set to NEVER the Web Listener never resolves, even on demand (this provides the best performance).
SERVERPID	none	Name of file in which to store the Web Listener's process identifier (PID).

Table 3 – 1 Oracle Web Listener Network Configuration Parameters

Logging Configuration Parameters

Table 3 – 2 contains the Oracle Web Listener logging parameters.

Section	Parameter Name	Default Value	Description
[Log]	LogDir	none	Name of directory to contain log files. This directory must exist before the Web Listener starts.
	LogInfoFile	none	Name of file in which to store routine log information.
	LogErrorFile	none	Name of file in which to log errors.

Table 3 – 2 Oracle Web Listener Logging Configuration Parameters

MAC003096

ORCL00941772

Note: Be sure that the user ID under which the Web Listener runs has privileges to read and write to the log file directory.

Directory Mapping Parameters

Directories are mapped individually in the [DirMaps] section. Each line looks like the following:

Directory_Name R|N|C *Virtual_Path_Name*

- The directory name on the left is the physical directory name on the Web Listener file system.
- The choice of R|N|C controls how the directory name is interpreted.
  - R means to map 'recursively', which means the directory is a tree of subdirectories, all of which are mapped into a tree starting at the specified virtual path name.
  - N means to map only the single directory specified, and no subdirectories.
  - C means that the directory contains CGI applications instead of files to be transmitted.
- Virtual_Path_Name is the location this directory should appear on the Web.

Consider the follow example of the directory mapping section of a Web Listener configuration file:

```
;
; directory mapping section of Web Listener configuration file
;
[DirMaps]
c:\html    R      /
```

For instance, in the above example, the directory `c:\html` and all subdirectories below it are mapped to the root directory (/) on the Web. So an access to the URL:

http://*listenername*/index.html

would actually read the following file on the Web Listener's disk.

c:\html\index.html

File Cache Definitions

This section of the configuration file, starting with the key "[FileCache]" is simply a list of all the files that should be cached in memory. This

MAC003097

ORCL00941773

causes the Web Listener to keep the file open for the life of the Web Listener.

If a file is not listed here, it is not cached. When a non–cached file is requested by a client, the server maintains the resources to access that file only as long as the client refers to the file. The server will release the resources required to access a file requested by a client once there are no outstanding client references to the file.

There are three options for specifying files in the file cache.

- An explicit filename can be given.
- An incomplete filename ending in an asterisk can be used to indicate that all files that match the portion before the asterisk will be cached.
- A directory can be specified, in which case all its files and subdirectories will be cached.

The following example specifies that the specific file `index.html`, all files starting with `a` in the directory `/marketing`, and all files in the directory `/products` and its subdirectories should be cached.

```
;
; File Cache Definitions
;
[FileCache]
/index.html
/marketing/a*
/products
```

> Note: The filenames specified are virtual pathnames (the names that appear in URLs on the Web), not necessarily the physical names of the files on disk if the Directory Mapping feature has been used.

Encoding Definitions

This section contains a list of file encodings that the Web Listener understands, and the file extensions that indicate a file is of the given type.

For example:

```
;
; encoding definitions
; List for each combination of encodings
; one or more unique extensions
;
[Encodings]
```


MAC003098

ORCL00941774

```
compress        Z
gzip            gz
```

MIME Types Definitions

This section of the configuration file defines what Multimedia Internet Mail Extensions (MIME) file types the Web Listener recognizes, and how it maps the extensions of specific files to those types. This allows the proper information about file type to be transmitted to the client with the file data, which results in proper interpretation of the file on the client. The client may spawn a helper application to interpret the file if the type is not interpreted natively by the browser in use. This process is controlled entirely on the client side.

Note that files that do not map to any type are by default considered MIME type application/octet–stream which denotes a binary file.

Each line in this section consists of:

MIME–type list of file extensions mapping to that type

For example:

```
[MIMEType]
text/html       htm html
image/jpeg      jpg jpeg
image/gif       gif
```

This example tells the Web Listener that files with extensions:

- htm and html are treated as MIME type text/html
- jpg and jpeg indicate an image/jpeg file
- gif indicates it is an image/gif file

These are all common MIME types that clients understand. There are many specialized types as well for conveying other forms of data, such as audio or video clips, and so forth.

Language Extensions

This section defines filename extensions that the Web Listener recognizes and the languages and character sets to which those extensions are mapped. Extensions are case sensitive. Note that each mapping conveys both a language and a character set mapping. All language extensions that differ only in character set are equivalent for non–text files. The ability to specify a language and a character set is

MAC003099

ORCL00941775

important for languages which have more than one character set representation (for example, Arabic).

Each line consists of:

language_ID character_set_ID list_of_file-extensions

For example:

```
[LangExt]
en iso-8859-1   eng
en unicode-1-1  engUuc
fr-CA  iso-8859-1      frc
```

In this example, all files with the extension `eng` are mapped to English, ISO 8859–1 character set. Files with extension `engU` or `uc` are mapped to English, in the unicode character set. Files with extension `frc` are mapped to French Canadian language, ISO–8859–1 character sets.

The names of the character sets are as specified in RFC 1521, the language identifiers are defined in RFC 1766.

In order to use language mapping capability within the Web Listener, URLs must be specified without trailing extensions. The Web Listener adds the appropriate language/ character set extension as required if the client supports natural language negotiation.

For instance:

`http://www.oracle.com/index`

results in access to the file:

`/index.frc.html`

if the client had configured French Canadian as the preferred language.

A list of languages may be configured on the client, so that several languages may be specified and if any of them is present, the file will be sent according to the first match. Note that if a file has no language extension before its type extension, that file is assumed to be language neutral. The default character set is ISO–8859–1.

To summarize: if the files `/index.frc.html` and `/index.html` existed, users who specified French Canadian as a language preference would see the first version of the file, and all other users would see the second version.

MAC003100

ORCL00941776

Basic and Digest Authentication Configuration

This section of the configuration file specifies the type of authentication accepted for a certain file access, and the parameters for that authentication type, including usernames, passwords, and file names.

There are several sections in the file, one for each type of security.

The concept in Basic and Digest authentication is the same. Users are listed in username/ password combinations, followed by specifying which users are in what groups. Finally, groups are assigned to security realms which are the parameters assigned to specific files to protect them.

Basic authentication is configured identically to Digest authentication, with the module name `Digest` above replaced with `Basic`.

Digest authentication is recommended wherever possible, to eliminate the possibility of passwords being intercepted on the network.

An example:

```
[Security]
        Digest {
                (Users)
                jjones: glorkz
                kmaco:  plugh
                ajames: foo

                (Groups)
                mktg:   ajames
                dev:    kmaco
                all:    jjones kmaco ajames

                (Realms)
                restricted:     all
                secret:         dev mktg
        }
```

This segment defines three users, (`jjones`, `kmaco`, `ajames`), with passwords (`glorkz`, `plugh`, `foo`) respectively. The group `all` has everyone in it, the groups `mktg` and `dev` have a single individual in each. The security realm `restricted` is accessible to all three people, but only the members of groups `dev` and `mktg` can access the `secret` realm.

In this situation, any file that was defined in the `restricted` realm would be accessible to the three users specified here (but no one else who used the same Web Listener), and any file in the `secret` realm would be accessible only to users `ajames` and `kmaco`.

MAC003101

ORCL00941777

Network Address (IP) and Hostname Restriction Configuration

IP and hostname restriction is configured more simply. Hosts or IP addresses are assembled into groups, which can have specific files assigned to their access privileges.

To include a host or a group of hosts, specify the plus sign (+) before the identifier for that host and exclude it by including a minus sign (–) before the identifier. In addition, the asterisk wildcard (*) can be used to match groups of hosts.

For example:

```
[Security]
        IP {
                st:     +144.25.16.*
                        +144.25.20.*
                dc:     +130.35.*.*
                        –130.35.1.1
        }
```

This would configure everything on the networks 144.25.16 and 144.25.20 to be in group `st`, and everything on network 130.35 except host 130.35.1.1 to be in group `dc`. If the client's address doesn't match any of these, access will be denied to any file specified to require `st` or `dc` security access.

The same technique is used with hostnames in the case of domain name restriction.

The following example configures all machines in the domain oracle.com to be in group oracle except machine www.oracle.com

```
[Security]
        Domain {
                oracle: +*.oracle.com
                        –www.oracle.com
        }
```

Once the security groups have been defined, you can assign specific protections to individual files.

Assigning Security to Individual Files

When you assign protections, you specify the filename—possibly with wildcard characters to specify a group of files, or a directory name to include that directory and everything under it—followed by a list of security schemes and a parentheses-enclosed string. The string is a realm name for Digest or Basic authentication, and a group name for IP or domain restriction. The types may be combined, by requiring either scheme to be satisfied if the vertical bar (|) is used to separate security

MAC003102

ORCL00941778

schemes, or BOTH schemes must be satisfied if the ampersand (&) is used. If a combination of security schemes is used, one scheme must be an authentication type, and the other a restriction type.

For example:

```
[Protection]
/secret/*          IP(st) | Basic(secret)
/company/*         Domain(acme)
```

This example protects everything under the directory /secret using either IP–based restriction from the st group or by anyone who can log into the secret realm using Basic authentication. All the files in /company are protected so that only people in the acme domain may see them.

Oracle Web Listener Miscellaneous Configuration Parameters

Section	Parameter Name	Default Value	Description
Server	UserDir	none	Directory under a user's home directory searched when the URL /~username/ is received. This allows users to store their own home pages in their home directories (UNIX only).
	InitialFile	initial	The file that the Listener looks for when a URL ending in a directory rather than a file is sent to the Listener. This is the 'index' file for the Web Listener.
	UserDirInitialFile	initial.html	Default filename when the /~username/ construction is used in a URL with no file specified.
	DefaultMIMEType	application/octet–stream	Default MIME type used by the Web Listener when the filename extension requested in a URL is not recognized.
	DefaultCharset	iso–8859–1	Default character set if none is inferred from the file extension.
	PreferredLanguage	en	Language a Web Listener will prefer given a choice of languages when searching for a file. Default is English (en).

MAC003103

ORCL00941779

ImageMap	none	Set this to 'map'. This is the extension the Web Listener expects imagemaps to have.
ServiceTimeout	none	On Windows NT only: Specifies the time in seconds that the NT control panel will wait for a pending start, stop, or continue command to complete before reporting a failure.

Table 3 – 3 Oracle Web Listener Miscellaneous Listener Configuration Parameters

MAC003104

ORCL00941780

CHAPTER

4

Introduction to the Oracle7 Server and to PL/ SQL

This chapter provides an overview of the Oracle7 Server. It covers the basics of relational databases, SQL, and PL/ SQL. If you already are familiar with some or all of this material, you can reasonably skip part or all of this chapter. Since there is more material on these subjects than a chapter can possibly cover, there are frequent references to other Oracle manuals that provide you with further information.

What is the Oracle7 Server?

The Oracle7 Server is a Relational Database Management System (RDBMS). That is to say, the job of the Oracle7 Server is to manage data. Users or other processes store, alter, and obtain the data by issuing statements that the RDBMS executes, but they never directly access the data. Having all the data under the control of a single entity enables that entity to ensure, for example, that the data maintains a coherent structure and that simultaneous changes by different users do not interfere with one another.

MAC003105

ORCL00941781

Database Tables

Saying that Oracle7 is a Relational Database Management System implies that all of the data it contains is structured as tables (tables are called "relations" in mathematical jargon). Even the metadata—the internally generated information describing the content of the database, such as how many tables there are and what they are called—is contained in a special set of tables called the Data Dictionary (for more information on the Data Dictionary, see Chapter 1 of the Oracle7 Server Concepts Manual).

Here is a simple table, such as you might find in an Oracle database:

CNUM	FNAME	LNAME	ADDRESS
4005	Julia	Peel	197 Myrtle Court, Brisbane, CA
4007	Terry	Subchak	2121 Oriole Way, Boston, MA
4008	Emilio	Lopez	31D San Bruno Ave. SF, CA
4011	Kerry	Lim	455 32nd St. #45, Brinton, KY

Table 4 – 1 Customers

Each row of this table describes one person, and each column has one type of information about that person. Note the column cnum. This is simply a number we generate to distinguish the customers from one another, as names are not necessarily unique. As you will see shortly when we discuss SQL, you refer to data in a relational database by its content, not by such things as where it is stored. Therefore, every table must have an identifying group of one or more columns whose values, taken as a set, are always different for every row of the table. This group, in this case the single column cnum, is called the primary key of the table. Locally generated numbers, as in this example, are a common and easy way to create primary keys.

Foreign Keys

Suppose we wanted to add our customers' phone numbers to our database. Since one person can have multiple phone numbers, these do not fit into our structure well. If we want to include the phone numbers in the Customers table, there are three possibilities, none of them good:

MAC003106

ORCL00941782

1. You could fit all the phone numbers for a given person into a single column in a single row, in which case it would be difficult to access the phone numbers independently.
2. You could create a column for each type of phone number, in which case you would have to redesign your table each time a new type arose. This also could create an unwieldy number of mostly empty columns.
3. You could enter a new row for each phone number, in which case each such row would consist of redundant information except for the new phone number. This approach would be error–prone and waste space.

The good solution is simply to create a second table, like this:

CNUM	PHONE	TYPE
4005	375–296–8226	home
4005	375–855–3778	beeper
4008	488–255–9011	home
4011	577–936–8554	home
4008	488–633–8591	work

Table 4 – 2 Customers_Phone

Notice that in this table cnum is not the primary key; it identifies the customer and therefore is the same for each phone number associated with a given customer. What, then, is the primary key? The combination of cnum and phone. If we list the same number for the same person twice, we really have made a duplicate entry and should eliminate one anyway.

The cnum column does have a special function, however, because it defines the relationship between Customers_Phone and Customers by associating each phone number with a customer. We say that it references the cnum column in Customers. A group of one or more columns, such as this, that references another group is known as a foreign key. The group of columns a foreign key references is called its parent key or its referenced key. Each foreign key value references a specific row in the table containing the parent key. Clearly, then, all sets of values in the foreign key have to be present once and only once in the parent key (although they may be present any number of times in the foreign key itself, as above) for the reference to be both meaningful and unambiguous. For that reason, the parent key must be either a primary key (the usual case) or another group of columns that is unique, which is known as a unique key.

MAC003107

ORCL00941783

Oracle can make sure that all primary and unique keys stay unique and that all foreign key references are valid; this is called maintaining referential integrity. For more information on foreign and parent keys, see Chapter 7 of the Oracle7 Server Concepts Manual and "CONSTRAINT clause" in Chapter 4 of the Oracle7 SQL Reference.

Although Oracle tables may look similar to, and often be presented as, HTML tables, there is a subtle but important difference. In HTML, tables are a way of presenting data to the user. The data itself may not actually have a tabular structure or any structure at all. For example, since not all browsers can interpret graphics, you could represent a bar graph in HTML as a table.

In the Oracle RDBMS, tables are how all data is structured and stored. Once the data from the tables is retrieved by other processes, however, those processes can reformat it so that it is not presented to the users as tables at all. For example, a very simple two–column table might consist of graphics (Oracle can store graphics, sound, video, or any binary value in a table using the datatype LONG RAW) in one column and a name for each graphic in the other. An application could use the names to request the graphics it wants from the database and present the graphics to the user, never revealing that they came from a table.

Users, Connections, Privileges, and Roles

The Oracle7 Server controls which users can do what to its data. To do this, it uses a log on procedure that is separate from that of the operating system. Once logged on the operating system, you establish a connection to the Oracle7 Server with a username and password known to Oracle that may have no relationship to the ones used for you by the operating system. You can also be Identified Externally, in which case you still connect to the database with an independent username, but Oracle7 uses the operating system to verify your identity, rather than requesting another password. For more information, see "CREATE USER" and "CONNECT" in the Oracle7 Server SQL Reference.

The name under which you connect to Oracle7—your Oracle username— is associated with a number of privileges, which are the rights to perform various actions. Privileges can be granted and revoked dynamically, and sometimes you can grant privileges you have received to other users as well. Some privileges apply to specific database objects—these are called object privileges. Others are more general in nature—these are called system privileges. For more information on privileges, see "GRANT" and "REVOKE" in Chapter 4 of the Oracle7 Server SQL Reference.

Roles are groups of privileges that are granted and revoked as single units. They simplify complex privilege assignments. They also differ from simple privileges in that they are not always in force. You can be granted a role, but have it disabled, in which case you cannot use the privileges the role contains until you enable it. For more information on roles, see "CREATE ROLE", "ALTER ROLE", and "GRANT" in the Oracle7 Server SQL Reference.

User actions can also be constrained by profiles. These control such things as how many simultaneous connections you can have and how long they can last. Profiles are beyond the scope of this manual. For more information on profiles, see "CREATE PROFILE" in the Oracle7 Server SQL Reference.

Distributed Databases

If you have a full license for the Oracle7 Server, you can use it to create distributed databases. These are multiple Oracle7 databases—located on different machines and possibly separated by considerable geographical distance—that work together to maintain a single consistent body of data that looks and behaves as though drawn from a single source. In this situation, you connect to one database, which is called the local database, and that database interacts with other databases, called remote databases, as needed. The mechanism used to refer to objects on a remote database is called a database link or dblink. For more information on distributed databases, see "Distributed Databases" in the Oracle7 Server Concepts Manual.

The software that enables distributed databases to communicate with one another is SQL*Net, an Oracle product that enables communication between various servers that possibly are on different platforms and using different network protocols. For more information on SQL*Net, see Understanding SQL*Net.

What is SQL?

SQL (Structured Query Language) is the language you use to issue instructions to the Oracle7 Server. It is, in fact, the standard language used by all major relational database vendors, and Oracle complies at Entry Level with SQL92, the most recent ISO (International Standards Organization) standard. There are several aspects of SQL that may differ from computer languages that you are familiar with, such as the following:

MAC003109

ORCL00941785

- SQL is non–procedural. In SQL, you tell the Server what to do but not how it is to be done. This frees you from dealing with a lot of detail.
- SQL statements are independent of one another. Although PL/ SQL (discussed later in this chapter) addresses this, SQL itself has no conditional or other control–flow statements.
- SQL employs set–at–a–time operation. It operates on arbitrarily large sets of data in a single step.
- SQL uses three–valued logic. In most languages, Boolean expressions are either TRUE or FALSE. In SQL, they are TRUE, FALSE, or NULL. This will be explained shortly.

Retrieving Data

Suppose you wanted to pull from the Customers table the information on customers named "Peel". This is called making a query. To do it, you could issue the following statement:

```
SELECT *
    FROM Customers
    WHERE LNAME = 'Peel';
```

This produces the following:

```
CNUM  FNAME       LNAME   ADDRESS
4005  Julia       Peel    197 Myrtle Court, Brisbane, CA
```

Oracle interprets the statement as follows: Any number of spaces and/ or line breaks are equivalent to one space or line break. These are delimiters, and the extra spaces and line breaks are for readability: all are equivalent "white space". Likewise, case is not significant, except in literals like the string you are searching for ('Peel').

SELECT is a keyword telling the database that this is a query. All SQL statements begin with keywords. The asterisk means to retrieve all columns; alternatively, you could have listed the desired columns by name, separated by commas. The FROM Customers clause identifies the table from which you want to draw the data.

WHERE LNAME = 'Peel' is a predicate. When a SQL statement contains a predicate, Oracle tests the predicate against each row of the table and performs the action (in this case, SELECT) on all rows that make the predicate TRUE. This is an example of set–at–a–time operation. The predicate is optional, but in its absence the operation is performed on the entire table, so that, in this case, the entire table would have been retrieved. The semi–colon is the statement terminator.

MAC003110

ORCL00941786

Nulls and Three–Valued Logic

With predicates, you should be aware of three–valued logic. In SQL, the basic Boolean values of TRUE and FALSE are supplemented with another: NULL, also called UNKNOWN. This is because SQL acknowledges that data can be incomplete or inapplicable and that the truth value of a predicate may therefore not be knowable. Specifically, a column can contain a null, which means that there is no known applicable value. A comparison between two values using relational operators—for example, a = 5—normally is either TRUE or FALSE. Whenever nulls are compared to other values, however, including other nulls, the Boolean value is neither TRUE nor FALSE but itself NULL.

In most respects, NULL has the same effect as FALSE. The major exception is that, while NOT FALSE = TRUE, NOT NULL = NULL. In other words, if you know that an expression is FALSE, and you negate (take the opposite of) it, then you know that it is TRUE. If you do not know whether it is TRUE or FALSE, and you negate it, you still do not know. In certain cases, three–valued logic can create problems with your programming logic if you have not accounted for it. You can treat nulls specially in SQL with the ISNULL predicate, as explained in Chapter 3 of the Oracle7 Server SQL Reference.

Creating Tables

This is how you create tables in SQL. You can use the following SQL statement to create the Customers table:

```
CREATE TABLE Customers
(cnum       integer  NOT NULL PRIMARY KEY,
FNAME       char(15)  NOT NULL,
LNAME       char(15)  NOT NULL,
ADDRESS             varchar2 );
```

After the keywords CREATE TABLE come the table's name and a parenthesized list of its columns with a definition of each. Integer, char, and varchar2 are datatypes: all of the data in a given column is always of the same type (char means a fixed and varchar2 a varying length string). For more information on SQL datatypes, see Chapter 2 of the Oracle7 Server SQL Reference.

NOT NULL and PRIMARY KEY are constraints on the columns they follow. They restrict the values you can enter in those columns. Specifically, NOT NULL *forbids you from entering nulls in the column.* PRIMARY KEY prevents you from entering duplicate values into the column and makes the column eligible to be the parent for some

MAC003111

ORCL00941787

foreign key. For more information, see "CREATE TABLE" and "CONSTRAINT clause" in Chapter 4 of the Oracle7 Server SQL Reference.

Ownership and Naming Conventions

Note that when you create a table in SQL, you own it. This means you generally have control over who has access to it, and that it is part of a schema that bears your Oracle username. A schema is a named collection of database objects under the control of a single Oracle user. Schemas inherit the names of their owners. When other users refer to an object you have created, they have to precede its name by the schema name followed by a dot (no spaces). SQL utilizes a hierarchical naming convention with the levels of the hierarchy separated by dots. In fact, you sometimes have to precede column names by table names to avoid ambiguity, in which case you also use a dot. The following is an example in the form schemaname.tablename.columnname:

```
scott.Customers.LNAME
```

You can simplify references like this by using synonyms, which are aliases for tables or other database objects. Synonyms can be private, meaning that they are part of your schema and you control their usage, or public, meaning that all users can access them. For example, you can create a synonym "Cust" for scott.Customers as follows:

```
CREATE SYNONYM Cust FOR scott.Customers;
```

This would be a private synonym, which is the default. Now you could rewrite the example above like this:

```
Cust.LNAME
```

You still have to refer to the column directly. Synonyms can only be for tables, not table components like columns.

For more information on synonyms, see "CREATE SYNONYM" in Chapter 4 of the Oracle7 Server SQL Reference. For more information on SQL naming conventions, see Chapter 2 of the Oracle7 Server SQL Reference. For more on schemas, see "CREATE SCHEMA" in Chapter 4 of the Oracle7 Server SQL Reference.

Inserting and Manipulating the Data

Which SQL statements determine the actual data content? Chiefly, three—INSERT, UPDATE, and DELETE. INSERT places rows in a table, UPDATE changes the values they contain, and DELETE removes them.

MAC003112

ORCL00941788

The INSERT Statement

For INSERT, you simply identify the table and its columns and list the values, as follows:

```
INSERT INTO Customers (cnum, FNAME, LNAME)
    VALUES (2004, 'Harry', 'Brighton');
```

This statement inserts a row with a value for every column but ADDRESS. Since you did not, in your CREATE TABLE statement, place a NOT NULL constraint on the ADDRESS column, and since you did not give that column a value here, Oracle sets this column to null. If you are inserting a value into every column of the table, and you have the values ordered as the columns are in the table, you can omit the column list. You optionally can put a SELECT statement in place of the VALUES clause of the INSERT statement to retrieve data from elsewhere in the database and duplicate it here. For more information on the INSERT and the SELECT statements, see "INSERT" and "SELECT", respectively, in Chapter 4 of the Oracle7 Server SQL Reference.

The UPDATE Statement

UPDATE is similar to SELECT in that it takes a predicate and operates on all rows that make the predicate TRUE. For example:

```
UPDATE Customers
    SET ADDRESS = null
    WHERE LNAME = 'Subchak';
```

This sets to null all addresses for customers named 'Subchak'. The SET clause of an UPDATE command can refer to current column values. "Current" in this case means the values in the column before any changes were made by this statement. For more information on the UPDATE statement, see "UPDATE" in Chapter 4 of the Oracle7 Server SQL Reference.

The DELETE Statement

DELETE is quite similar to UPDATE. The following statement deletes all rows for customers named 'Subchak':

```
DELETE FROM Customers
    WHERE LNAME = 'Subchak';
```

You can only delete entire rows, not individual values. To do the latter, use UPDATE to set the values to null. Be careful with DELETE that you do not omit the predicate; this empties the table. For more information on DELETE, see "DELETE" in Chapter 4 of the Oracle7 Server SQL Reference.

Querying Multiple Tables Through Joins

Even though it only retrieves data, SELECT is the most complex statement in SQL. One reason for this is that you can use it to query any number of tables in one statement, correlating the data in various ways. One way to do this is with a join, which is a SELECT statement that correlates data from more than one table. A join finds every possible combination of rows, such that one row is taken from each table joined. This means that three tables of ten rows each can produce a thousand rows of output (10 * 10 * 10) when joined. Typically, you use the predicate to filter the output in terms of some relationship. The most common type of join, called a natural join, filters the output in terms of the foreign key/ parent key relationship explained earlier in this chapter. For example, to see the people in the Customers table coupled with their various phone numbers from the Customers_Phone table, you could enter the following:

```
SELECT a.CNUM, LNAME, FNAME, PHONE, TYPE
    FROM Customers a, Customer_Phone b
    WHERE a.CNUM = b.CNUM;
```

In the above, a and b are range variables, also called correlation variables. They are simply alternate names for the tables whose names they follow in the FROM clause, so that a = Customers and b = Customers_Phone. You can see that here you need the range variables to distinguish Customers.CNUM from Customers_Phone.CNUM in the SELECT and WHERE clauses. Even when not needed, range variables are often convenient.

Here is the output of the natural join:

CNUM	LNAME	FNAME	PHONE	TYPE
4005	Peel	Julia	375–296–8226	home
4005	Peel	Julia	375–855–3778	beeper
4008	Lopez	Emilio	488–255–9011	home
4008	Lopez	Emilio	488–633–8591	work
4011	Lim	Kerry	577–936–8554	home

This output represents every combination of rows from the two tables *where both rows have the same CNUM value.*

Outer Joins

Notice in the preceding example that people from the Customers table who did not have phones (namely, CNUM 4007) were not selected. If a row has no match in the other table, the predicate is never true for that row. Sometimes, you do not want this effect, and you can override it by using an outer join. An outer join is a join that includes all of the rows from one of the tables joined, regardless of whether there were matches

MAC003114

ORCL00941790

in the other table. Such a join inserts nulls in the output in whichever columns were taken from the table that failed to provide matches for the outer–joined table. Here is the same query done as an outer join:

```
SELECT a.CNUM, LNAME, FNAME, PHONE, TYPE
    FROM Customers a, Customer_Phone b
    WHERE a.CNUM = b.CNUM(+);
```

This is the output of the above:

CNUM	LNAME	FNAME	PHONE	TYPE
4005	Peel	Julia	375–296–8226	home
4005	Peel	Julia	375–855–3778	beeper
4007	Subchak	Terry	NULL	NULL
4008	Lopez	Emilio	488–255–9011	home
4008	Lopez	Emilio	488–633–8591	work
4011	Lim	Kerry	577–936–8554	home

Notice that the only difference in the query is the addition of (+) to the WHERE clause. This follows the table for which nulls are to be inserted. The output from the query, then, includes at least one row for each row of the table that did not have (+) appended in the predicate.

You can also use SELECT statements to produce values for processing within queries (these are called subqueries), and you can perform standard set operations (UNION, INTERSECTION) on SELECT statement output. For more information on the SELECT statement, subqueries, and joins, see "SELECT" in Chapter 4 of the Oracle7 Server SQL Reference.

Where to Look for More Information

Oracle7 SQL is a very complex subject, and we have been able only to scratch the surface of it here. To make it easier for you to find the specific information you need to perform the task at hand, we provide the following table, which identifies where in the Oracle7 Server documentation set you can find information on specific SQL topics. Unless otherwise noted, find the headings in Chapter 4 of the Oracle7 Server SQL Reference.

To Find Out About	Look Under
aggregate data (totals, counts, averages, and so on)	SQL Functions in Chapter 3 of the Oracle7 Server SQL Reference
changing user passwords	ALTER USER
connecting to the database	CONNECT
constraints	CONSTRAINT clause; CREATE TABLE; ENABLE clause

controlling user access to objects and user actions	GRANT; REVOKE; CREATE ROLE; SET ROLE; see also Chapters 17 and 18 in the Oracle7 Server Concepts Manual
creating databases	CREATE DATABASE
creating users	CREATE USER
functions that change simple values	SQL Functions in Chapter 3 of the Oracle7 Server SQL Reference.
linking databases at different locations	CREATE DATABASE LINK; see also "Distributed Databases" in the Oracle7 Server Concepts Manual.
making changes to the data permanent	COMMIT; SET TRANSACTION; SAVEPOINT
making SQL statements execute more quickly	CREATE INDEX; see also "Indexes" in the Oracle7 Server Concepts Manual.
monitoring database usage	AUDIT
reversing (undoing) changes to the data	ROLLBACK; SET TRANSACTION; SAVEPOINT

Table 4 – 3 Guide to Further Information on SQL

What is PL/SQL?

As you may have noticed from the preceding section, SQL statements are very concise and powerful, but do not do more as a group than they do individually. Generally speaking, SQL statements operate independently, having little effect on one another. This is of limited use for writing programs, where you must create a body of code that is going to vary its behavior according to the data and to user or other input. To develop applications with SQL, you generally have to either interface it to a standard programming language such as C, or extend it so that it becomes a useful programming language in itself. Oracle supports both approaches, but the latter approach has many advantages that are relevant to the Web, and is therefore the approach that the Oracle WebServer takes.

PL/ SQL, then, is an application–development language that is a superset of SQL, supplementing it with standard programming–language features that include the following:

MAC003116

ORCL00941792

- block (modular) structure
- flow–control statements and loops
- variables, constants, and types
- structured data
- customized error handling

Another feature of PL/ SQL is that it allows you to store compiled code directly in the database. This enables any number of applications or users to share the same functions and procedures. In fact, once a given block of code is loaded into memory, any number of users can use the same copy of it simultaneously (although behavior is as though each user had her own copy), which is useful for the Oracle WebServer. PL/ SQL also enables you to define triggers, which are subprograms that the database executes automatically in response to specified events.

Unlike SQL, PL/ SQL is not an industry standard, but is an exclusive product of Oracle Corporation.

> Note: For the sake of efficiency, PL/ SQL code is compiled prior to runtime. It cannot refer at compile time to objects that do not yet exist, and, for that reason, the one part of SQL that PL/ SQL does not include is DDL (Data Definition Language)—the statements, such as CREATE TABLE, that create the database and the objects it contains. However, you can work around this by using the package DBMS_SQL, included with the server, to generate the DDL code itself dynamically at runtime. For more information, see "Using DDL and Dynamic SQL" in the PL/SQL User's Guide and Reference.

Basic Structure and Syntax

PL/ SQL, like many programming languages, groups statements into units called blocks. These can either be named, in which case they are called subprograms, or unnamed, in which case they are anonymous blocks. Subprograms can be either functions or procedures. The difference between these, as in most languages, is that a function is used in an expression and returns a value to that expression, while a procedure is invoked as a standalone statement and passes values to the calling program only through parameters. Subprograms can be nested within one another and can be grouped in larger units called packages.

MAC003117

ORCL00941793

A block has three parts:

- A DECLARE section. This is where you define local variables, constants, types, exceptions, and nested subprograms. PL/ SQL has a forward declaration, but you can use it only for subprograms. Therefore, you must define all variables, constant, and types before referencing them. For more information on forward declarations, see "Declaring Subprograms" in the PL/SQL User's Guide and Reference.
- An EXECUTABLE section. This is the actual code that the block executes. This is the only part of the block that must always be present.
- An EXCEPTION section. This is a section for handling runtime errors and warnings.

These divisions are explained further in the sections that follow.

The DECLARE Section

The DECLARE section begins with the keyword DECLARE and ends when the keyword BEGIN signals the arrival of the EXECUTABLE section. You can declare types, constants, variables, exceptions, and cursors in any order, as long as they are declared before they are referenced in another definition. You declare subprograms last. A semi–colon terminates each definition.

Datatypes

PL/ SQL provides a number of predefined datatypes for variables and constants. It also enables you to define your own types, which are subtypes of the predefined types. The types fall into the following three categories:

- Scalar. These include all string, number, and binary types. All of the SQL datatypes, which are the datatypes that you can store in the database, fall into this category. To find out about these datatypes, see "Datatypes" in the Oracle7 Server SQL Reference.
- Composite. These are structured datatypes, which is to say data structures that have components you can address independently. The PL/ SQL composite types are TABLE (which is distinct from both database and HTML tables) and RECORD. These types are explained later in this chapter.
- Reference. There is one kind of reference datatype—REF CURSOR—which is a pointer to a cursor. Cursors are explained

MAC003118

ORCL00941794

later in this chapter. For more information on the REF CURSOR datatype, see "Using Cursor Variables" in the PL/SQL User's Guide and Reference.

For a list and explanation of all PL/SQL datatypes, see "Datatypes" in the PL/SQL User's Guide and Reference.

In many cases, you can convert from one datatype to another, either explicitly or automatically. The possible conversions and the procedure involved are explained in the PL/SQL User's Guide and Reference under "Datatype Conversion".

You can also define a variable so that it inherits its datatype from a database column or from another variable or constant, as explained in the next section.

Declaring Variables

For variables, provide the name, datatype, and any desired attributes, as follows:

```
cnum INTEGER(5) NOT NULL;
```

This declares a five–digit integer called cnum that will not accept nulls. The use of case above serves to distinguish keywords from identifiers; PL/SQL is not case–sensitive. NOT NULL is the only SQL constraint that you can use as a PL/SQL attribute.

> Note: PL/SQL initializes all variables to null. Therefore, a NOT NULL variable, such as the above, produces an error if referenced before it is assigned a value.

Optionally, you can assign an initial value to the variable when you declare it by following the datatype specification with an assignment, as follows:

```
cnum INTEGER(5) := 254;
```

This sets cnum to the initial value of 254. Alternatively, you can use the keyword DEFAULT in place of the assignment operator := to achieve the same effect. For more information on setting defaults, see "Declarations" in the PL/SQL User's Guide and Reference.

Inheriting Datatypes

To have the variable inherit the datatype of a database column or of another variable, use the %TYPE attribute in place of a declared datatype, as follows:

```
snum cnum%TYPE;
```

This means that snum inherits the datatype of cnum. You can inherit datatypes from database columns in the same way, by using the notation tablename.columnname in place of the variable name. Normally, you do this if the variable in question is to place values in or retrieve them from the column. The advantages are that you need not know the exact datatype the column uses and that you need not change your code if the datatype of that column changes. If you do not own the table containing the column, precede the tablename with the schemaname, as described under "Naming Conventions" elsewhere in this chapter. For more information on %TYPE assignments, see "Declarations" in the PL/SQL User's Guide and Reference.

Declaring Constants

You declare constants the same way as variables, except for the addition of the keyword CONSTANT and the mandatory assignment of a value. Constants do not take attributes other than the value. An example follows:

```
interest CONSTANT REAL(5,2) := 759.32;
```

Defining Types

User–defined types in PL/SQL are subtypes of existing datatypes. They provide you with the ability to rename types and to constrain them by specifying for your subtype lengths, maximum lengths, scales, or precisions, as appropriate to the standard datatype on which the subtype is based. For more information on the datatype parameters, see "Datatypes" in Chapter 2 of the Oracle7 Server SQL Reference. For more information on PL/SQL datatypes, see "Datatypes" in the PL/SQL User's Guide and Reference. You can also use the %TYPE attribute in defining a subtype. Here is an example:

```
SUBTYPE shortnum IS INTEGER(3);
```

This defines SHORTNUM as a 3–digit version of INTEGER. For more information see "User–Defined Subtypes" in the PL/SQL User's Guide and Reference.

Scope and Visibility

Nested subprograms, defined in the DECLARE section, can be called from either of the other sections, but only from within the same block where they are defined or within blocks contained in that block. Variables, constants, types, and subprograms defined within a block are local to that block, and their definitions are not meaningful outside of it. Objects that are local to a block may be used by subprograms

MAC003120

ORCL00941796

contained at any level of nesting in that same block. Such objects are global to the block that calls them.

The area of a program within which an object can be used is called the object's scope. An object's scope is distinct from its visibility. The former is the area of the program that can reference the object; the latter is the, generally smaller, portion that can reference it without qualification.

An ambiguous reference can arise because objects or subprograms contained in different blocks can have the same names, even if they have overlapping scopes. When this happens, the reference by default means the object most local in scope—in other words, the first one PL/ SQL finds by starting in the current block and working out to the enclosing ones. Qualification is the method used to override this. It is similar to the system of qualification used for database objects, as explained under "Naming Conventions" elsewhere in the chapter. To qualify an object's name, precede it with the name of the subprogram where it is declared, followed by a dot, as follows:

```
relocate.transmit(245, destination);
```

This invokes a procedure called transmit declared in some subprogram called relocate. The subprogram relocate must be global to the block from which it is called.

Data Structures

PL/ SQL provides two structured datatypes: TABLE and RECORD. It also provides a data structure called a cursor that holds the results of queries. Cursors are different from the other two in that you declare variables and constants to be of type TABLE or RECORD just as you would any other datatype. Cursors, on the other hand, have their own syntax and their own operations. Explanations of these types follow:

PL/SQL Tables

These are somewhat similar to database tables, except that they always consist of two columns: a column of values and a primary key. This also makes them similar to one–dimensional arrays, with the primary key functioning as the array index. Like SQL tables, PL/ SQL tables have no fixed allocation of rows, but grow dynamically. One of their main uses is to enable you to pass entire columns of values as parameters to subprograms. With a set of such parameters, you can pass an entire table. The primary key is always of type BINARY_INTEGER, and the values can be of any scalar type.

You declare objects of type TABLE in two stages:

MAC003121

ORCL00941797

1. You declare a subtype using the following syntax:

```
TYPE type_name IS TABLE OF
    datatype_spec
    [ NOT NULL ]
    INDEX BY BINARY INTEGER;
```

Where datatype_spec means the following:

```
datatype | variablename%TYPE | tablename.columnname%TYPE
```

In other words, you can either specify the type of values directly or use the %TYPE attribute (explained under "Declaring Variables", elsewhere in this chapter) to inherit the datatype from an existing variable or database column.

2. You assign objects to this subtype in the usual way. You cannot assign initial values to tables, so the first reference to the table in the EXECUTABLE section must provide it at least one value.

When you reference PL/SQL tables, you use an array–like syntax of the form:

```
column_value(primary_key_value)
```

In other words, the third row (value) of a table called "Employees" would be referenced as follows:

```
Employees(3)
```

You can use these as ordinary expressions. For example, to assign a value to a table row, use the following syntax:

```
Employees(3) := 'Marsha';
```

For more information, see "PL/SQL Tables" in the PL/SQL User's Guide and Reference.

Records

As in many languages, these are data structures that contain one or more fields. Each record of a given type contains the same group of fields with different values. Each field has a datatype, which can be RECORD. In other words, you can nest records, creating data structures of arbitrary complexity. As with tables, you declare records by first declaring a subtype, using the following syntax:

```
TYPE record_type IS RECORD
    (fieldname datatype[, fieldname datatype]...);
```

The second line of the above indicates a parenthesized, comma–separated, list of fieldnames followed by datatype specifications. The datatype specifications can be direct or be inherited

MAC003122

ORCL00941798

using the %TYPE attribute, as shown for TABLE and as explained under "Declaring Variables", elsewhere in this chapter.

You can also define a record type that automatically mirrors the structure of a database table or of a cursor, so that each record of the type corresponds to a row, and each field in the record corresponds to a column. To do this, use the %ROWTYPE attribute with a table or cursor name in the same way you would the %TYPE attribute with a variable, or column. The fields of the record inherit the column names and datatypes from the cursor or table. For more information, see "Records" and "%ROWTYPE Attribute" in the PL/SQL User's Guide and Reference.

Cursors

A cursor is a data structure that holds the results of a query (a SELECT statement) for processing by other statements. Since the output of any query has the structure of a table, you can think of a cursor as a temporary table whose content is the output of the query.

When you declare a cursor, you associate it with the desired query. When you want to use that cursor, you open it, executing the associated query and filling the cursor with its results. You then fetch each row of the query's output in turn for processing by other statements in the program. You can also use a cursor to update a table's contents. To do this, use a FOR UPDATE clause to lock the rows in the table. See "Using FOR UPDATE" in the PL/SQL User's Guide and Reference for more information. Sometimes, you may need to use cursor variables, which are not associated with a query until runtime. This is a form of dynamic SQL.

For more information on cursor variables, see "Using Dynamic SQL" in the Oracle7 Server Application Developers Guide and "Cursor Variables" in the PL/SQL User's Guide and Reference.

For more information on cursors in general, see "Cursors" in the PL/SQL User's Guide and Reference. See also "DECLARE CURSOR," "OPEN", and "FETCH" in the Oracle7 Server SQL Reference.

You can simplify some cursor operations by using cursor FOR loops. For more information on these, see "Using Cursor FOR Loops" in the PL/SQL User's Guide and Reference.

Exceptions

You also use the DECLARE section to define your own error conditions, called "exceptions". Explanation of this is deferred until the "EXCEPTION Section" portion of this chapter.

MAC003123

ORCL00941799

Declaring Subprograms

You must place all subprogram declarations at the end of the declare section, following all variable, constant, type, and exception declarations for the block. The syntax is as follows:

```
PROCEDURE procedure_name (parameter_name datatype, parameter_name
datatype...) IS
            {local declarations}
    BEGIN {executable code}
    EXCEPTION
    END;
```

> Note: For subprograms, the keyword DECLARE is omitted before the local declarations. Place local declarations before the keyword BEGIN, as shown.

The names you give the parameters in the declaration are the names that the procedure itself uses to refer to them. These are called the formal parameters. When the procedure is invoked, different variables or constants may be used to pass values to or from the formal parameters; these are called the actual parameters.

When calling the procedure, you can use each parameter for input of a value to the procedure, output of a value from it, or both. These correspond to the three parameter modes: IN, OUT, and IN/ OUT. For more information, see "Parameter Modes" in the PL/SQL User's Guide and Reference.

When you call the procedure, you can match the actual to the formal parameters either implicitly, by passing them in the same order they are given in the declaration, or explicitly, by naming the formal followed by the actual parameter as shown:

```
transmit(destination => address);
```

This invokes a procedure called transmit, assigning the value of address as the actual parameter for the formal parameter destination. This implies that the parameter destination is used within the transmit procedure and that the parameter address is used outside of it. Usually, it is good programming practice to use different names for matching formal and actual parameters. For more information on this, see "Positional and Named Notation" in the PL/SQL User's Guide and Reference.

Functions are the same, except for the addition of a return value, specified as follows:

```
FUNCTION function_name (parameter_name, parameter_name
datatype...)
    RETURN datatype IS
```


MAC003124

ORCL00941800

```
{local declarations}
BEGIN {executable code}
EXCEPTION {local exception handlers}
END;
```

Again, line breaks are only for readability. A RETURN statement in the executable section actually determines what the return value is. This consists of the keyword RETURN followed by an expression. When the function executes the RETURN statement, it terminates and passes the value of that expression to whichever statement called it in the containing block.

You can also use the RETURN statement without an expression in a procedure to force the procedure to exit.

For more information on procedures and functions, see "Declaring Subprograms" in the PL/SQL User's Guide and Reference.

The EXECUTABLE Section

The executable section is the main body of code. It consists primarily of SQL statements, flow control statements, and assignments. SQL statements are explained earlier in this chapter; assignments and flow–control statements are explained in the sections that follow.

Assignments

The assignment operator is :=. For example, the following statement assigns the value 45 to the variable a:

```
a := 45;
```

Character strings should be set off with single quotes (') as in all expressions. An example follows:

```
FNAME := 'Clair';
```

There are other examples of assignments in other parts of this chapter.

Flow Control

PL/SQL supports the following kinds of flow–control statements:

- IF statements. These execute a group of one or more statements based on whether a condition is TRUE.

MAC003125

ORCL00941801

- Basic loops. These repeatedly execute a group of one or more statements until an EXIT statement is reached.
- FOR loops. These repeatedly execute a group of one or more statements a given number of times or until an EXIT statement is reached.
- WHILE loops. These repeatedly execute a group of one or more statements until a particular condition is met or an EXIT statement is reached.
- GOTO statements. These pass execution directly to another point in the code, exiting loops and enclosing blocks as necessary. Use these sparsely, as they make code difficult to read and debug.

If you know other programming languages, you probably are familiar with most or all of these types of statements. The following sections describe the PL/ SQL versions of them in greater detail. For more information on any of these, see "Control Structures" in the PL/SQL User's Guide and Reference.

You can nest flow control statements within one another to any level of complexity.

IF Statements

These are similar to the IF statement in many other languages, except that they use predicates, which are three–valued Boolean expressions like the SQL predicates discussed earlier in this chapter. In most respects, a Boolean NULL behaves like a Boolean FALSE, except that negation does not make it positive, but leaves it NULL.

The IF statement has the following forms:

```
1   IF <condition> THEN <statement–list>;
    END IF;
```

If the condition following IF is TRUE, PL/ SQL executes the statements in the list following THEN. A semicolon terminates this list. END IF (not ENDIF) is mandatory and terminates the entire IF statement. Here is an example:

```
IF balance > 500 THEN send_bill(customer);
END IF;
```

We are assuming that send_bill is a procedure taking a single parameter.

```
2   IF <condition> THEN <statement–list>;
    ELSE <statement–list>;
    END IF;
```


MAC003126

ORCL00941802

This is the same as the preceding statement, except that, if that condition is FALSE or NULL, PL/ SQL executes the statement list following ELSE instead of that following THEN.

```
3   IF <condition> THEN <statement-list>;
    ELSIF <condition> THEN <statement-list>;
    ELSIF <condition> THEN <statement-list>; . . . . .
    ELSE <statement-list>;
    END IF;
```

You can include any number of ELSIF (not ELSEIF) conditions. Each is tested only if the IF condition and all preceding ELSIF conditions are FALSE or NULL. As soon as PL/ SQL finds an IF or ELSIF condition that is TRUE, it executes the associated THEN statement list and skips ahead to END IF. The ELSE clause is optional, but, if included, must come last. It is executed if all preceding IF and ELSIF conditions are FALSE or NULL.

NULL Statements

If you do not want an action to be taken for a given condition, you can use the NULL statement, which is not to be confused with database nulls, Boolean NULLs, or the SQL predicate IS NULL. The syntax of this statement is simply:

```
NULL;
```

The statement performs no action, but fulfills the syntax requirement that a statement list must follow every THEN keyword. In some cases, you can also use it to increase the readability of your code. For more information on the NULL statement, see "NULL Statement" in the PL/SQL User's Guide and Reference.

Basic Loops

A basic loop is a loop that keeps repeating until an EXIT statement is reached. The EXIT statement must be within the loop itself. If no EXIT (or GOTO) statement ever executes, the loop is infinite. An example follows:

```
credit := 0;
 LOOP
    IF c = 5 THEN EXIT;
    END IF;
    credit := credit + 1;
END LOOP;
```

This loop keeps incrementing credit until it reaches 5 and then exits. An alternative to placing an exit statement inside an IF statement is to use the EXIT–WHEN syntax, as follows:

```
EXIT WHEN credit = 5;
```

This is equivalent to the earlier IF statement.

> Note: The EXIT statement cannot be the last statement in a PL/ SQL block. If you want to exit a PL/ SQL block before its normal end is reached, use the RETURN statement. For more information, see "RETURN Statement" in the PL/SQL User's Guide and Reference.

FOR Loops

A FOR loop, as in most languages, repeats a group of statements a given number of times. The following FOR loop is equivalent to the example used for basic loops, except that it also changes a variable called interest.

```
FOR credit IN 1..5 LOOP
    interest := interest * 1.2;
END LOOP;
```

The numbers used to specify the range (in this case, 1 and 5) can be variables, so you can let the number of iterations of the loop be determined at runtime if you wish.

WHILE Loops

A WHILE loop repeats a group of statements until a condition is met. Here is a WHILE loop that is the equivalent of the preceding example:

```
credit := 1;
WHILE credit <= 5 LOOP
    interest := interest * 1.2;
    credit := credit + 1;
END LOOP;
```

Unlike some languages, PL/ SQL has no structure, such as REPEAT–UNTIL, that forces a LOOP to execute at least once. You can create this effect, however, using either basic or WHILE loops and setting a variable to a value that will trigger the loop, as in the above example. For more information on loops, see "Iterative Control" in the PL/SQL User's Guide and Reference.

MAC003128

ORCL00941804

GOTO Statements

A GOTO statement immediately transfers execution to another point in the program. The point in the program where the statement is to arrive must be preceded by a label. A label is an identifier for a location in the code. It must be unique within its scope and must be enclosed in double angle brackets, as follows:

```
<<this_is_a_label>>
```

You only use the brackets at the target itself, not in the GOTO statement that references it, so a GOTO statement transferring execution to the above label would be:

```
GOTO this_is_a_label;
```

> Note: An EXIT statement can also take a label, if that label indicates the beginning of a loop enclosing the EXIT statement. You can use this to exit several nested loops at once. See "Loop Labels" in the PL/SQL User's Guide and Reference for more information.

A GOTO statement is subject to the following restrictions:

- It must branch to an executable statement, not, for example, an END.
- It cannot branch to a point within the body of IF or a LOOP statement, unless it is contained in the body of that statement itself.
- It cannot branch to a subprogram or enclosing block of the present block (with one exception, explained shortly).
- It cannot branch from one IF statement clause to another. That is to say, it cannot jump between THEN, ELSIF, and ELSE clauses that are part of the same IF statement.
- It cannot branch from the EXCEPTION section to the EXECUTABLE section of the same block.
- It can, however, branch from the EXCEPTION section of a block to the EXECUTABLE section of an enclosing block, which is the exception to the third rule above.

The EXCEPTION Section

The EXCEPTION section follows the END that matches the BEGIN of the EXECUTABLE section and begins with the keyword EXCEPTION.

MAC003129

ORCL00941805

It contains code that responds to runtime errors. An exception is a specific kind of runtime error. When that kind of error occurs, you say that the exception is raised. An exception handler is a body of code designed to handle a particular exception or group of exceptions. Exception handlers, like the rest of the code, are operative only once the code is compiled and therefore can do nothing about compilation errors.

There are two basic kinds of exceptions: predefined and user–defined. The predefined exceptions are provided by PL/ SQL in a package called STANDARD. They correspond to various runtime problems that are known to arise often—for example, dividing by zero or running out of memory. These are listed in the PL/SQL User's Guide and Reference under "Predefined Exceptions".

The Oracle Server can distinguish between and track many more kinds of errors than the limited set that STANDARD predefines. Each of Oracle's hundreds of messages are identified with a number, and STANDARD has simply provided labels for a few of the common ones. You can deal with the other messages in either or both of two ways:

- You can define your own exception labels for specified Oracle messages using a pragma (a compiler directive). This procedure will be explained shortly.
- You can define a handler for the default exception OTHERS. Within that handler, you can identify the specific error by accessing the built–in functions SQLCODE and SQLERRM, which contain, respectively, the numeric code and a prose description of the message.

You can also define your own exceptions as will be shown. It is usually better, however, to use Oracle exceptions where possible, because then the conditions are tested automatically when each statement is executed, and an exception is raised if the error occurs.

Declaring Exceptions

PL/ SQL predefined exceptions, of course, need not be declared. You declare user–defined exceptions or user–defined labels for Oracle messages in the DECLARE section, similarly to variables. An example follows:

```
customer_deceased  EXCEPTION;
```

In other words, an identifier you choose followed by the keyword EXCEPTION. Notice that all this declaration has done is provide a name. The program still has no idea when this exception should be

MAC003130

ORCL00941806

raised. In fact, there is at this point no way of telling if this is to be a user–defined exception or simply a label for an Oracle message.

Labeling Oracle Messages

If a previously–declared exception is to be a label for an Oracle error, you must define it as such with a second statement in the DECLARE section, as follows:

```
PRAGMA EXCEPTION_INIT (exception_name, Oracle_error_number);
```

A PRAGMA is a instruction for the compiler, and EXCEPTION_INIT is the type of PRAGMA. This tells the compiler to associate the given exception name with the given Oracle error number. This is the same number to which SQLCODE is set when the error occurs. The advantage of this over defining your own error condition is that you pass the responsibility for determining when the error has occurred and raising the exception to Oracle. You can find the numeric codes and explanations for Oracle messages in Oracle7 Server Messages.

User–Defined Exceptions

If the declared condition is not to be a label for an Oracle error, but a user–defined error, you do not need to put another statement referring to it in the DECLARE section. In the EXECUTABLE section, however, you must test the situation you intend the exception to handle whenever appropriate and raise the condition manually, if needed. Here is an example:

```
IF cnum < 0 THEN RAISE customer_deceased;
```

You can also use the RAISE statement to force the raising of predefined exceptions. For more information, see "Error Handling" in the PL/SQL User's Guide and Reference.

Handling Exceptions

Once an exception is raised, whether explicitly with a RAISE statement or automatically by Oracle, execution passes to the EXCEPTION section of the block, where the various exception handlers reside. If a handler for the raised exception is not found in the current block, enclosing blocks are searched until one is found. If PL/ SQL finds an OTHERS handler in any block, execution passes to that handler. An OTHERS handler must be the last handler in its block. If no handler for an exception is found, Oracle raises an unhandled exception error. Note: this does not automatically roll back (undo) changes made by the

subprogram, which might leave the database in an undesirable intermediate state.

This is the syntax of an exception handler:

```
WHEN exception_condition THEN statement_list;
```

The exception is the identifier for the raised condition. If desired, you can specify multiple exceptions for the same handler, separated by the keyword OR. The exception can be either one the package STANDARD provided or one you declared. The statement list does what is appropriate to handle the error—writing information about it to a file, for example—and arranges to exit the block gracefully if possible. Although exceptions do not necessarily force program termination, they do force the program to exit the current block. You cannot override this with a GOTO statement. You can use a GOTO within an exception handler, but only if its destination is some enclosing block.

> Note: If you have an error prone statement and want execution to continue following this statement, even when an exception occurs, put the statement, including the appropriate exception handlers, in its own block, so that the current block becomes the enclosing block.
>
> Note: If an exception occurs in the DECLARE section or the EXCEPTION section itself, local exception handlers cannot address it; execution passes automatically to the EXCEPTION section of the enclosing block.

Storing Procedures and Functions in the Database

To have a procedure or function stored as a database object, you issue a CREATE PROCEDURE or a CREATE FUNCTION statement directly to the server using SQL*PLUS or Server Manager. The easy way to do this is to use your ordinary text editor to produce the CREATE statement and then to load it as a script. This process is explained under "Creating Stored Procedures and Functions" in the Oracle7 Server Application Developers Guide. This approach is recommended because you often create entire groups of procedures and functions together. These groups are called "packages" and are explained later in this chapter.

The syntax for these statements is slightly different than that used to declare subprograms in PL/ SQL, as the following example shows:

```
CREATE PROCEDURE fire_employee (empno INTEGER) IS
    BEGIN
```


MAC003132

ORCL00941808

```
    DELETE FROM Employees WHERE enum = empno;
END;
```

As you can see, the main difference is the addition of the keyword CREATE. You also have the option of replacing the keyword IS with AS, which does not affect the meaning. To replace an existing procedure of the same name with this procedure (as you frequently may need to do during development and testing), you can use CREATE OR REPLACE instead of simply CREATE. This destroys the old version, if any, without warning.

Privileges Required

A stored procedure or function (for the rest of this discussion, "procedure" shall mean "procedure or function" unless otherwise indicated or clear from context) is a database object like a table. It resides in a schema, and its use is controlled by privileges. To create a procedure and have it compile successfully, you must meet the following conditions:

- If the procedure is to be in your own schema, you must have the CREATE PROCEDURE or the CREATE ANY PROCEDURE system privilege. These privileges apply as well to functions.
- If the procedure is to be in a schema you do not own, you must have the CREATE ANY PROCEDURE system privilege.
- You must have the object privileges necessary to perform all operations contained in the procedure. You must have these privileges as a user, not through roles. If your privileges change after you have created the procedure, the procedure may no longer be executable.

To enable others to use the procedure, grant them the EXECUTE privilege on it using the SQL statement GRANT (see "GRANT" in Chapter 4 of the Oracle7 Server SQL Reference). When these users execute the procedure, they do so under your privileges, not their own. Therefore, you do not have to grant them the privileges to perform these actions outside the control of the procedure, which is a useful security feature. To enable all users to use the procedure, grant EXECUTE to PUBLIC. The following example permits all users to execute a procedure called show_product.

```
GRANT EXECUTE ON show_product TO PUBLIC;
```

Of course, the public normally does not execute such a procedure directly. This statement enables you to use the procedure in your PL/SQL code that is to be publicly executable. If multiple users access

the same procedure simultaneously, each gets his own instance. This means that the setting of variables and other activities by different users do not affect one another.

For more information on privileges and roles, see "GRANT" in Chapter 4 of the Oracle7 Server SQL Reference. There are three versions of GRANT listed—one each for object privileges, system privileges, and roles.

For more information on storing procedures and functions in the database, see "Storing Procedures and Functions" in the Oracle7 Server Application Developers Guide and see "CREATE FUNCTION" and "CREATE PROCEDURE" in the Oracle7 Server SQL Reference.

Packages

A package is a group of related PL/SQL objects (variables, constants, types, and cursors) and subprograms that is stored in the database as a unit. Being a database object, a package resides in a schema, and its use is controlled by privileges. Among its differences from regular PL/SQL programs are that a package as such does not do anything. It is a collection of subprograms and objects, at least some of which are accessible to applications outside of it. It is the subprograms in the package that contain the executable code. A package has the following two parts:

- The package specification is the public interface to the package. It declares all objects and subprograms that are to be accessible from outside the package. Packages do not take parameters, so these constitute the entire public interface.
- The package body is the internal portion of the package. It contains all objects and subprograms that are to be local to the package. It also contains definitions of the public cursors and subprograms. The package specification declares but does not define these.

One of the advantages of using packages is that the package specification is independent of the body. You can change the body and, so long as it still matches the specification, no changes to other code are needed, nor will any other references become invalid.

Packages cannot be nested, but they can call one another's public subprograms and reference one another's public objects.

MAC003134

ORCL00941810

Instantiation of Packages

It is important to realize that a package is instantiated once for a given user session. That is to say, the values of all variables and constants, as well as the contents and state of all cursors, in a package, once set, persist for the duration of the session, even if you exit the package. When you reenter the package, these objects retain the values and state they had before, unless they are explicitly reinitialized. Of course, another user has another session and therefore another set of values. Nonetheless, a global reinitialization of a package's objects for you does not take place until you disconnect from the database.

There is an exception, however. When one package calls another, execution of the second has a dependency on the first. If the first is invalidated, for example because its creator loses a privilege that the package requires, the second, while not necessarily invalidated, becomes deinstantiated. That is to say, all its objects are reinitialized.

> Note: In PL/ SQL, stored procedures and packages are automatically recompiled if changes to the database mandate it. For example, a change to the datatype of a column can automatically cascade to a variable referencing that column if the former is declared with the %TYPE attribute, but that change requires that the PL/ SQL procedure declaring that variable be recompiled. So long as the PL/ SL code as written is still valid, the recompilation occurs automatically and invisibly to the user.

Creating Packages

To create a package, you use the SQL statement CREATE PACKAGE for the specification and CREATE PACKAGE BODY for the body. You must create the specification first. Sometimes, a package may consist of only public variables, types, and constants, in which case no body is necessary. Generally, however, you use both parts.

> Note: Before you can create a package, the special user SYS must run the SQL script DBMSSTDX.SQL. The exact name and location of this script may vary according to your operating system. Contact your database administrator if you are not sure this script has been run.

Creating the Package Specification

The syntax of the CREATE PACKAGE statement is as follows:

MAC003135

ORCL00941811

```
CREATE [OR REPLACE] PACKAGE package_name IS
    {PL/SQL declarations}
    END;
```

The optional OR REPLACE clause operates just as it does for stored procedures, as explained elsewhere in this chapter. The PL/SQL declarations are as outlined under DECLARE SECTION elsewhere in this chapter, except that the keyword DECLARE is not used and that the subprogram and cursor declarations are incomplete. For subprograms, you provide only the name, parameters, and, in the case of functions, the datatype of the return value. For cursors, provide the name and a new item called the return type. This approach hides the implementation of these objects from the public while making the objects themselves accessible.

The syntax for declaring a cursor with a return type is as follows:

```
CURSOR c1 IS RETURN return_type;
```

The return type is always some sort of record type that provides a description of the cursor's output. The structure of this record is to mirror the structure of the cursor's rows. You can specify it using any of the following:

- A record subtype previously defined and in scope. For more information, see "Records" elsewhere in this chapter.
- A type inherited from such a record subtype using the %TYPE attribute. For more information, see "Declaring Variables" elsewhere in this chapter.
- A type inherited from a table, most likely the table the cursor queries, using the %ROWTYPE attribute. For more information, see "Records" elsewhere in this chapter.
- A type inherited from a cursor using the %ROWTYPE attribute. For more information, see "Records" elsewhere in this chapter.

For more information, see CREATE PACKAGE in Chapter 4 of the Oracle7 Server SQL Reference, "Packages" in the PL/SQL User's Guide and Reference, and "Using Procedures and Packages" in the Oracle7 Server Application Developers Guide.

Creating the Package Body

To create the package body, use the CREATE PACKAGE BODY statement. The syntax is as follows:

```
CREATE [OR REPLACE] PACKAGE BODY package_name IS
    {PL/SQL declarations}
    END;
```


MAC003136

ORCL00941812

Since a package as such does not do anything, the PL/ SQL code still consists only of a DECLARE section with the keyword DECLARE omitted. It is the subprograms within the package that contain the executable code. Variables, constants, types, and cursors declared directly (in other words, not within a subprogram) in the declare section have a global scope within the package body. Variables, constants, and types already declared in the package specification are public and should not be declared again here.

Public cursors and subprograms, however, must be declared again here, as their declarations in the specification is incomplete. This time the declarations must include the PL/ SQL code (in the case of subprograms) or the query (in the case of cursors) that is to be executed. For subprograms, the parameter list must match that given in the package specification word for word (except for differences in white space). This means, for example, that you cannot specify a datatype directly in the specification and use the %TYPE attribute to specify it in the body.

You can create an initialization section at the end of the package body. This is a body of executable code—chiefly assignments—enclosed with the keywords BEGIN and END. Use this to initialize constants and variables that are global to the package, since otherwise they could be initialized only within subprograms, and you have no control of the order in which subprograms are called by outside applications. This initialization is performed only once per session.

For more information, see CREATE PACKAGE BODY in the Oracle7 Server SQL Reference, "Packages" in the PL/SQL User's Guide and Reference, and "Using Procedures and Packages" in the Oracle7 Server Application Developers Guide.

Overloading Subprograms

Within a package, subprogram names need not be unique, even at the same level of scope. There can be multiple like-named subprograms in the same declare section, provided that the parameters that they take differ in number, order, or datatype and that, when the procedures are called, the values passed by the calling procedure (the actual parameters) match or can be automatically converted to the datatypes specified in the declaration (the formal parameters). To find out which datatypes PL/ SQL can convert automatically, look under "Datatype Conversion" in the PL/SQL User's Guide and Reference.

The reason this is permitted is so you can overload subprograms. Overloading permits you to have several versions of a procedure that are conceptually similar but behave differently with different

MAC003137

ORCL00941813

parameters. This is one of the properties of object–oriented programming. For more information on overloading, see "Overloading" in the PL/SQL User's Guide and Reference.

Database Triggers

Triggers are blocks of PL/ SQL code that execute automatically in response to events. Database triggers reside in the database and respond to changes in the data. They are not to be confused with application triggers, which reside in applications and are beyond the scope of this discussion. Database triggers are a technology that for the most part has superceded application triggers.

You create triggers as you do stored procedures and packages, by using your text editor to write scripts that create them and then using SQL*Plus or Server Manager to run these scripts. A trigger is like a package in that:

- It takes no parameters as such. It refers to, responds to, and possibly affects the data in the database.
- It cannot be directly called like a procedure. To fire (execute) a trigger, you must make the database change to which it responds. If you only want to test the trigger, you can rollback (undo) the database change that you made after the trigger fires.

Triggers can be classified in three ways:

- INSERT triggers, UPDATE triggers, and DELETE triggers. This is a classification based on the statement to which the trigger responds. The categories are not mutually exclusive, meaning one trigger can respond to any or all of these statements.
- Row triggers and statement triggers. Any of the above statements can affect any number of rows in a table at once. A row trigger is fired once for each row affected. A statement trigger is fired once for each statement, however many rows it affects.
- BEFORE triggers and AFTER triggers. This specifies whether the trigger is fired before or after the data modification occurs.

As you can see, all three of these classifications apply to all triggers, so that there are, for example, BEFORE DELETE OR INSERT statement triggers and AFTER UPDATE row triggers.

MAC003138

ORCL00941814

Creating Triggers

The syntax of the CREATE TRIGGER statement is as follows:

```
CREATE [OR REPLACE] TRIGGER trigger_name
    BEFORE | AFTER
    DELETE | INSERT | UPDATE [OF column_list]
    ON table_name
    [ FOR EACH ROW [ WHEN predicate ] ]
    {PL/SQL block};
```

In the above, square brackets ([ ]) enclose optional elements. Vertical bars (|) indicate that what precedes may be replaced by what follows.

In other words, you must specify the following:

- A trigger name. This is used to alter or drop the trigger. The trigger name must be unique within the schema.
- BEFORE or AFTER. This specifies whether this is a BEFORE or AFTER trigger.
- INSERT, UPDATE, or DELETE. This specifies the type of statement that fires the trigger. If it is UPDATE, you optionally can specify a list of one or more columns, and only updates to those columns fire the trigger. In such a list, separate the column names with commas and spaces. You may specify this clause more than once for triggers that are to respond to multiple statements; if you do, separate the occurrences with the keyword OR surrounded by white space.
- ON table_name. This identifies the table with which the trigger is associated.
- PL/SQL Block. This is an anonymous PL/SQL block containing the code the trigger executes.

You optionally can specify the following:

- OR REPLACE. This has the usual effect.
- FOR EACH ROW [WHEN predicate]. This identifies the trigger as a row trigger. If omitted, the trigger is a statement trigger. Even if this clause is included, the WHEN clause remains optional. The WHEN clause contains a SQL (not a PL/SQL) predicate that is tested against each row the triggering statement alters. If the values in that row make the predicate TRUE, the trigger is fired; else it is not. If the WHEN clause is omitted, the trigger is fired for each altered row.

Here is an example:

MAC003139

ORCL00941815

```
CREATE TRIGGER give_bonus
    AFTER UPDATE OF sales
    ON salespeople
    FOR EACH ROW WHEN sales > 8000.00
    BEGIN
    UPDATE salescommissions SET bonus = bonus + 150.00;
    END;
```

This creates a row trigger called give_bonus. Every time the sales column of the salespeople table is updated, the trigger checks to see if it is over 8000.00. If so, it executes the PL/ SQL block, consisting in this case of a single SQL statement that increments the bonus column in the salescommissions table by 150.00.

Privileges Required

To create a trigger in your own schema, you must have the CREATE TRIGGER system privilege and one of the following must be true:

- You own the table associated with the trigger.
- You have the ALTER privilege on the table associated with the trigger.
- You have the ALTER ANY TABLE system privilege.

To create a trigger in another user's schema, you must have the CREATE ANY TRIGGER system privilege. To create such a trigger, you precede the trigger name in the CREATE TRIGGER statement with the name of the schema wherein it will reside, using the conventional dot notation.

Referring to Altered and Unaltered States

You can use the correlation variables OLD and NEW in the PL/ SQL block to refer to values in the table before and after the triggering statement had its effect. Simply precede the column names with these variables using the dot notation.

If these names are not suitable, you can define others using the REFERENCING clause of the CREATE TRIGGER statement, which is omitted from the syntax diagram above for the sake of simplicity. For more information on this clause, see CREATE TRIGGER in the Oracle7 Server SQL Reference.

Note: if a trigger raises an unhandled exception, its execution fails and the statement that triggered it is rolled back if necessary. This enables you to use triggers to define complex constraints. If the effects of the trigger have caused a change in the value of package body variables, however, this change is not reversed. You should try to design your

MAC003140

ORCL00941816

packages to spot this eventuality. For more information, see "Using Database Triggers" in the Oracle7 Server Application Developers Guide.

Enabling and Disabling Triggers

Just because a trigger exists does not mean it is in effect. If the trigger is disabled, it does not fire. By default, all triggers are enabled when created, but you can disable a trigger using the ALTER TRIGGER statement. To do this, the trigger must be in your schema, or you must have the ALTER ANY TRIGGER system privilege. Here is the syntax:

```
ALTER TRIGGER trigger_name DISABLE;
```

Later you can enable the trigger again by issuing the same statement with ENABLE in place of DISABLE. The ALTER TRIGGER statement does not alter the trigger in any other way. To do that you must replace the trigger with a new version using CREATE OR REPLACE TRIGGER. For more information on enabling triggers, see ALTER TRIGGER in the *Oracle7 Server SQL Reference*.

For more information on triggers generally, see "Using Database Triggers" in the Oracle7 Server Application Developer's Guide and CREATE TRIGGER and DROP TRIGGER in the Oracle7 Server SQL Reference.

MAC003141

ORCL00941817

MAC003142

ORCL00941818

CHAPTER

5 Oracle Web Agent

The Oracle Web Agent uses the PL/ SQL language to integrate the Oracle7 Server and the World Wide Web. It provides transparent access to Oracle7 directly from the World Wide Web (hereafter Web).

The Oracle Web Agent provides an easy–to–use environment for building applications in PL/ SQL. These applications are then stored in the Oracle7 database. A programmer who has chosen PL/ SQL as the language in which to implement a CGI application need only worry about implementing the logic specific to the application itself, because the Oracle Web Agent provides utilities which take care of the repetitive tasks associated with deploying a CGI application.

The HTML output generated by a PL/ SQL procedure is sent from the Oracle7 Server to the Oracle Web Agent. The output is then passed on to the Oracle Web Listener. The Web Listener sends the generated HTML document back to the Web client.

The HTML administrative forms that simplify the task of configuring the Oracle Web Agent are documented in Chapter 7, "Oracle WebServer Administration Utility."

MAC003143

ORCL00941819

Components of the Oracle Web Agent

The Oracle Web Agent is a CGI program that the Web Listener executes when a request is received for a dynamic document. This program logs into the database and executes stored PL/ SQL procedures that have been specified as part of the URL.

To facilitate the development of PL/ SQL for the Web Agent, a WebServer Developer's Toolkit is provided. This toolkit includes the following packages:

- HTP (hypertext procedures) and HTF (hypertext functions) ease the generation of HTML tags from within the user's PL/ SQL code. HTP and HTF automatically generate the HTML tags retrieved by the Oracle Web Agent.
- OWA_UTIL contains utility functions to enable the programmer to more rapidly build HTML output.

Note: The Web Agent also includes a PL/ SQL package called OWA that contains utility functions called by the Web Agent to do such things as set up the CGI environment variables and extract the output generated by the PL/ SQL code from a PL/ SQL table. A PL/ SQL developer does not make calls to the OWA package directly, although it can be called from a URL and, therefore, directly by a user..

For detailed information about the Oracle WebServer Developer's Toolkit, see Chapter 6. For information on the WebServer Administration Utility for administering the Web Agent, see Chapter 7, "Oracle WebServer Administration Utility."

Who Uses the Oracle Web Agent

The Oracle Web Agent can be used by:

- A programmer who wants to develop Web applications that interact with Oracle7. The Oracle Web Agent enables the programmer to create PL/ SQL procedures that perform database operations and format the results into HTML documents. These documents then get returned to the Web client.

  Therefore, the programmer needs to know the following:

  - the hypertext functions and hypertext procedures that can be used to generate an HTML–formatted document. These are described in Chapter 6, "The WebServer Developer's Toolkit."

MAC003144

ORCL00941820

- the basics of the PL/ SQL procedural language. These are contained in Chapter 4, "Introduction to the Oracle7 Server and to PL/ SQL".

In addition, the following is useful for the programmer:

- which Common Gateway Interface environment variables are available to the PL/ SQL application, and how to access them (This is covered in this chapter.).
- how the Oracle Web Agent is invoked (This is covered in this chapter.)
- how the Web Agent handles parameters (covered in this chapter.)
- how to use the OWA_UTIL PL/ SQL Utility Package (covered in Chapter 6, "The WebServer Developer's Toolkit".

- An administrator who wants to install or set up the Oracle Web Agent, and use the Administration Utility to maintain the Web Agent.

  The administrator needs to know the following:

  - how to create, modify, and delete Web Agent services
  - where to install the Developer's Toolkit PL/ SQL packages provided by the Web Agent

  The following information is useful for the administrator:

  - how the Oracle Web Agent works (covered in this chapter.)
  - how the Web Agent handles errors (covered in this chapter.)

How the Oracle Web Agent Works

The Oracle Web Agent enables you to create PL/ SQL stored procedures that can access data in Oracle7, dynamically create an HTML document from that data using the Developer's Toolkit, and return that document to the client.

This is a step by step description of how the Oracle Web Agent works:

1. The user submits a URL from the browser.

   The user submits the URL by one of the following methods:

   - by selecting a link from an HTML page
   - by specifying a URL in the Open URL box of their browser

MAC003145

ORCL00941821

– by submitting an HTML form

This action invokes a GET or POST request method operation on the associated URL. The request method is passed to the Web Listener using the HTTP protocol.

2. The Web Listener starts the Web Agent.

   The Web Listener determines that the URL is associated with the Oracle Web Agent (by extracting the /owa portion of the URL), and spawns the Web Agent. The Web Listener then makes several environment variables available to the Web Agent in compliance with the CGI specification.

3. The Web Agent connects to the Oracle7 Server.

   In order to connect to an Oracle7 Server, the Web Agent requires certain information, such as which server to connect to and what username and password to use. This information is stored in the owa.cfg file (located in the OWSAdministrative directory) as part of a Web Agent service. The Web Agent determines which service to use by parsing the SCRIPT_NAME environment variable, which is set up by the Web Listener in accordance with the CGI 1.1 specification. The Web Listener derives the value of this environment variable from the path section just before the /owa in the URL. For example, if /ows-bin/hr/owa is the leading portion of the URL, then hr is the service to use.

   For information on how to set up a Web Agent service, please see the section, "Web Agent Service", later in this chapter.

4. The PL/SQL procedure is invoked.

   After connecting to the database, the Web Agent invokes the appropriate PL/SQL procedure, whose name is obtained by parsing the PATH_INFO environment variable.

   If the PL/SQL procedure requires parameters, these would have been passed to the Web Agent by the Web Listener through the QUERY_STRING environment variable or on standard input depending on the REQUEST METHOD ("GET" or "POST"). The Web Agent would then pass these on to the PL/SQL procedure.

   For more information on PATH_INFO, SCRIPT_NAME, and QUERY_STRING and how they are used by the Oracle Web Agent, please see the section, "How the Web Agent Uses CGI Environment Variables," later in this chapter.

MAC003146

ORCL00941822

5. The PL/SQL procedure executes, generating an HTML document.

   With the help of the Developer's Toolkit, the PL/ SQL procedure extracts data from the Oracle7 database and generates an HTML document in a PL/ SQL table.

   For more information on the Developer's Toolkit, please see Chapter 6, "Oracle WebServer Developer's Toolkit."

6. The Web Agent passes the HTML document produced to the Oracle Web Listener.

   If the PL/ SQL procedure terminates without an error, the Web Agent takes the generated HTML document from the PL/ SQL table and sends it to standard output, which is where the Web Listener expects CGI applications to send their output.

   If the PL/ SQL procedure returns with an error, the Web Agent sends a programmer–defined error page to standard output.

   For more information on how errors are handled, please see the section, "Oracle Web Agent Error Handling".

7. The Web Listener sends the HTML document to the requesting browser.

   The Web Listener returns the HTML document to the browser using HTTP. The browser is unaware that the document requested was generated on–the–fly and presents it on the screen like any other HTML document.

Web Agent Service

Since it is desirable that a single Web Listener be able to access multiple databases or schemas within a database, the Oracle Web Agent supports the concept of services. For more information on databases and schemas, see Chapter 4 of this manual. The concept of services is best described by an example:

Company A would like data stored in its Oracle7 database to be accessible to the public via an Oracle Web Listener. However, different departments within Company A have access to different parts of the database (schemas), which are partitioned by way of user names. Thus, the Human Resources department and the Accounts Payable department use different username/ password combinations to log on to the database. If both want to write CGI applications using the Oracle Web Agent to access their data, the Web Agent needs to use the correct username/ password when it logs on to the database. In this scenario, the `owa.cfg` file, which contains the configuration information, will

MAC003147

ORCL00941823

contain two services, an HR service, and an AP service. Each service will have an associated username/ password, an ORACLE_HOME parameter, and an ORACLE_SID parameter. When a request from a Web browser comes in, the Web Listener will extract the service name that is embedded in the URL and find out which parameters to use by reading the owa.cfg file.

To connect to the Oracle7 Server, the Web Agent needs the following information to be specified in the Web Agent service:

- username
- password
- ORACLE_HOME
- ORACLE_SID (for local databases only)
- SQL*Net V2 Service Name or Connect String (for remote databases only)

The Administration Utility allows the administrator to display, create, modify or delete a Web Agent service. With the Web Agent Creation form, you do not need to modify the configuration file for the Oracle Web Agent service (owa.cfg) directly. See the section "Creating or Modifying Web Agent Services," in Chapter 7 for more information.

The following is an example of a Web Agent service entry in the owa.cfg file, and a corresponding explanation for each parameter:

Developer's T#

```
(
owa_service = es
(
owa_user = www_es
)
(
owa_password = tiger
)
(
oracle_home = /opt/oracle7
)
(
oracle_sid = esprod
)
(
owa_err_page = /es_err.html
)
```


MAC003148

ORCL00941824

```
(
owa_valid_ports = 8000 8888
)
(
owa_log_dir = /opt/oracle7/ows/log
)
(
owa_nls_lang = AMERICAN_AMERICA.US7ASCII
)
)
```

owa_service

Name of the Web Agent service

owa_user

Database username that the Web Agent will use to connect to the database.

owa_password

Database password that the Web Agent will use to connect to the database.

oracle_home

The location of the Oracle7 code tree in the file system. This should be the ORACLE_HOME for the database to which this Web Agent service connects, unless the Web Agent service is set up to connect to a remote database (over SQL*Net). In that case, specify the ORACLE_HOME where the Web Agent is installed. The Web Agent is always installed in an ORACLE_HOME directory or some subdirectory thereof.

oracle_sid

Name of the database system ID to connect to. Does not apply for connections to a remote database.

owa_err_page

This is the path of the HTML document that is returned by the Web Agent when an error occurs in the PL/SQL procedure that the Web Agent invoked. This is the actual path, not the virtual path as configured in the Web Listener.

owa_valid_ports

The valid Web Listener network ports the Web Agent will service.

owa_log_dir

MAC003149

ORCL00941825

The directory where the Oracle Web Agent writes its error file. The error file will have the name service_name.err.

owa_nls_lang

The NLS_LANG of the Oracle7 database to which the Web Agent connects. If not specified, the Web Agent administration program looks up the database NLS_LANG when the the service is submitted.

How the Web Agent Uses CGI Environment Variables

The Web Agent uses the environment variables shown in Table 5–1:

Variable	Contains
REQUEST_METHOD	GET or POST
PATH_INFO	the name of PL/ SQL procedure to invoke
SCRIPT_NAME	contains the service the Web Agent is to use when logging on to Oracle7
QUERY_STRING	parameters to the PL/ SQL procedure (for GET method only. POST method parameters are passed via standard input.)

Table 5–1 CGI Variables Used by the Oracle Web Agent

The following is an example of how a typical URL is parsed by the Web Listener in compliance with the CGI 1.1 specification.

http://www.nhl.com8080/ows-bin/nhl/owa/hockey.pass?person=Gretzky

1. The substring http://www.nhl.com8080 in the above URL signals the Web browser to connect to the www.nhl.com host's port 8080 using the HTTP protocol.
2. When the Oracle Web Listener that is running on www.nhl.com receives the request, the substring /ows-bin/nhl/owa signals the Web Listener to execute the Web Agent instead of returning a file to the browser as it normally would have done with a static HTML document.
   Please see the section on CGI (Common Gateway Interface) in Chapter 3, "The Oracle Web Listener," for more information on how the Web Listener recognizes the difference between a CGI program and a static HTML document.
3. After spawning the Web Agent, the Web Listener passes /ows-bin/nhl/owa to the Web Agent as the environment variable

MAC003150

ORCL00941826

SCRIPT_NAME. The rest of the URL is passed to the Web Agent in this manner: The substring /hockey_pass gets passed to the Web Agent in PATH_INFO and the substring "person=Gretzky" is passed to the Web Agent in QUERY_STRING.

4. The Web Agent parses the SCRIPT_NAME to extract the service name nhl .
   - Note that if the Web Agent cannot resolve the passed service name, it defaults to owa_default_service.
5. The Web Agent parses PATH_INFO to determine that hockey_pass is the PL/ SQL procedure to invoke after logging on to Oracle7.
6. The Web Agent makes the parameters passed to it in QUERY_STRING available to the PL/ SQL procedure it invokes.

   Note: The fact that the QUERY_STRING environment variable is used indicates that this is a GET request. You can also pass parameters using the POST method. For an example of how to pass parameters using the POST method, please see the next section, "Passing Parameters to PL/ SQL".

Passing parameters to PL/SQL

A PL/ SQL procedure often requires parameters in order to execute and generate the appropriate HTML document. The following section discusses several key concepts and tips that a PL/ SQL developer should understand with respect to how parameters get passed to the specified PL/ SQL routine.

These key concepts and tips are:

- Getting Parameters from the Web Browser to the Web Agent
- Passing Parameters Using an HTML Form
- Ordering Parameters
- Defaulting Parameter Values
- Using Multiple HTML Form Fields with the Same Name
- Overloading Procedures

Getting Parameters from the Web Browser to the Web Agent

Depending on the REQUEST_METHOD used, parameters are passed from the Web Browser to the Web Listener to the Web Agent in one of two ways:

MAC003151

ORCL00941827

- QUERY_STRING environment variable — If the GET method is used by the Web browser, the Web Listener passes the parameters to the Web Agent in this environment variable, as shown in the example in the preceding section.
- standard input — If the POST method is used, the Web Listener passes the parameters to the Web Agent using standard input.

The method used to pass parameters from the Web Listener to the Web Agent is transparent to the PL/ SQL procedure that is the actual consumer of the parameter(s). This is an important feature of the Oracle Web Agent: the PL/ SQL programmer need not be aware of which method is used and need not be concerned with parsing either the QUERY_STRING environment variable or standard input. Thus, the PL/ SQL programmer can concentrate on what he or she knows best: developing the logic to extract data from the Oracle database, based on pre–parsed parameters passed by the Oracle Web Agent.

It is recommended that you use POST whenever possible. GET is the method used for links and non–form URLs. For HTML forms, one has a choice. Because the GET method uses operating system environment variables, there are limits on the length of the QUERY_STRING.

Passing Parameters Using an HTML Form

The following example is analogous to the one in the previous section, except that it uses an HTML form that employs the POST REQUEST_METHOD.

```
<FORM METHOD="POST"
ACTION="http://www.nhl.com:8080/ows-bin/nhl/owa/hockey.pass">
Please type the name of the person you wish to search for:
<INPUT TYPE="text" NAME="person"><P>
To submit the query, press this button:
<INPUT TYPE="submit" VALUE="Submit Query">. <P>
</FORM>
```

The above form will cause the Oracle Web Listener to behave in the same fashion as the previous example, except that instead of populating the QUERY_STRING environment variable with "person=Gretzky," the Web Listener will write "person=Gretzky" to standard input. This assumes, of course, user typed "Gretzky" in the entry field of the HTML form above.

Note that the name of the HTML input variable, in this case "person", has to be the same as the PL/ SQL parameter it is to match.

The PL/ SQL procedure that is the recipient of the above parameters follows:

MAC003152

ORCL00941828

```
create or replace
procedure hockey_pass (person in varchar2) is
   n_assists integer;
begin
   select num_assists into n_assists
      from hockey_stats
   where name=person;
   htp.print(person||' has '||to_char(n_assists)||' assists this
   season');
end;
```

Ordering Parameters

Generally, the PL/ SQL developer does not need to be concerned with the order in which the Oracle Web Agent receives parameters from an HTML form or through a URL. The only case that it will be relevant is when passing multiple values for the same form field. See "Using Multiple HTML Form Fields with the Same Name" later in this section.

Defaulting Parameter Values

If the PL/ SQL developer cannot guarantee that a value will be passed from a Web Browser for a particular PL/ SQL procedure parameter, then it is recommended that he or she give the parameter a default value. For example:

```
create or replace procedure showvals(a in varchar2 DEFAULT NULL,
                                     b in varchar2 DEFAULT NULL)
is
begin
   htp.print('a = '||a||htp.br);
   htp.print('b = '||b||htp.br);
end;
```

If the Web Agent were to receive a request to call procedure showvals where there was no value for "a", the value for "b" was, say, "Hello", and the DEFAULT NULL clause was not part of the procedure's definition, then the request would generate an error with the following message:

```
OWS-05111: Agent : no procedure matches this call
  OWA SERVICE: test_service
  PROCEDURE: showvals
 PARAMETERS:
  ==========
  B:
   Hello
```


MAC003153

ORCL00941829

By "defaulting" the parameters, the above request would properly output:

a =

b = Hello

which to the end user would look like:

a =
b = Hello

Using Multiple HTML Form Fields with the Same Name

There are a number of instances where one would want to have multiple values passed for the same HTML form variable, and hence to the same PL/ SQL parameter. To handle this situation, one can use PL/ SQL tables to create an array of values.

One case where multiple values are passed with the same HTML form variable name is in the use of the HTML form tag "SELECT". If one sets the SIZE parameter to something greater than 1, then the user will be able to select multiple values for the same form field.

Another case where one has a set of values corresponding to a single form field follows:

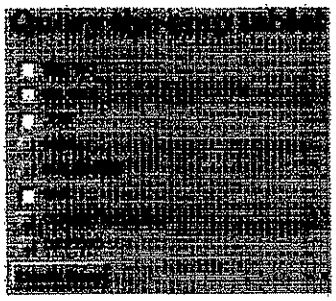
```
-- QUERY_FORM prints an HTML page with all the columns for the
-- specified table.  Invoke the procedure from a Web Browser with
-- a URL like:
http://yourhost:port_num/service_name/owa/query_form?the_table=emp

create or replace procedure query_form(the_table in varchar2) is
    cursor cols is
    select column_name
        from user_tab_columns
    where table_name = upper(the_table);
begin
    htp.htmlOpen;
    htp.headOpen;
    htp.htitle('Query the '||the_table||' table!');
    htp.headClose;
    htp.bodyOpen;
    -- Use owa_util.get_owa_service path to automatically retrieve
    htp.formOpen(owa_util.get_owa_service_path||'do_query');
    -- Put in the table as a hidden field to pass on to do_query
    htp.formHidden('the_table', the_table);
    -- Put in a dummy value, as we cannot DEFAULT NULL a PL/SQL
```


MAC003154

ORCL00941830

```
    table.
htp.formHidden('COLS', 'dummy');
    for crec in cols loop
        -- Create a checkbox for each column. The form field name
        -- will be COLS and the value will be the given column name.
        -- Will need to use a PL/SQL table to retrieve a set of
        -- values like this. Can use the owa_util.ident_arr type
        -- since the columns are identifiers.
        htp.formCheckbox('COLS', crec.column_name);
        htp.print(crec.column_name);
        htp.nl;
    end loop;
    -- Pass a NULL field name for the Submit field; that way, a
    -- name/value pair is not sent in. Wouldn't want to do this
    -- if there were multiple submit buttons.
    htp.formSubmit(NULL, 'Execute Query');
    htp.formClose;
    htp.bodyClose;
    htp.htmlClose;
end;
```

Invoking this procedure will bring up a page which looks like:



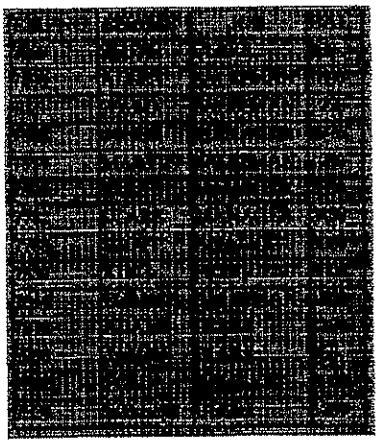
In this example, the user has already selected to query the EMPNO, ENAME, JOB, and SAL columns:

Here is a procedure to process this form submission:

```
-- DO_QUERY executes the query on the specified columns and
-- tables. The OWA_UTIL.IDENT_ARR datatype is defined as:
--   -- type ident_arr is table of varchar2(30) index by
binary_integer
-- create or replace procedure do_query(the_table in varchar2,
                                        cols in owa_util.ident_arr) is

    column_list varchar2(32000);
    col_counter integer;
```

```
    ignore        boolean;
begin
-- For PL/SQL tables, have to just loop through until you hit
-- no_data_found. Start the counter at 2 since we put in
-- a dummy hidden field.
    col_counter := 2;
    loop
        -- build a comma-delimited list of columns
        column_list := column_list||cols(col_counter)||',';
        col_counter := col_counter + 1;
    end loop;
exception
    when no_data_found
    then
         -- strip out the last trailing comma
         column_list := substr(column_list,1,length(column_list)-1);

          -- print the table - assumes HTML table support
         ignore := owa_util.tablePrint(the_table, 'BORDER',
                                              OWA_UTIL.HTML_TABLE,
 column_list);
end;
```

Then after selecting the "Execute Query" button, the user would see:



It is a good idea to use a hidden place-holder variable as the first value if you cannot guarantee that at least one value will be submitted for the PL/SQL table. The reason is that one cannot DEFAULT a PL/SQL table. And a call to this procedure with just one argument (the_table) would cause the Web Agent to generate an error.

MAC003156

ORCL00941832

Note that the Web Agent can only pass parameters to PL/ SQL tables which have a base type of VARCHAR2. This should not provide a significant limitation, as the PL/ SQL type VARCHAR2 is the largest PL/ SQL datatype with a maximum length of 32767 bytes. The values can then be explicitly converted to NUMBER, DATE, or LONG within a stored procedure (using TO_NUMBER or TO_DATE -- no conversion needed for LONGs).

Overloading Procedures

PL/ SQL allows developers to overload procedures and functions that are in PL/ SQL packages (but not standalone functions and procedures, For a discussion of overloading as implemented in PL/ SQL, see "Overloading Subprograms" in Chapter 4 of this manual). For example:

```
create or replace package overload is
      procedure proc1(charval  in varchar2);
      procedure proc1(numval   in number);
end;

create or replace package body overload is
      procedure proc1(charval  in varchar2) is
      begin
         htp.print('The character value is '||charval);
      end;

      procedure proc1(numval   in number);
         htp.print('The number value is '||numval);
      end;
end;
```

This functionality can be utilized by the Web Agent, but with the restriction that procedures that are overloaded on datatypes should not have the same parameter names. For example:

```
create or replace package overload is
      procedure proc1(val  in varchar2);
      procedure proc1(val  in number);
end;
```

When the Web Agent attempts to determine which procedure to call, it will not be able to distinguish between the two and will generate an error.

This limitation is imposed by the lack of HTML-form datatypes, but should not provide a significant limitation, as the PL/ SQL type VARCHAR2 is the largest PL/ SQL datatype with a maximum length of 32767 bytes. The values can then be explicitly converted to NUMBER,

MAC003157

ORCL00941833

DATE, or LONG within a stored procedure (using TO_NUMBER or TO_DATE – no conversion needed for LONGs).

Accessing CGI Environment Variables

The Oracle Web Listener conforms to the Common Gateway Interface (CGI) 1.1 specification. Thus, all CGI environment variables that are part of this specification are passed from the Oracle Web Listener to the Web Agent. The Web Agent, in turn, makes all relevant environment variables accessible from within PL/ SQL.

The Web Agent makes available the CGI environment variables shown in Table 5 – 2.

Variable	Variable Meaning
AUTH_TYPE	Method used to validate user
GATEWAY_INTERFACE	The revision of the CGI specification to which the server complies
HTTP_USER_AGENT	The browser the client is using to send the request
PATH_INFO	Extra path information given by the client
PATH_TRANSLATED	Translated version of PATH_INFO provided by server for mapping. This consists of the document root concatenated to the PATH_INFO.
REMOTE_HOST	Hostname making the request if it can be determined
REMOTE_ADDR	IP address of the remote host making the request
REMOTE_USER	Used to authenticate user
REMOTE_IDENT	Set to the remote username retrieved from the server
SERVER_PROTOCOL	Name and revision of the information protocol used in the request
SERVER_SOFTWARE	Name and version of information server software answering the request
SERVER_NAME	The server's hostname, or IP address

MAC003158

ORCL00941834

SERVER_PORT	Port number on which the server is running
SCRIPT_NAME	Virtual path to the script being executed, used for self–referencing URL

Table 5 – 2 CGI Environment Variables

These environment variables can be accessed from within PL/ SQL using the owa_util.get_cgi_env function, which is documented in Chapter 6," The Oracle WebServer Developer's Toolkit" .

Oracle Web Agent Error Handling

There are two types of errors that are handled by the Oracle Web Agent:

- application errors
- system errors

Application Errors

Application errors are specific to the PL/ SQL application. All applications written by a programmer should have their own exception handling in the PL/ SQL procedure that produces the appropriate output.

Because the Oracle Web Agent does not read the HTML output to determine its content, handled exceptions are transparent. As far as the Web Agent is concerned, if the PL/ SQL code generates HTML output, the operation was successful. The user will see whatever handled exception message is generated by the PL/ SQL procedure.

System Errors

System errors are detected by the Oracle Web Agent itself. These are errors that occur when the Web Agent is unable to launch the PL/ SQL procedure or when a PL/ SQL exception is not handled by the stored procedure, causing the exception to be propagated back to the Web Agent as a system error. This causes a standard HTML error document to be returned to the browser.

For example, if the Oracle Web Agent cannot make a connection to the Oracle7 Server, the PL/ SQL procedure cannot run and a system error occurs. The Web Agent then returns a default error message to the browser from the HTTP Server, or returns a customized HTML error

MAC003159

ORCL00941835

page (if one was previously configured as part of the Web Agent service using the OWA_ERR_PAGE parameter).

How the Web Agent Knows What Error Page to Generate

The Web Agent checks for the error page field in the OWA service configuration. If a value is found there, the Web Agent looks for the specified page in the file system. If found, that page is returned to the Web client. If the lookup fails, a default error message is returned to the Web client.

Creating Error Pages

The Oracle WebServer allows the administrator to create customized error pages that are returned whenever the Web Agent encounters a system error. The purpose of customized error pages is to tell the user what action to take.

Only one custom error page per service can be returned when the Web Agent encounters a system error.

To create a custom HTML error page, use any text or HTML editor you are comfortable with to write your HTML document.

The following is an example of a user-defined HTML error page for a system error:

```
<HTML>
<HEAD>
<TITLE>ERROR</TITLE>
</HEAD>
<BODY>
<HR>
<H1>Your Request Failed</H1>
<HR>
Bob's Big Bulldozer Company could not process your request. Please
try again later, or call us at 1-800-TRACTOR
Thanks, Bob
</BODY>
</HTML>
```

Once the customized HTML error page is created, the entry for OWA error page in the Web Agent Service configuration file must reflect where the new HTML error page is stored. Use the Web Service Administration page to insert or change the OWA error page entry.

A full path must be specified for the customized HTML error page. If a path is not specified, the Web Agent will search for it in the same directory in which the Web Agent executable is stored.

MAC003160

ORCL00941836

See the section, "Creating or Modifying the Web Agent Service," in Chapter 7, "Oracle WebServer Administration Utility," for more information.

Error Log File

The Oracle Web Agent keeps an error log file, defined by the OWA Service Name and the OWA Log Directory value for that service. The file will be named service_nameerr. An entry is appended to this file whenever the Oracle Web Agent encounters an error. The error log records all relevant information, such as date, error number, and URL.

Remember, there is only one error log for each Web Agent service. However, if the Web Agent cannot resolve a service name, and there is no OWA_DEFAULT_SERVICE configured, then errors will be written to OWA.err in the default log directory.

The following is an example of an error log file, and the corresponding meaning for each line in the file:

```
Wed Jun 28 08:14:27 1995                          /*Timestamp*/
OWS-05100: Agent : unable to connect due to Oracle error 1017
ORA-01017: invalid username/password; logon denied /*Error stack*/
OWA SERVICE: HR                                   /*Web Agent Service*/
PROCEDURE: hrinfo.emp                             /*PL/SQL procedure called*/
PARAMETERS:                                       /*parameters used by procedure*/
============
ENAME:
RayBorque
```

Oracle Web Agent Version Number

To find out what version of the Oracle Web Agent is installed on your operating system use the following procedure.

At the command line enter the following: (UNIX only)

$ORACLE_HOME/ows/bin/owa –v

The command will return the following (your result may vary):

```
Oracle Web Agent Release 1.0.0.0.0
```


MAC003161

ORCL00941837

MAC003162

ORCL00941838

CHAPTER

6

Oracle WebServer Developer's Toolkit

This chapter describes the hypertext procedures, hypertext functions, and utilities that make up the Oracle WebServer Developer's Toolkit.

One of the main goals of the Oracle Web Agent is to eliminate the PL/ SQL programmer's need to be intimately familiar with World Wide Web technology. To this end, the Oracle WebServer includes a Developer's Toolkit made up of several PL/ SQL packages that minimize the programmer's need to know HTML syntax. Although the programmer is still required to have a working knowledge of HTML, by using the Toolkit he or she will not need to hard code the exact syntax of HTML tags into PL/ SQL procedures. For instance, a programmer still needs to realize that an anchor tag is needed, but he or she doesn't need to know the exact sequence of characters needed to generate an anchor.

The Oracle WebServer Developer's Toolkit includes the following PL/ SQL Packages:

- Hypertext Procedures (HTP)

  A hypertext procedure generates a line in an HTML document that contains the HTML tag that corresponds to its name. For instance, the htp.anchor procedure generates an anchor tag. The HTP package will be the most commonly used package of the three.

MAC003163

ORCL00941839

- Hypertext Functions (HTF)

  A hypertext function returns the HTML tag that corresponds to its name. However, it is not sufficient to call an HTF function on its own because the HTML tag is not passed to the Web Agent. The output of an HTF function must be passed to htp.print in order to actually be part of an HTML document. Thus, the following line:

  ```
  htp.print(htf.italic('Title'));
  ```

  is functionally equivalent to:

  ```
  htp.italic('Title');
  ```

  Every hypertext function (HTF) has a corresponding hypertext procedure (HTP). Thus, HTF functions are generally used only when the programmer needs to nest calls, such as:

  ```
  htp.header(1, htf.italic('Title'));
  ```

  In this example, 'htf.italic' will generate the following character string:

  ```
  <I>Title</I>
  ```

  This string is then passed to the 'htp.header' procedure and the following line will appear in the HTML document being formatted:

  ```
  <H1><I>Title</I></H1>
  ```

- OWA Utilities (OWA_UTIL)

  A collection of useful utility procedures and functions. The purposes of these range from printing a signature tag on HTML pages to easy formatting of Oracle tables into HTML.

- OWA

  A set of procedures called only by the Oracle Web Agent itself. None of the subprograms in this package should be called directly from user-developed PL/SQL.

Installing the Oracle WebServer Developer's Toolkit

To install the Developer's Toolkit, use the Oracle Web Agent Service Administration forms. Both the Service Creation and Service Modification pages provide a checkbox for installing the Developer's Toolkit. Selecting this button and submitting the form will do the following:

MAC003164

ORCL00941840

- grant the CONNECT and RESOURCE roles to the OWA database user For more information on these roles or on granting roles, see "GRANT (roles)" in Chapter 4 of the Oracle7 Server SQL Reference.
- execute the OWAINS.SQL sql script, which can be found in the ows Administration directory. If run manually, the script should be run from SQL*DBA or from Server Manager. If you want to run it from SQL*Plus, see the header of the script for instructions.

  The OWAINS.SQL script installs all of the Developer's Toolkit packages.

Optimizing Multiple-Service Installations

If your site has multiple Web Agent services, you can minimize the amount of storage space used and enhance PL/ SQL performance by doing the following:

1. Install the Developer's Toolkit in one database user's schema. This user becomes the toolkit owner.
2. Drop the Developer's Toolkit PL/ SQL from the schemas of the OWA database users for other Web Agent services, if you have already installed them.

```
connect <user> / <password>
drop package HTF;
drop package HTP;
drop package OWA_UTIL;
drop package OWA;
```

3. Grant the system privilege EXECUTE on the PL/ SQL packages to OWA database users for other Web Agent services. For more information on this command, see "GRANT (system privileges)" in Chapter 4 of the Oracle7 Server SQL Reference.

```
connect <toolkit owner> / <password>
grant execute on HTF to <user>;
grant execute on HTP to <user>;
grant execute on OWA_UTIL to <user>;
grant execute on OWA to <user>;
```

4. Create synonyms for the Developer's Toolkit PL/ SQL packages in the schemas of all OWA database users who are specified in Web Agent services. Synonyms are alternate names that make it possible for the packages to be referred to without being qualified by schema names. For more information, see "Ownership and Naming Conventions" in Chapter 4 of this manual and see "CREATE SYNONYM" in Chapter 4 of the Oracle7 Server SQL Reference.

MAC003165

ORCL00941841

```
connect <user>/<password>
create synonym HTF for <Toolkit owner>.HTF;
create synonym HTP for <Toolkit owner>.HTP;
create synonym OWA_UTIL for <Toolkit owner>.OWA_UTIL;
create synonym OWA for <Toolkit owner>.OWA;
```

Security Note

PL/SQL procedures run with the privileges of the creator of the PL/SQL code. For the Developer's Toolkit, this is only an issue for the owa_util package. Two of the subprograms, show source and tableprint, access user data. Granting execute privileges on this package to users allows those users to view the tables, views, and stored PL/SQL code of the owner of owa_util.

If this is a security issue for your installation, install the owa_util package separately for each OWA database user.

The scripts to do this are:

- PUBUTIL.SQL
- PRIVUTIL.SQL

Both scripts reside in the OWSAdministration directory, and should be run in the order listed.

Procedure and Function Reference

This section describes each procedure and function in the htp, htf, and owa_util packages. Please note that for every htp procedure that generates HTML tags, a corresponding htf function exists with identical parameters. Note that defaulted parameters do not need to be passed.

The description of each procedure or function is broken down into the following parts:

Name	Name of procedure or function
Syntax	Actual syntax of procedure or function
Purpose	What the procedure or function does
Parameters	Parameters passed in
Generates	What the procedure or function generates

MAC003166

ORCL00941842

Parameters Passed into Procedures and Functions

All parameters passed into a hypertext procedure or function are of data type varchar2 (varying–length character string), integer, or date. The data type is indicated by a "c" for character, (varchar2) "n" for integer, or "d" for date as the first character of the parameter name. For example:

- cname in varchar2

  where cname is of data type character

- nsize in integer

  where nsize is of data type integer

- dbuf in date

  where dbuf is a date field

Note: Many HTML 3.0 tags have a large number of optional attributes that, if made as individual parameters to the hypertext procedures or functions would make the calls quite cumbersome. In addition, some browsers support non–standard attributes. Therefore, each hypertext procedure or function that generates an HTML tag has as its last parameter cattributes, an optional parameter. This parameter enables you to pass the exact text of the desired HTML attributes to the PL/ SQL procedure.

For example, the syntax for htp.em is:

```
htp.em(ctext, cattributes);
```

A call that uses HTML 3.0 attributes might look like the following:

```
htp.em('This is an example', 'ID="SGML_ID" LANG="en"');
```

This line would generate the following:

```
<EM ID="SGML_ID" LANG="en">This is an example</EM>
```

Print Procedures

The following print procedures are used in conjunction with htf functions to generate a line in the HTML document being constructed. They can also be passed hard–coded text that will appear in the HTML document as–is. The generated line is passed to the Web Agent which sends it to standard output. As documented in the CGI 1.1 specification, the Oracle Web Listener takes the contents of standard output and returns it to the Web browser that requested the dynamic HTML document.

MAC003167

ORCL00941843

htp.print

Syntax	htp.print (cbuf | dbuf | nbuf);
Purpose	generates a line in an HTML document.
Parameters	cbuf in varchar2 or dbuf in date or nbuf in number
Generates	Generates a line in an HTML document based on the value passed to it.

htp.p Alias for htp.print

htp.prn

Syntax	htp.prn (cbuf | dbuf | nbuf);
Purpose	Just like htp.print, but doesn't put a new line at the end of the value submitted.

htp.prints

Syntax	htp.prints (ctext);
Purpose	Generates a line in an HTML document and replaces all occurrences of the following special characters with escape characters. If not replaced, the special characters would be interpreted as HTML control characters, and would produce garbled output. '<' with '<' '>' with >' '"' with '"' '&' with '&'
Parameters	ctext in varchar2
Generates	Generates a line in an HTML document based on the value passed to it. This procedure is the same as htp.print or htp.p but first replaces the special characters listed above with escape characters.

htp.ps Alias for htp.prints

MAC003168

ORCL00941844

Structure Tags

The following tags are used to identify the major parts of an HTML document.

> *Note: Although this section shows hypertext procedures (HTP), all of them are also available as hypertext functions (HTF).*

htp.htmlOpen

Syntax	htp.htmlOpen;
Purpose	Prints a tag that indicates the beginning of an HTML document
Parameters	none
Generates	<HTML>

htp.htmlClose

Syntax	htp.htmlClose;
Purpose	Prints a tag that indicates the end of an HTML document
Parameters	none
Generates	</ HTML>

htp.headOpen

Syntax	htp.headOpen;
Purpose	Prints a tag that indicates the beginning of the HTML document head
Parameters	none
Generates	<HEAD>

htp.headClose

Syntax	htp.headClose;
Purpose	Prints a tag that indicates the end of the HTML document head
Parameters	none

MAC003169

ORCL00941845

Generates	</HEAD>

htp.bodyOpen

Syntax	htp.bodyOpen (cbackground, cattributes);
Purpose	Prints the tag that identifies the beginning of the body of an HTML document, and allows you to specify an image as the background of the document
Parameters	cbackground in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<BODY background="*cbackground*" *cattributes*> Note: If cbackground and cattributes are NULL, this tag generates <BODY>.
Example	htp.bodyOpen ('/img/background.gif'); This line produces: <BODY background="background.gif">

htp.bodyClose

Syntax	htp.bodyClose;
Purpose	Defines the end of the HTML document body
Parameters	none
Generates	</BODY>

Head Related Tags

The following procedure tags should be placed between the htp.headOpen and htp.headClose procedure tags.

> Note: Although this section shows hypertext procedures (HTP), all of them are also available as hypertext functions (HTF).

htp.title

Syntax	htp.title (ctitle);
Purpose	Prints an HTML tag with the text you pass in as the value of TITLE. Most Web Browsers display the text value enclosed between <TITLE> and </TITLE> at the top of the document viewing window.
Parameters	ctitle in varchar2

MAC003170

ORCL00941846

Generates <TITLE>*ctitle*</TITLE>

htp.htitle

Syntax htp.htitle (ctitle, nsize, calign, cnowrap, cclear, cattributes);

Purpose Prints the HTML tags for both the title and a top heading using the title you specify.

Parameters

ctitle	in varchar2	
nsize	in integer	DEFAULT 1
calign	in varchar2	DEFAULT NULL
cnowrap	in varchar2	DEFAULT NULL
cclear	in varchar2	DEFAULT NULL
cattributes	in varchar2	DEFAULT NULL

Generates <TITLE>*ctitle*</TITLE><H*nsize* ALIGN="*calign*" NOWRAP CLEAR="*cclear*" *cattributes*>*ctitle*</H*nsize*>

Note that htp.htitle produces non–standard HTML. However, most Web Browsers accept it. To produce standard HTML, the <TITLE> tag must be in the HEAD section of the page, and the level heading tag (<Hn>) must be in the BODY section. For example:

```
htp.headOpen;
htp.title(...);
htp.headClose;

htp.bodyOpen;
htp.header (1,...));
...
```

htp.base

Syntax htp.base;

Purpose Prints an HTML tag that records the URL of the document

Parameters none

Generates Inserts absolute pathname of current document.

MAC003171

ORCL00941847

htp.isindex

Syntax	htp.isindex (cprompt, curl);
Purpose	Creates a single entry field with a prompting text, such as "enter value," then sends that value to the URL of the page or program.
Parameters	cprompt in varchar2 DEFAULT NULL curl in varchar2 DEFAULT NULL
Generates	<ISINDEX PROMPT="*cprompt*" HREF="*curl*">

htp.linkRel

Syntax	htp.linkRel (crel, curl, ctitle);
Purpose	Prints the HTML tag that gives the relationship described by the hypertext link from the anchor to the target. This is only used when the HREF attribute is present. This tag indicates a relationship between documents, but does not create a link. To do that, use htp.anchor.
Parameters	crel in varchar2 curl in varchar2 ctitle in varchar2 DEFAULT NULL
Generates	<LINK REL="*crel*" HREF="*curl*" TITLE="*ctitle*">

htp.linkRev

Syntax	htp.linkRev (crev, curl, ctitle);
Purpose	Gives the relationship described by the hypertext link from the target to the anchor. This is the opposite of htp.linkRel. This tag indicates a relationship between documents, but does not create a link. To do that, use htp.anchor.
Parameters	crev in varchar2 curl in varchar2 ctitle in varchar DEFAULT NULL
Generates	<LINK REV="*crev*" HREF="*curl*" TITLE="*ctitle*">

htp.meta

Syntax	htp.meta (chttp_equiv, cname, ccontent);
Purpose	

MAC003172

ORCL00941848

Prints an HTML tag that identifies and embeds document meta-information that supplies the Web browser with information about the objects returned in HTTP.

Parameters	chttp_equiv	in varchar2	
	cname	in varchar2	
	ccontent	in varchar2	

Generates `<META HTTP-EQUIV="chttp_equiv" NAME="cname" CONTENT="ccontent">`

Example
```
http.meta ('Refresh', NULL, 120);
```
This line produces:
```
<META HTTP-EQUIV="Refresh" CONTENT=120>
```
which on some Web browsers will cause the current URL to be reloaded automatically every 120 seconds.

Body Tags

Body tags are used in the main text of your HTML page. They can format a paragraph, allow you to add hidden comments to your text, and add images within the body of your HTML text.

> Note: Although this section shows hypertext procedures (HTP), all of them are also available as hypertext functions (HTF).

htp.line

Syntax `htp.line (cclear, csrc, cattributes);`

Purpose Prints the HTML tag that generates a line in the HTML document. csrc enables you to specify a custom image as the source of the line.

Parameters	cclear	in varchar2	DEFAULT NULL
	csrc	in varchar2	DEFAULT NULL
	cattributes	in varchar2	DEFAULT NULL

Generates `<HR CLEAR="cclear" CSRC="csrc" cattributes>`

htp.hr

Alias for htp.line

htp.nl

Syntax `htp.nl (cclear, cattributes);`

MAC003173

ORCL00941849

Purpose	Prints the HTML tag that inserts a new line		
Parameters	cclear	in varchar2	DEFAULT NULL,
	cattributes	in varchar2	DEFAULT NULL,
Generates	<BR CLEAR="*cclear*" *cattributes*>		

htp.br

Alias for htp.nl

htp.header

Syntax	htp.header (nsize, cheader, calign, cclear, cattributes);		
Purpose	Prints the HTML tag for a heading level, with the value of the heading level assigned in the nsize parameter. Valid levels are 1 through 6.		
Parameters	nsize	in integer	
	cheader	in varchar2	
	calign	in varchar2	DEFAULT NULL
	cnowrap	in varchar2	DEFAULT NULL
	cclear	in varchar2	DEFAULT NULL
	cattributes	in varchar2	DEFAULT NULL
Generates	<H*nsize* ALIGN="*calign*" NOWRAP CLEAR="*cclear*" *cattributes*>*cheader*</H*nsize*>		
Example	htp.header (1, 'Overview'); produces <H1>Overview</H1>		

htp.anchor

Syntax	htp.anchor (curl, ctext, cname, cattributes);		
Purpose	Prints the HTML tag for an anchor to be the start or end destination of a hypertext link. This anchor can accept several attributes, but either HREF or NAME is required. HREF specifies where to link to. NAME allows this tag to be a target of a hypertext link.		
Parameters	curl	in varchar2	
	ctext	in varchar2	
	cname	in varchar2	DEFAULT NULL
	cattributes	in varchar2	DEFAULT NULL
Generates	<A HREF="curl" NAME="cname" *cattributes*>*ctext*</A>		

MAC003174

ORCL00941850

htp.mailto

Syntax	htp.mailto (caddress, ctext, cname, cattributes);
Purpose	Prints the HTML tag for an anchor with 'mailto' concatenated ahead of the mail address argument.
Parameters	caddresss in varchar2 ctext in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<A HREF="mailto:*caddress*" *cattributes*>*ctext*</A>
Example	htp.mailto('pres@white_house.gov','Send Email to the President'); prints <A HREF="mailto:pres@white_house.gov">Send Email to the President</A>

htp.img

Syntax	htp.img (curl, calign, calt, cismap, cattributes);
Purpose	Prints an HTML tag that signals the browser to load an image to be placed into the HTML page. ALT allows you to specify alternate text to be shown while the image is being loaded, or instead of the image if the browser does not support images. The ISMAP attribute indicates that the image is an image map.
Parameters	curl in varchar2 DEFAULT NULL calign in varchar2 DEFAULT NULL calt in varchar2 DEFAULT NULL cismap in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<IMG SRC="*curl*" ALIGN="*calign*" ALT="*calt*" ISMAP *cattributes*>

htp.para

Syntax	htp.para;
Purpose	Prints an HTML tag that indicates that the text previous to it should be formatted as a paragraph.
Parameters	none
Generates	<P>

MAC003175

htp.paragraph

Syntax	htp.paragraph (calign, cnowrap, cclear, cattributes);
Purpose	Prints the same HTML tag as htp.para except that parameters pass in exact alignment, leading, wrapping , and attributes.
Parameters	calign in varchar2 DEFAULT NULL cnowrap in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<P ALIGN="*calign*" NOWRAP CLEAR="*cclear*" *cattributes*>

htp.address

Syntax	htp.address (cvalue, cnowrap, cclear, cattributes);
Purpose	Prints an HTML tag that enables you to specify address, author and signature of document
Parameters	cvalue in varchar2 cnowrap in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<ADDRESS CLEAR="*cclear*" NOWRAP *cattributes*>*cvalue*</ADDRESS>

htp.comment

Syntax	htp.comment (ctext);
Purpose	Prints an HTML tag that allows you to store comments or lines in HTML pages. These comments are not visible to the end user.
Parameters	ctext in varchar2
Generates	<!— *ctext* —>

htp.preOpen

Syntax	htp.preOpen (cclear, cwidth, cattributes);
Purpose	Prints an HTML tag that indicates the beginning of preformatted text in the body of the HTML page.

MAC003176

ORCL00941852

Parameters	cclear	in varchar2 DEFAULT NULL
	cwidth	in varchar2 DEFAULT NULL
	cattributes	in varchar2 DEFAULT NULL

<PRE CLEAR="*cclear*" WIDTH="*cwidth*" *cattributes*>

htp.preClose

Syntax	htp.preClose;
Purpose	Prints an HTML tag that ends the preformatted section of text.
Parameters	none
Generates	</PRE>

htp.blockquoteOpen

Syntax	htp.blockquoteOpen (cnowrap, cclear, cattributes);	
Purpose	Prints an HTML tag that precedes a paragraph of quoted text.	
Parameters	cnowrap	in varchar2 DEFAULT NULL
	cclear	in varchar2 DEFAULT NULL
	cattributes	in varchar2 DEFAULT NULL;
Generates	<BLOCKQUOTE CLEAR="*cclear*" NOWRAP *cattributes*>	

htp.blockquoteClose

Syntax	htp.blockquoteClose;
Purpose	Ends the <BLOCKQUOTE> section of quoted text.
Parameters	none
Generates	</BLOCKQUOTE>

List Tags

List tags allow you to display information in any of the following ways:

- ordered: these lists have numbered items
- unordered: these lists have bullets to mark each item
- definition: these lists alternate a term with its definition

MAC003177

ORCL00941853

Note: All the hypertext procedures (HTP) shown in this section are also available as hypertext functions (HTF).

htp.listHeader

Syntax	htp.listHeader (ctext, cattributes);
Purpose	Prints an HTML tag at the beginning of the list
Parameters	ctext in varchar2 cattributes in varchar2 DEFAULT NULL
Generates	<LH *cattributes*>*ctext*</LH>

htp.listItem

Syntax	htp.listItem (ctext, cclear, cdingbat, csrc, cattributes);
Purpose	Prints an HTML tag that formats a listed item.
Parameters	ctext in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cdingbat in varchar2 DEFAULT NULL csrc in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<LI CLEAR="*cclear*" DINGBAT="*cdingbat*" SRC="*csrc*" *cattributes*>*ctext*

htp.ulistOpen

Syntax	htp.ulistOpen (cclear, cwrap, cdingbat, csrc, cattributes);
Purpose	Prints an HTML tag that is used to open an unordered list that presents listed items separated by white space and marked off by bullets.
Parameters	cclear in varchar2 DEFAULT NULL cwrap in varchar2 DEFAULT NULL cdingbat in varchar2 DEFAULT NULL csrc in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<UL CLEAR="*cclear*" WRAP="*cwrap*" DINGBAT="*cdingbat*" SRC="*csrc*" *cattributes*>

MAC003178

ORCL00941854

htp.ulistClose

Syntax	htp.ulistClose;
Purpose	Prints an HTML tag that ends the unordered list.
Parameters	none
Generates	</UL>

htp.olistOpen

Syntax	htp.olistOpen (cclear, cwrap, cattributes);
Purpose	Prints an HTML tag that is used to open an ordered list that presents listed items marked off with numbers.
Parameters	cclear in varchar2 DEFAULT NULL cwrap in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<OL CLEAR="*cclear*" WRAP="*cwrap*" *cattributes*>

htp.olistClose

Syntax	htp.olistClose;
Purpose	Prints an HTML tag that ends an ordered list.
Parameters	none
Generates	</OL>

htp.dlistOpen

Syntax	htp.dlistOpen (cclear, cattributes);
Purpose	Prints an HTML tag that starts a definition list
Parameters	cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<DL CLEAR="*cclear*" *cattributes*>

htp.dlistClose

Syntax	htp.dlistClose
Purpose	Prints an HTML tag that Ends a definition list
Parameters	none

MAC003179

ORCL00941855

	Generates	</DL>

htp.dlistDef

	Syntax	htp.dlistDef (ctext, cclear, cattributes);
	Purpose	Prints an HTML tag that is used to insert terms, and their corresponding definitions in an indented list format. The htp.dlistTerm must immediately follow this tag.
	Parameters	ctext in varchar2 DEFAULT NULL clear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
	Generates	<DD CLEAR="*cclear*" *cattributes*>*ctext*

htp.dlistTerm

	Syntax	htp.dlistTerm (ctext, cclear, cattributes);
	Purpose	Prints an HTML tag used to insert the definition term inside the definition list. This tag must immediately follow the htp.dlistDef.
	Parameters	ctext in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
	Generates	<DT CLEAR="*cclear*" *cattributes*>*ctext*

htp.menulistOpen

	Syntax	htp.menulistOpen;
	Purpose	Prints an HTML tag that begins a list that presents one line per item, and appears more compact than an unordered list. The htp.listItem will follow this tag.
	Parameters	none
	Generates	<MENU>

htp.menulistClose

	Syntax	htp.menulistClose;
	Purpose	Prints an HTML tag that ends a menu list.

MAC003180

ORCL00941856

Paramenters	none
Generates	</MENU>

htp.dirlistOpen

Syntax	htp.dirlistOpen;
Purpose	Prints an HTML tag that begins a directory list. This presents information in a list of items that contain up to 20 characters. Items in this list are typically arranged in columns, typically 24 characters wide. The <LI> or htp.listItem must appear directly after you use this tag.
Parameters	none
Generates	<DIR>

htp.dirlistClose

Syntax	htp.dirlistClose;
Purpose	Prints an HTML tag that closes the directory list tag, htp.dirlistOpen.
Parameters	none
Generates	</DIR>

Character Format Tags

The character format tags are used to specify or alter the appearance of the marked text. Character format tags have opening and closing elements, and affect only the text that they surround.

Character format tags give hints to the browser as to how a character or character string should appear, but each browser determines its actual appearance. Essentially, they place text into categories such that all text in a given category is given the same special treatment, but the browser determines what that treatment is. For example, the HTML string <STRONG>Here is some text</STRONG> might appear as bold in some browsers, or might flash instead.

If a specific text attribute, such as bold is desired, a physical format tag may be necessary. See the section, "Physical Format Tags," for more information.

MAC003181

ORCL00941857

Note: All the hypertext procedures (HTP) shown in this section are also available as hypertext functions (HTF).

htp.cite

Syntax	htp.cite (ctext, cattributes);		
Purpose	Prints a pair of HTML tags that specify the text they surround as a citation. Usually rendered as italics.		
Parameters	ctext	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
Generates	<CITE *cattributes*>*ctext*</CITE>		

htp.code

Syntax	htp.code (ctext, cattributes);		
Purpose	Prints a pair of HTML tags that specify the text they surround as an example of code output. Usually rendered in monospace format, e.g. Courier.		
Parameters	ctext	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
Generates	<CODE *cattributes*>*ctext*</CODE>		

htp.emphasis

Syntax	htp.emphasis (ctext, cattributes);		
Purpose	Prints a pair of HTML tags that specify the text they surround as requiring typographic emphasis. This tag is equivalent to htp.em. Usually rendered as italics.		
Parameters	ctext	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
Generates	<EM *cattributes*>*ctext*</EM>		

htp.em

Alias for htp.emphasis

htp.keyboard

Syntax	htp.keyboard (ctext, cattributes);
Purpose	Prints a pair of HTML tags that specify the text they surround as text typed in by the user, which is usually

MAC003182

ORCL00941858

rendered as monospace. This tag is equivalent to htp.kbd.

Parameters	ctext	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
Generates	<KBD *cattributes>ctext* </ KBD>		

htp.kbd

Alias for htp.keyboard

htp.sample

Syntax	htp.sample (ctext, cattributes);		
Purpose	Prints a pair of HTML tags that specify the text they surround as a sequence of literal characters that must be typed in the exact sequence in which they appear. Usually rendered as monospace font.		
Parameters	ctext	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
Generates	<SAMP *cattributes>ctext* </ SAMP>		

htp.strong

Syntax	htp.strong (ctext, cattributes);		
Purpose	Prints a pair of HTML tags that specify the text they surround as needing strong typographic emphasis. Usually rendered as bold.		
Parameters	ctext	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
Generates	<STRONG *cattributes>ctext* </ STRONG>		

htp.variable

Syntax	htp.variable (ctext, cattributes);		
Purpose	Prints a pair of HTML tags that specify the text they surround as a variable name, or a variable that might be entered by the user. Usually rendered as italics.		
Parameters	ctext	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
Generates	<VAR *cattributes>ctext* </ VAR>		

MAC003183

ORCL00941859

Physical Format Tags

The physical format tags are used to specify the format of the marked text.

Note: All the hypertext procedures (HTP) shown in this section are also available as hypertext functions (HTF).

htp.bold

Syntax	htp.bold (ctext, cattributes);		
Purpose	Prints a pair of HTML tags that specify the text they surround is to be rendered as boldface.		
Parameters	ctext	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
Generates	<B *cattributes*>*ctext*</B>		

htp.italic

Syntax	htp.italic (ctext, cattributes);		
Purpose	Prints a pair of HTML tags that specify the text they surround is to be rendered as italics.		
Parameters	ctext	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
Generates	<I *cattributes*>*ctext*</I>		

htp.teletype

Syntax	htp.teletype (ctext, cattributes);		
Purpose	Prints a pair of HTML tags that specify the text they surround is to be rendered in a fixed width typewriter font, e.g. Courier.		
Parameters	ctext	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
Generates	<TT *cattributes*>*ctext*</TT>		

MAC003184

ORCL00941860

Form Tags

The form tags are used to create and manipulate an HTML form. Forms are used to allow interactive data exchange between a Web Browser and a CGI program.

Forms can have the following types of elements:

- Input: used for a large variety of types of input fields, for example:
  - single line text
  - single line password fields
  - checkboxes
  - radio buttons
  - submit buttons
- Text area: used to create a multi–line input field.
- Select: used to allow the user to chose one or more of a set of alternatives described by textual labels. Usually rendered as a pulldown, pop up, or a fixed size list.

Note: All the hypertext procedures (HTP) shown in this section are also available as hypertext functions (HTF).

htp.formOpen

Syntax	htp.formOpen (curl, cmethod);	
Purpose	Prints an HTML tag that starts the form. The curl value is required and is the URL of the CGI script, normally owa, to which the contents of the Form will be sent. The method is either "GET" or "POST."	
Parameters	curl	in varchar2
	cmethod	in varchar2 DEFAULT 'POST'
Generates	<FORM ACTION=" *curl* " METHOD=" *cmethod* " >	

htp.formClose

Syntax	htp.formClose;
Purpose	Prints an HTML tag that closes the <FORM> tag
Parameters	none
Generates	</ FORM>

MAC003185

ORCL00941861

htp.formCheckbox

Syntax	htp.formCheckbox (cname, cvalue, cchecked, cattributes);
Purpose	Prints an HTML tag that inserts a checkbox which the user can toggle off or on.
Parameters	cname in varchar2 cvalue in varchar2 DEFAULT 'on' cchecked in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<INPUT TYPE="checkbox" NAME="*cname*" VALUE="*cvalue*" CHECKED *cattributes*>

htp.formHidden

Syntax	htp.formHidden (cname, cvalue, cattributes);
Purpose	Prints an HTML tag that sends the content of a field along with a submitted form. The field is not visible to the end user.
Parameters	cname in varchar2 cvalue in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<INPUT TYPE="hidden" NAME="*cname*" VALUE="*cvalue*" *cattributes*>

htp.formImage

Syntax	htp.formImage (cname, csrc, calign, cattributes);
Purpose	Prints an HTML tag that creates an image field that can be clicked on causing the Form to be immediately submitted. The coordinates of the selected point are measured in pixels, and returned (along with other contents of the form) in two name/ value pairs. The x–coordinate is submitted under the name of the field with ".x" appended, and the y–coordinate with the ".y" appended. Any value attribute is ignored. The image itself is specified by the CSRC attribute.
Parameters	cname in varchar2 csrc in varchar2 calign in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL;

MAC003186

ORCL00941862

	Generates	<INPUT TYPE="image" NAME="*cname*" SRC="*csrc*" ALIGN="*calign*" *cattributes*>

htp.formPassword

	Syntax	htp.formPassword (cname, csize, cmaxlength, cvalue, cattributes);
	Purpose	Prints an HTML tag that creates a single line text entry field. Text will not be displayed as it is entered. When the user enters a password, characters are represented by asterisks on single line text entry field.
	Parameters	cname in varchar2 csize in varchar2 cmaxlength in varchar2 DEFAULT NULL cvalue in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
	Generates	<INPUT TYPE="password" NAME="*cname*" SIZE="*csize*" MAXLENGTH="*cmaxlength*" VALUE="*cvalue*" *cattributes*>

htp.formRadio

	Syntax	htp.formRadio (cname, cvalue, cchecked, cattributes);
	Purpose	Prints an HTML tag that inserts a radio button on the HTML Form. Used to create a set of radio buttons, each representing a different value, only one of which will be toggled on by the user. Each radio button field should have the same name. Only the selected radio button will generate a name/ value pair in submitted data area. This will require an explicit VALUE attribute.
	Parameters	cname in varchar2 cvalue in varchar2 cchecked in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
	Generates	<INPUT TYPE="radio" NAME="*cname*" VALUE="*cvalue*" CHECKED *cattributes*>

htp.formReset

	Syntax	htp.formReset (cvalue, cattributes);

MAC003187

ORCL00941863

Purpose	Prints an HTML tag that creates a RESET button that, when selected, resets all the form fields to their initial values.	
Parameters	cvalue	in varchar2 DEFAULT 'Reset'
	cattributes	in varchar2 DEFAULT NULL
Generates	<INPUT TYPE="*reset*" VALUE="*cvalue*" *cattributes*>	

htp.formSubmit

Syntax	htp.formSubmit (cname, cvalue, cattributes);
Purpose	Prints an HTML tag that creates a button that, when selected, submits the form. If a SUBMIT button is selected to submit the Form, and that button has a name attribute specified, the submit button then contributes a name/ value pair to the submitted data.

Parameters	cname	in varchar2
	cvalue	in varchar2 DEFAULT 'Submit'
	cattributes	in varchar2 DEFAULT NULL
Generates	<INPUT TYPE="submit" NAME="*cname*" VALUE="*cvalue*" *cattributes*>	

htp.formText

Syntax	htp.formText (cname, csize, cmaxlength, cvalue, cattributes);
Purpose	Prints an HTML tag that creates a field for a single line of text.

Parameters	cname	in varchar2
	csize	in varchar2 DEFAULT NULL
	cmaxlength	in varchar2 DEFAULT NULL
	cvalue	in varchar2 DEFAULT NULL
	cattributes	in varchar2 DEFAULT NULL
Generates	<INPUT TYPE="text" NAME="*cname*" SIZE="*csize*" MAXLENGTH="*cmaxlength*" VALUE="*cvalue*" *cattributes*>	

htp.formSelectOpen

Syntax	htp.formSelectOpen (cname, cprompt, nsize, cattributes);

MAC003188

ORCL00941864

Purpose	Prints an HTML tag that begins a Select list of alternatives. Contains the attribute NAME which specifies the name that will be submitted as a name/ value pair.
Parameters	cname in varchar2 cprompt in varchar2 DEFAULT NULL nsize in integer DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	*cprompt* <SELECT NAME="cname" SIZE="nsize" *cattributes*>

Example	htp.formSelectOpen('greatest_player'; 'Pick the greatest player:');
	htp.formSelectOption('Messier');
	htp.formSelectOption('Howe');
	htp.formSelectOption('Hull');.
	htp.formSelectOption('Gretzky');.
	htp.formSelectClose;
	
	generates:
	
	Pick the greatest player:
	<SELECT NAME="great player">
	<OPTION>Messier
	<OPTION>Howe
	<OPTION>Hull
	<OPTION>Gretzky
	</SELECT>

Note: See htp.formSelectOption and htp.formSelectClose.

htp.formSelectOption

Syntax	htp.formSelectOption (cvalue, cselected, cattributes);
Purpose	Prints an HTML tag that represents one choice in the Select element.
Parameters	cvalue in varchar2 cselected in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<OPTION SELECTED *cattributes*>*cvalue* See example for htp.formSelectOpen.
Example	See htp.formSelectOpen.

MAC003189

ORCL00941865

htp.formSelectClose

Syntax	htp.formSelectClose;
Purpose	Prints an HTML tag that ends a Select list of alternatives.
Parameters	none
Generates	</ SELECT>
Example	See htp.formSelectOpen.

htp.formTextarea

Syntax	htp.formTextarea (cname, nrows, ncolumns, calign, cattributes);
Purpose	Prints an HTML tag that creates a text field that has no predefined text in the text area. Used to enable the user to enter several lines of text.
Parameters	cname in varchar2 nrows in integer ncolumns in integer calign in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<TEXTAREA NAME="*cname*" ROWS="*nrows*" COLS="*ncolumns*" ALIGN="*calign*" *cattributes*></ TEXTAREA>

htp.formTextareaOpen

Syntax	htp.formTextareaOpen (cname, nrows, ncolumns, calign, cattributes);
Purpose	Prints an HTML tag that opens a text area where you can insert predefined text that will always appear in the text field.
Parameters	cname in varchar2 nrows in integer ncolumns in integer calign in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<TEXTAREA NAME="*cname*" ROWS="*nrows*" COLS="*ncolumns*" ALIGN="*calign*" *cattributes*>

MAC003190

ORCL00941866

htp.formTextareaClose

Syntax	htp.formTextareaClose;
Purpose	Prints an HTML tag that ends TextArea field
Parameters	none
Generates	</TEXTAREA>

Table Tags

The Table tags allow the user to insert tables and manipulate the size and columns of the table in a document.

Note: All the hypertext procedures (HTP) shown in this section are also available as hypertext functions (HTF).

htp.tableOpen

Syntax	htp.tableOpen (cborder, calign, cnowrap, cclear, cattributes);
Purpose	Prints an HTML tag that begins an HTML table.
Parameters	cborder in varchar2 DEFAULT NULL calign in varchar2 DEFAULT NULL cnowrap in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL;
Generates	<TABLE BORDER NOWRAP ALIGN="*calign*" CLEAR="*cclear*" *cattributes*>

Note: The BORDER attribute is only determined by whether cborder is null or not null.

htp.tableClose

Syntax	htp.tableClose;
Purpose	Prints an HTML tag that ends an HTML table.
Parameters	none
Generates	</TABLE>

MAC003191

ORCL00941867

htp.tableCaption

Syntax	htp.tableCaption (ccaption, calign, cattributes);
Purpose	Prints an HTML tag that places a caption in the inserted table.
Parameters	ccaption in varchar2 DEFAULT NULL calign in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<CAPTION ALIGN="*calign*" *cattributes*>*ccaption*</CAPTION>

htp.tableRowOpen

Syntax	htp.tableRowOpen (calign, cvalign,cdp, cnowrap, cattributes);
Purpose	Prints an HTML tag that inserts a row tag into a table.
Parameters	calign in varchar2 DEFAULT NULL cvalign in varchar2 DEFAULT NULL cdp in varchar2 DEFAULT NULL cnowrap in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<TR ALIGN="*calign*" VALIGN="*cvalign*" DP="*cdp*" NOWRAP *cattributes*>

htp.tableRowClose

Syntax	htp.tableRowClose;
Purpose	Prints an HTML tag that ends a row in a table.
Parameters	none
Generates	</TR>

htp.tableHeader

Syntax	htp.tableHeader (cvalue, calign, cdp, cnowrap, crowspan, ccolspan, cattributes);
Purpose	Prints an HTML tag that inserts a table header.
Parameters	cvalue in varchar2 DEFAULT NULL calign in varchar2 DEFAULT NULL cdp in varchar2 DEFAULT NULL

MAC003192

ORCL00941868

cnowrap	in varchar2	DEFAULT NULL
crowspan	in varchar2	DEFAULT NULL
ccolspan	in varchar2	DEFAULT NULL
cattributes	in varchar2	DEFAULT NULL

Generates <TH ALIGN="*calign*" DP="*cdp*" ROWSPAN="*crowspan*" COLSPAN="*ccolspan*" NOWRAP *cattributes*>*cvalue*</TH>

htp.tableData

Syntax: htp.tableData (cvalue, calign, cdp, cnowrap, cattributes);

Purpose: Prints an HTML tag that inserts data into the rows and columns of a selected table.

Parameters:

cvalue	in varchar2	DEFAULT NULL
calign	in varchar2	DEFAULT NULL
cdp	in varchar2	DEFAULT NULL
cnowrap	in varchar2	DEFAULT NULL
cattributes	in varchar2	DEFAULT NULL

Generates <TD ALIGN="*calign*" DP="*cdp*" ROWSPAN="*crowspan*" COLSPAN="*ccolspan*" NOWRAP *cattributes*>*cvalue*</TD>

OWA_UTIL Package

The OWA_UTIL Package is a set of useful utility procedures built on top of hypertext functions and hypertext procedures.

owa_util.signature

Syntax: owa_util.signature;

Purpose: Prints an HTML line followed by a signature line on the HTML document.

Parameters: none

Generates: Prints a signature line in the HTML document that might look like the following:
"This page was produced by the Oracle Web Agent on August 9, 1995 09:30"

MAC003193

ORCL00941869

owa_util.signature (cname)

Syntax	owa_util.signature (cname);
Purpose	Allows the programmer to create a signature line on the bottom of the HTML document that has a hypertext link to view the PL/ SQL source for that procedure.
Parameters	cname in varchar2
Generates	Prints a signature line on the bottom of the HTML document that has a link to the actual PL/ SQL source for that procedure. The link calls the procedure showsource. The line would look like the following: "This page was produced by the Oracle Web Agent on 6/ 14/ 95 09:30" View PL/SQL Source

owa_util.showsource (cname)

Syntax	owa_util.showsource (cname);
Purpose	Prints the source of the specified PL/ SQL procedure, function, or package. If a procedure or function which belongs to a package is specified, then the entire package is displayed.
Parameters	cname in varchar2
Generates	Generates the source code of the specified PL/ SQL procedure.

owa_util.showpage

Syntax	owa_util.showpage;
Purpose	This procedure allows a user to view the HTML output of a PL/ SQL procedure call from SQL*Plus, SQL*DBA, or Oracle Server Manager. The PL/ SQL procedure must use HTP and/ or HTF to generate the HTML page, and owa_util.showpage must be issued after the PL/ SQL procedure has been called and before any other HTP or HTF subprograms are directly or indirectly called. This method is useful for generating pages filled with static data.

MAC003194

ORCL00941870

Note that this procedure uses dbms_output and thus is limited to 255 characters per line and an overall buffer size of 1,000,000 bytes.

Parameters	none
Generates	One can use this procedure to generate static pages in SQL*Plus that can then be accessed as a standard HTML page. For example:

```
SQL>set serveroutput on
SQL>spool gretzky.html
SQL>execute hockey.pass('Gretzky').
SQL>execute owa_util.showpage
SQL>exit
```

This would generate an HTML page which could be accessed from Web clients.

owa_util.get_cgi_env(function)

Syntax	owa_util.get_cgi_env(param_name in varchar2);
Purpose	Allows programmer to retrieve the value of the specified CGI environment variable in the PL/ SQL procedure.
	Note that param_name is case-insensitive.
Parameters	param_name in varchar2
Returns	Returns value of specified CGI environment variable for PL/ SQL procedure. If the value is not set, returns null

owa_util.print_cgi_env

Syntax	owa_util.print_cgi_env;
Purpose	enables programmer to print all of the CGI environment variables made available by the Web Agent to the PL/ SQL procedures. This utility is good for testing purposes.
Parameters	none
Generates	Prints CGI environment variables made available by the Web Agent to the PL/ SQL procedures.

MAC003195

ORCL00941871

owa_util.mime_header

Syntax	owa_util.mime_header;
Purpose	Enables programmer to change the default MIME header that the Web Agent returns. This must come before any htp.print or htp.prn calls in order to signal the Web Agent not to use the default.
Parameters	ccontent_type in varchar2
Generates	Content-type: <ccontent_type>\ n\ n

owa_util.get_owa_service_path (Function)

Syntax	owa_util.get_owa_service_path;
Purpose	Returns the name of the currently active path with its full virtual path, plus the currently active service. For example, a call to get_owa_service_path could return / ows-bi n/ myser vi ce/ owa/ .
Parameters	none
Returns	The service path. The datatype is varchar2.

owa_util.tableprint

Syntax	owa_util.tablePrint;		
Purpose	Enables programmers to print Oracle tables as either preformatted or HTML tables, depending upon Web browser capabilities. Note that RAW COLUMNS are supported, however LONG RAW are not. References to LONG RAW columns will print the result 'Not Printable'. In this case, cat t r i but es is the second, rather than the last, parameter.		
Parameters	ctable	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
	ntable_type	in integer	DEFAULT HTML_TABLE
	ccolumns	in varchar2	DEFAULT '*'
	cclauses	in varchar2	DEFAULT NULL
	ccol_aliases	in varchar2	DEFAULT NULL
	nrow_min	in number	DEFAULT 0
	nrow_max	in number	DEFAULT NULL

MAC003196

ORCL00941872

Note that ntable_type can be either owa_util.html_table or owa_util.pre_table.

Generates — Prints out either a preformatted or HTML table.

Returns — True or False as to whether there are more rows available beyond the nrow_max requested.

Example

For browsers that don't support HTML tables, create the following procedure:

```
create or replace procedure showemps is
  ignore_more boolean;
begin
  ignore_more := owa_util.tablePrint('emp', 'BORDER',
OWA_UTIL.PRE_TABLE);
end;
 and requesting a URL like this example:
http://myhost:8080/ows-bin/hr/owa/showemps

returns to the client:
<PRE>
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

```
</PRE>
```

To view just the employees in department 10, and only their employee ids, names, and salaries, create the following procedure:

```
create or replace procedure showemps_10 is
   ignore_more boolean;
begin
 ignore_more := owa_util.tablePrint
```


MAC003197

ORCL00941873

```
    ('EMP', 'BORDER', OWA_UTIL.PRE_TABLE,
        'empno, ename, sal',
        'where deptno=10 order by empno',
        'Employee Number, Name, Salary');
end;
```

A request for a URL like
http://myhost:8080/ows-bin/hr/owa/showemps_10 would return the following to the client:

```
<PRE>
-------------------------------------
| Employee Number |   Name  | Salary |
-------------------------------------
| 7782            | CLARK   | 2450   |
| 7839            | KING    | 5000   |
| 7934            | MILLER  | 1300   |
-------------------------------------
</PRE>
```

For browsers that do support HTML tables, to view the department table in an HTML table, create the following procedure:

```
create or replace procedure showdept is
 ignore_more boolean;
begin
    ignore_more := owa_util.tablePrint('dept', 'BORDER');
end;
```

A request for a URL like http://myhost:8080/ows-bin/hr/owa/showdept would return the following to the client:

```
<TABLE BORDER>
<TR>
<TH>DEPTNO</TH>
<TH>DNAME</TH>
<TH>LOC</TH>
</TR>
<TR>
<TD ALIGN="LEFT">10</TD>
<TD ALIGN="LEFT">ACCOUNTING</TD>
<TD ALIGN="LEFT">NEW YORK</TD>
</TR>
<TR>
<TD ALIGN="LEFT">20</TD>
<TD ALIGN="LEFT">RESEARCH</TD>
<TD ALIGN="LEFT">DALLAS</TD>
</TR>
<TR>
<TD ALIGN="LEFT">30</TD>
```


MAC003198

ORCL00941874

```
<TD ALIGN="LEFT">SALES</TD>
<TD ALIGN="LEFT">CHICAGO</TD>
</TR>
<TR>
<TD ALIGN="LEFT">40</TD>
<TD ALIGN="LEFT">OPERATIONS</TD>
<TD ALIGN="LEFT">BOSTON</TD>
</TR>
</TABLE>
```

which a Web browser can format to look like this:

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO

Customized Extensions to HTP Packages

The design of the hypertext procedure and function packages allows you to use customized extensions. Therefore, as the HTML standard changes, you can add new functionality similar to the hypertext procedure and function packages to reflect those changes.

Here is an example of customized packages using non–standard <BLINK> and imaginary <SHOUT>tags:

```
create package nsf as
    function blink(cbuf in varchar2) return varchar2;
    function shout(cbuf in varchar2) return varchar2;
end;
create package body nsf as
    function blink(cbuf in varchar2) return varchar2 is
    begin return ('<BLINK>' || cbuf || '</BLINK>');
    end;
    function shout(cbuf in varchar2) return varchar2 is
    begin return ('<SHOUT>' || cbuf || '</SHOUT>');
    end;
end;

create package nsp as
    procedure blink(cbuf  in varchar2);
    procedure shout(cbuf  in varchar2);
end;
create package body nsp as
    procedure blink(cbuf  in varchar2) is
```


MAC003199

ORCL00941875

```
    begin htp.print(nsf.blink(cbuf)); end;
    procedure shout(cbuf  in varchar2) is
    begin htp.print(nsf.shout(cbuf)); end;
end;
```

Now you can begin to use these procedures and functions in your own procedure.

```
create procedure nonstandard as
begin
    nsp.blink('Gee this hurts my eyes!');
    htp.print('And I might ' || nsf.shout('get mad!'));
end;
```

For more examples of using the Developer's Toolkit, see the section, "Passing Parameters to PL/SQL" in Chapter 5, "Oracle Web Agent".

MAC003200

ORCL00941876

CHAPTER

7

The Oracle WebServer Administration Utility

This chapter shows you how to use the Administration Utility of the Oracle WebServer. The Administration Utility enables you to configure the components of the Oracle WebServer from any web browser, so you do not have to manually edit files to perform common operations.

To navigate to the Administration Utility, click Administration from the Oracle WebServer home page. The utility is divided into the following three areas:

- The Oracle7 Database Server. This enables you to start up or shut down an Oracle7 database.
- The Oracle Web Listener. This enables you to create an instance of the Oracle Web Listener or to modify the parameters of an existing instance.
- The Oracle Web Agent. This enables you to create or modify web agent services, including the specification of password-protected security.

The Administration page is shown in Figure 7 – 1. Note: If you are viewing this document online, there are no screen shots provided.

MAC003201

ORCL00941877

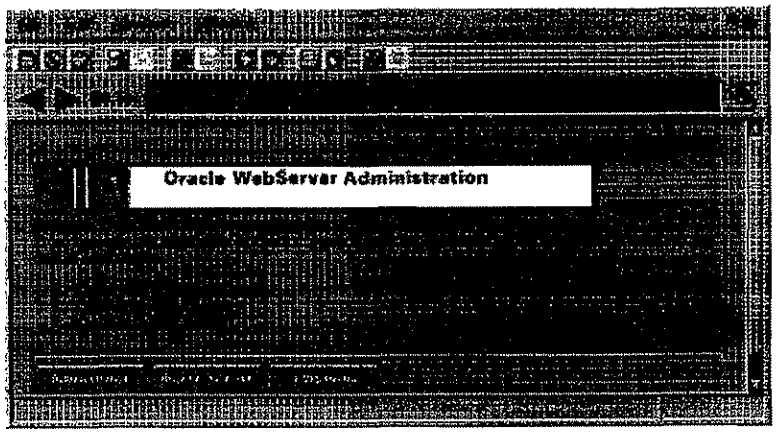


Figure 7–1 Administration Utility Form

Choose the area you want to use by clicking the appropriate button. This chapter contains a section for each administration area.

Database Administration

Use this page, shown in Figure 7–2, to start up or shut down any Oracle7 database running on the same machine as the Oracle Web Listener.

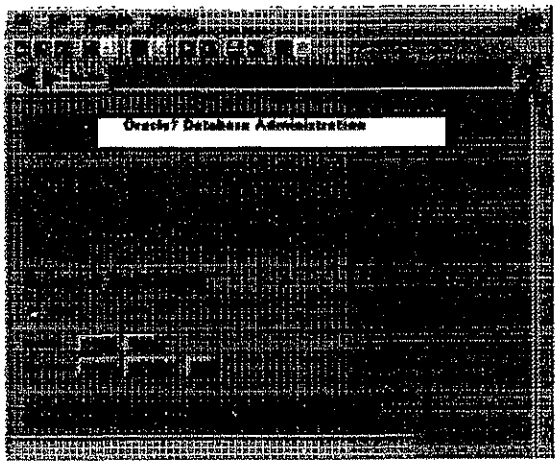


Figure 7–2 Database Administration Page

MAC003202

ORCL00941878

The page lists the available databases. This list comes from a file on the machine on which the Web Listener resides. On UNIX machines, this is known as the ORATAB file. See your platform-specific documentation for the location and format of this file. If no databases are listed, then the file contains no database descriptions.

The procedure is as follows:

1. Choose the database to which you want to apply the operation by clicking the radio button beside its name in the list.
2. Choose the operation you wish to perform and the mode by clicking the appropriate button. The modes are explained in the following sections.

For more information on database startup and shutdown, see the Oracle7 Server Administrator's Guide.

Startup Modes

You can perform database startup in either of the following two modes:

Normal

Starts up the database normally. This assumes that the database currently is shut down. If there are any errors during startup, they are reported, and the database is left shut down.

Force

Shuts down the database in the Abort mode (as explained under Shutdown Modes below), and then starts it up in the Normal mode.

Shutdown Modes

You can perform database shutdown in any of the following three modes:

Normal

Shuts down the database cleanly, waiting for current session to disconned normally. New connections are not allowed while shutdown is in progress.

MAC003203

ORCL00941879

Immediate

Attempts to shut the database down immediately and cleanly. Disconnect any existing user sessions, but wait for active calls to complete before disconnecting. Roll back all uncommitted transactions .

Abort

Shuts the database down as quickly as possible, killing all active Oracle7 processes and freeing the memory used. This leaves current transactions unresolved. When you start up the database again, instance recovery is performed automatically. This option is recommended only for emergency use.

For more information on Oracle7 databases including startup and shutdown, see the Oracle7 Server Concepts Manual and the Oracle7 Server Administrator's Guide.

Web Agent Administration

The Web Agent Administration page enables you to create, delete, or modify Web Agent Services, which Web Agents use to connect to the database. An Oracle7 Web Agent Service specifies values needed by the Web Agent every time it makes a database connection. For example, an Oracle7 username and password are needed, as well as an Oracle7 SID or SQL*Net V2 service name. For more information on Web Agent Services, see "Web Agent Service" in Chapter 5.

Creating or Modifying Web Agent Services

Use the Web Agent section of the Administration Utility to create a new Web Agent Service or to copy, delete, or modify an existing one. The initial Web Agent Administratin page is shown in Figure 7 – 3.

MAC003204

ORCL00941880



Figure 7–3 Initial Web Agent Administration Page

To create a new Oracle Web Agent Service, click Create New Service. The Oracle Web Agent Administration Creation form appears.

To modify, copy, or delete an existing Oracle Web Agent Service, click the name of that service in the list provided. The Oracle Web Agent Administration Modification form appears.

The Oracle Web Agent Administration Service Creation Form

Use this fill–out form, shown in Figure 7–4 to enter the parameters for the service you want to create. The parameters are explained below.

MAC003205

ORCL00941881

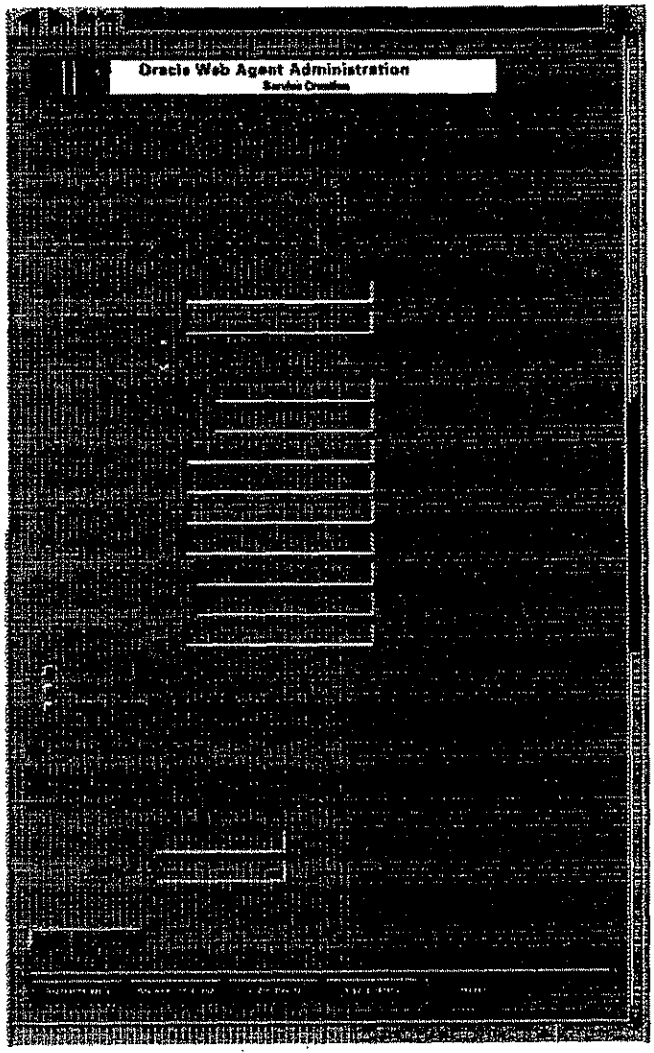


Figure 7 – 4 Create Web Agent Service Form

OWA Service:

The name of the service you want to create. The name is not case sensitive and must be unique.

MAC003206

ORCL00941882

OWA Database User:

The name of the Oracle7 user that will be used to access the database for this service. If operating system (OS) authentication is used, then the Web Listener must be run with the same effective user as the Oracle7 user. For more information on Oracle7 users, see "CREATE USER" in the Oracle7 Server SQL Reference.

Identified By

A radio button that determines whether Oracle7 or the Operating System performs authentication.

OWA User Password:

The Oracle7 password for the associated OWA Database User. If OS authentication is be used for the OWA Database User, as determined by the Identified By radio button, you can leave this field blank. Otherwise, the username and password for the OWA Database User are verified by Oracle7 upon submission of a Web Agent service.

If an error occurs when you attempt to submit this form, the password fields are blanked out and must be filled in again. This is to provide the tightest security currently available for your passwords. As a short–cut, you can select the Back button on your Web Browser to return to, and correct, the original submission form.

Confirm Password:

This field is used to verify the OWA user password you entered above before recording it.

ORACLE_HOME:

The location of the Oracle7 code tree in the file system. This should be the ORACLE_HOME for the database to which this Web Agent service connects, unless the Web Agent service is set up to connect to a remote database (over SQL*Net). In that case, specify the ORACLE_HOME where the Web Agent is installed.

ORACLE_SID:

The system ID of the database. You only need this if the Web Agent is to connect to a local database. For more on Oracle7 SIDs, see the Oracle7 Server Administrator's Guide.

MAC003207

ORCL00941883

SQL*Net V2 Service:

The SQL*Net service name that indicates which database to connect to. If the database is on the same machine as the Web Listener, then the SQL*Net V2 Service field can be left empty, and an Oracle7 SID specified instead. If both parameters are specified, the SQL*Net V2 Service takes precedence. If SQL*Net V1 is installed on this machine, then a SQL*Net V1 Connect String may also be specified here. For more on the SQL*Net V2 service name, see Understanding SQL*Net.

Authorized Ports:

To provide security for your Web Agent service, use the Authorized Ports field to list (space-delimited) the valid Web Listener network ports for the Web Agent to service. This field is required, and the network ports must be listed explicitly.

Log File Directory:

The directory where the Oracle Web Agent writes its error file. The error file will have the name service_name.err.

If the Web Agent is unable to resolve a service name, and no service called OWA_DEFAULT_SERVICE exists, then it will write error and logging information to the file owa.err.

The default value for the Log File Directory is the log directory under the OWS directory within ORACLE_HOME.

Note that these log files are different from those used by the Oracle7 Server for database recovery. The latter are explained in the Oracle7 Server Concepts Manual.

HTML Error Page :

If the Oracle Web Agent is unable to process a request, it returns this HTML page to the user making the request, and at the same time writes detailed errors and debugging information to the error message file in the Log File Directory. If this field is left blank, a default error message is returned to the client.

NLS Language :

This field specifies the NLS_LANG parameter, which specifies the character set to be used. If this field is left blank, then the Web Agent administration program looks it up in the configuration file for the

MAC003208

ORCL00941884

database upon service submission.

Create OWA Database User :

Create an Oracle7 user using the OWA user name and OWA user password provided. If the Identified by ... Operating System radio button is selected, then the user is created as identified externally, which means that the operating system, rather than Oracle, performs user authentication. In that case, this value is ignored.

Change OWA Database User Password:

Change the Oracle7 Database user's existing password to that specified by the OWA User Password. If the Identified by ... Operating System radio button is selected, then the user will be altered to be identified externally, which means that the operating system, rather than Oracle, performs user authentication.

Install Oracle WebServer Developer's Toolkit PL/SQL Packages

Create the Oracle WebServer Developer's Toolkit PL/ SQL packages—OWA, HTP, HTF, and OWA_UTIL—under the schema of the OWA Database User.

After filling in the above fields, click the Submit New Service button to create the service. To activate your new service, go to the Oracle WebServer Administration form and click Configure for the Web Listener you want to associate with this agent. Then go to "Directory Mappings" to create a virtual path, with the service name as the last part of the path, to the OWS/ bin directory. Virtual paths are discussed under "Directory Mappings", later in this chapter.

The Oracle Web Agent Administration Service Modification Form

This form contains the same fields as the Oracle Web Agent Administration Creation Form. The fields show their current values except for the password fields, which are left blank for security reasons. You can leave these fields blank unless you want to change the password.

MAC003209

ORCL00941885

At the bottom of the form are the following three buttons:

Copy Service

This creates a copy of the service, leaving the original intact. To do this, simply change the service name (the OWA service field) and any other parameters you wish to alter for the new copy and click this button. If you specify no new password, the original password is copied. After clicking this button, you have two services.

Modify Service

This changes the existing service. You can change the name or leave it intact as desired. Make the desired changes and click the button. After clicking this button, you have one service.

Delete Service

This destroys the service. For this purpose, it does not matter whether you change any parameters. After clicking this button, you have no service. Calls to the deleted service in existing Web pages or PL/ SQL blocks do not produce errors, however. Instead, they are reinterpreted as calls to OWA_DEFAULT_SERVICE.

Securing a Web Agent Service

The mechanism of "services" provides a security framework such that individual Web Agent services can be password protected by the Web Listener. For example, to password protect the hr service, you would do the following:

1. Create a virtual path for the hr service by specifying a Web Listener directory mapping. This mapping must use a value of 'C' to indicate an executable. For more information on setting the directory mappings, see Directory Mappings later in this chapter.
2. Create a realm consisting of those users and groups (UNIX only) who are to have access to the service. For instruction on how to do this, see Specifying the Privileged Users, later in this chapter.
3. Assign the realm to the service and specify either Basic or Digest Authentication. To learn how to perform the assignment, see Performing the Security Assignments, later in this chapter. To find out about Basic and Digest Authentication, see Security Schemes, later in this chapter.

MAC003210

ORCL00941886

For more information of Web Listener security in general, see "Security Definitions", later in this chapter, and "Client to Listener Security" in Chapter 3.

Web Listener Administration

The Oracle Web Listener is a reliable, high–performance HTTP server suitable for high–traffic web applications where quick response is crucial.

From the Web Listener Administration page, shown in Figure 7 – 5, you can move to other pages that provide a convenient way for you to do the following:

- Create a new Web Listener
- Change or copy an existing Web Listener
- Start a Web Listener
- Stop a Web Listener
- See whether an Oracle Web Listener is currently running.

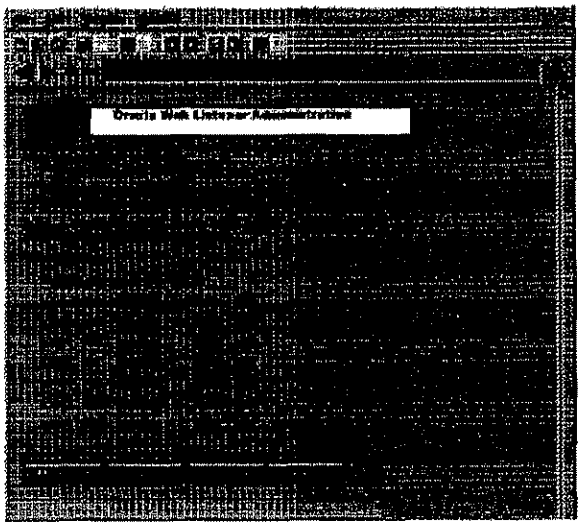


Figure 7 – 5 Web Listener Administration Page

To create a new Oracle Web Listener, click Create New Oracle Web Listener. This takes you to the Listener Creation form.

MAC003211

ORCL00941887

To perform any of the other operations, choose the appropriate existing Web Listener from the list provided. This list uniquely identifies each Web Listener by the network port on which it is configured to listen. For your convenience, the directory in which the configuration file resides is specified beside the port number. Unlike the port number, this directory need not be unique.

If a Web Listener is running, the word "running" appears beside it in the list.

To modify an existing Oracle Web Listener, select it in the list and click Configure. This takes you to the Configure Extended Parameters form, with the current parameter values filled in. Effect the changes by writing over the current values. For information on the parameters themselves, see "Configure Extended Parameters," later in this chapter.

To start a currently–stopped Oracle Web Listener, select it in the list and click Start.

To stop a currently–running Oracle Web Listener, select it in the list and click Stop.

> Note: You might need to perform a manual startup of an existing Web Listener if you inadvertently shut down the Web Listener that the Administration Utility itself is using. To find out how to do this, see "The WLCTL" in Chapter 3.

The Oracle Web Listener Creation Form

Use this form, shown in the next two figures, to create a new Oracle Web Listener by filling in the listed parameters. These are the parameters the Web Listener requires in order to run. After filling in all required parameters, press the appropriate button to choose a Basic Configuration or an Advanced Configuration. In either case, the Web Listener is created. If you choose Basic Configuration, the parameters you fill in here are all that are needed, and you can start the Web Listener you have created. If you choose Advanced Configuration, you go to the Configure Extended Parameters form to provide optional information. This is equivalent to selecting an existing Web Listener from the list in the initial Web Listener Administration page and clicking Configure. For more information on the Configure Extended Parameters form, see "The Configure Extended Parameters Form" later in this chapter.

MAC003212

ORCL00941888

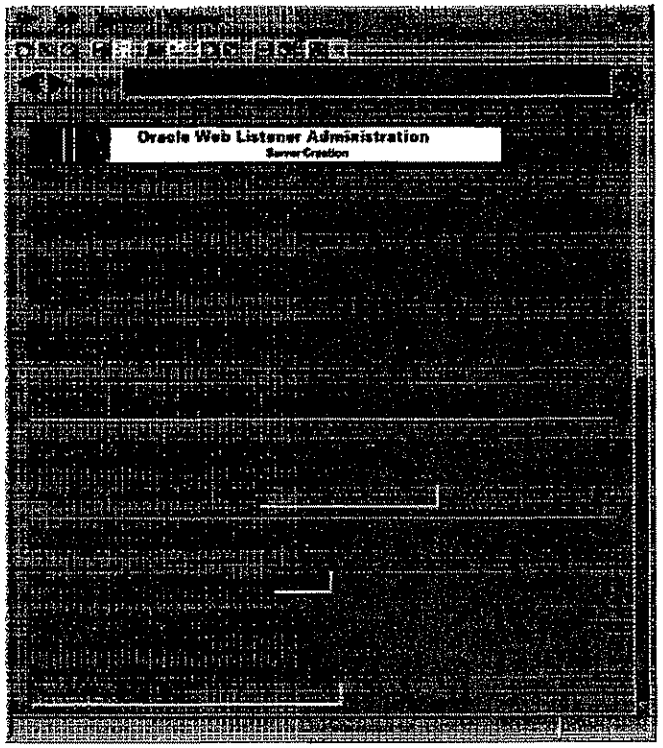


Figure 7 – 6 Listener Creation Form, Part 1

The Basic Configuration Parameters

The parameters required for a basic configuration are explained below:

Host Name

This is the name of the machine on which the Oracle Web Listener is to reside. It can either be an actual machine name or an alias in the form of a domain name, such as:

www. or acl e. com

Port Number

The Oracle Web Listener listens for network messages on a particular port. Use this parameter to specify that port. Numbers can range from 0 to 65535 with numbers below 1024 requiring superuser privileges on UNIX systems.

MAC003213

ORCL00941889

Configuration Directory

This is the directory where the configuration file for the Web Listener is to be located. A default is provided.

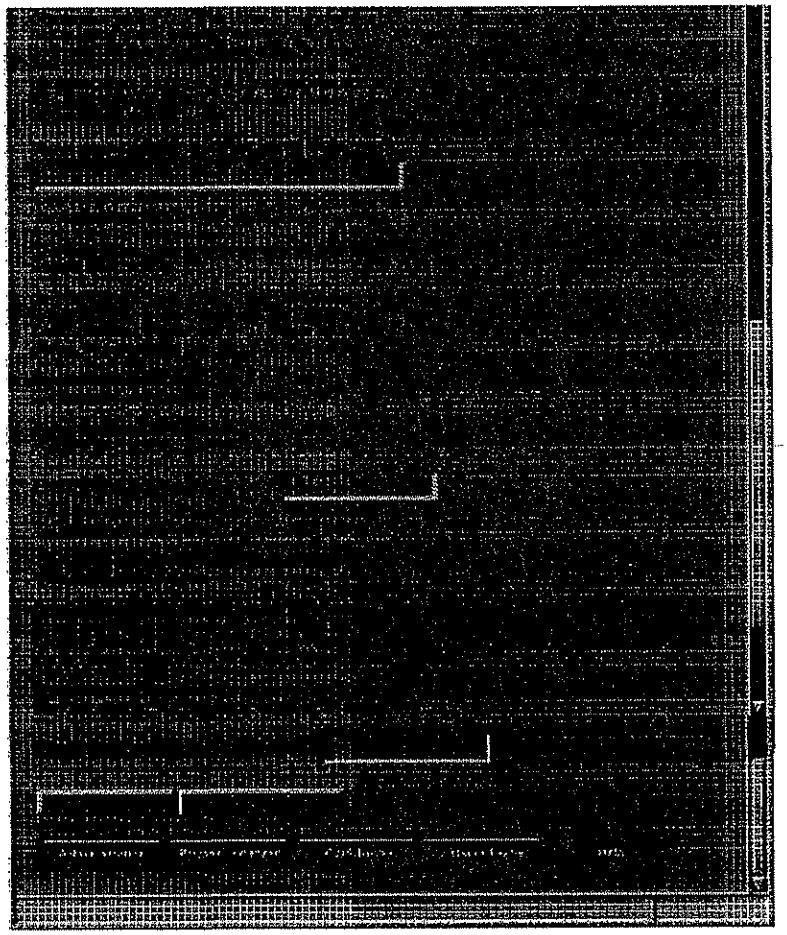


Figure 7 – 7 Listener Creation Form, Part 2

Web Listener Root (/) Directory

This is the directory that serves as the root (/) directory for the Web Listener. This is often called the "Document Root". It is used to resolve client calls that do not specify a directory. Thus a URL that does not contain a directory, such as http:/ / www.oracle.com/ index.html,

MAC003214

ORCL00941890

causes the Web Listener to retrieve the document "index.html" from the directory specified here.

User ID (UNIX only)

The User ID specifies the user (or user identification number) under whose privileges the Web Listener executes. To acquire these privileges, the Web Listener must be owned by this user or by root with "Set user ID" set. The User ID can be either a username or a user ID number. The default is the effective ID. Specifying the Oracle owner or an Oracle user with the DBA role is not advised, as this compromises database security. For more information, see "roles" and "Database Administrators" in the Oracle7 Server Concepts Manual.

Group ID (UNIX only)

The Group ID specifies the group (or group identification number) under whose privileges the Web Listener executes. To acquire these privileges, the Web Listener must be owned by this group or by root with "Set group ID" set. The Group ID can be either a group name or a GID number. The default is the effective GID.

Note that any Oracle Web Listener that invokes the Oracle Web Agent must have read access to the OWA configuration file (owa.cfg). However, since this file contains Oracle usernames and their corresponding passwords, it is recommended that read permissions only be given to the oracle owner and to one privileged group. This is the group that a Web Listener should run under if it is going to run the Oracle Web Agent.

A convenient group to use for this is the OracleDBA group. However, if you run an Oracle Web Listener under the DBA group, you will want to place strict controls on who can create CGI programs that will be invoked by that Web Listener. This is because the CGI program will run with group DBA privileges, and someone with malicious intent could potentially startup and shutdown your Oracle database, or even remove Oracle files. If you wish to avoid this potential security risk, then you need to do the following:

1. Create a new operating system group.
2. Make the Oracle owner a member of the group created in step 1.
3. Log in as the Oracle owner and change the group ownership of owa.cfg to the group created in step 1.

   Note: If at any future point in time you remove the owa.cfg file, you must reset its group ownership when it is recreated.

4. Configure the Group ID parameter of your Oracle Web Listener to be the name of the group created in step 1.
5. Restart your Administration Server (the Web Listener that the Administration Utility is using). You must do this to ensure that the Web Agent Administration program inherits the group created in step 1. Do not change the Group ID for the administration server. It must continue to run as the OracleDBA group.

Starting a New Oracle Web Listener

Once you have create a Web Listener, you are shown a Success! message. At this point, you are ready to start your new Oracle Web Listener. Ordinarily, you do this by navigating to the Oracle Web Listener Administration page and clicking start. Note that most web browsers buffer visited pages, and therefore do not reload them when you return to them using something like a Back button. For the new Web Listener to be listed as available for startup, you must navigate to it in such a way that the page is reloaded, for example by reentering the URL.

Reinitializing the Oracle Web Listener

If you selected Advanced Configuration and all the parameters you entered were valid, you will be shown a Success! message. You will then be presented with the Web Listener Configure Extended Parameters form which allows you to modify any of the Oracle Web Listener configuration parameters. It is important to remember that once the Web Listener is started, however, it does not automatically recognize new files added to its virtual file system or parameters changed in its configuration file. If you would like to add files to the Web Listener, you should navigate to the Web Listener Administration page and click Configure next to the Web Listener you want to modify.

The Configure Extended Parameters Form

This form provides a convenient way to customize an existing Oracle Web Listener for your particular environment. By using this form, you can avoid having to manually edit the Web Listener configuration file, thus eliminating the need to be familiar with the exact syntax for each parameter.

From this form, you will be able to customize the following:

- Network Parameters
- Log File Parameters

MAC003216

ORCL00941892

- Miscellaneous Web Listener Parameters
- User and Group Parameters
- Directory Mappings
- File Cache Definitions
- Language Extensions
- MIME Types
- Encoding Extensions
- Security Definitions for files and directories.

To go to the form for any of these sets of parameters, click it in the list. Alternatively, you can scroll through the Configure Extended Parameters Form to visit each parameter set in sequence except Security Definitions, which you can reach only by clicking.

These parameter sets are explained later in this chapter. For more information, see "Configuration Parameters" in Chapter 3.

Below each parameter set are three buttons. Click one to apply the changes you have made. The buttons are as follows:

Copy Web Listener

If you select this button, you must first change the Port Number listed in the Network Parameters section on this form. Then, the entire contents of this form will be used to create a new configuration file and thus a new Oracle Web Listener, leaving the existing Web Listener's configuration file intact. If you reached this form as part of a "create" operation, two Web Listeners are created.

Modify Web Listener

Pushing this button will overwrite the existing Oracle Web Listener's configuration file. This operation creates a new Web Listener and deletes the existing one. If you reached this form as part of a "create" operation, one Web Listener is created.

Delete Web Listener

Pushing this button will bring up a dialog for you to confirm the deletion. Confirming will do the following:

- Shut down the specified Oracle Web Listener if is currently running.

MAC003217

ORCL00941893

- Delete the specified Oracle Web Listener's configuration entry.
- Delete the specified Oracle Web Listener's configuration file.
- If you reached this form as part of a "create" operation, abort the create.

Network Parameters

This section defines the following parameters:

Host Name

The Host Name is the name of the machine on which the Web Listener executes. You must provide this parameter if the Web Listener cannot determine the host name or if the Web Listener executes under an alias. The Web Listener needs this information for logging purposes and to answer SNMP (Simple Network Management Protocol, an Internet standard protocol for checking the status of network components) requests for status. If no host name is specified here, the Web Listener uses the primary Internet address as the default.

Host Address

The Host Address is the IP address on which the Web Listener accepts connections. Specifying this address enables a Web Listener residing on a multi–homed machine to designate explicitly the direction from which it receives connections. Multi–homed machines are machines that are configured to have more than one IP address. Specifying ANY, which is the default, indicates that the Web Listener is willing to accept connections arriving on any of its available IP addresses. Leave the parameter set to ANY unless you have a specific reason for changing it.

Port Number

The Port Number is the advertised port on which the Web Listener listens for connections. Except for the value Port Number = 0, any port number is valid provided the Web Listener has the authority to use the requested port. Port numbers between 0 and 1023 are reserved ports and require root (or special) privileges in order to be used. Non–privileged processes can use any port greater than or equal to 1024 unless those ports are reserved for some other use. Ordinarily, the Web Listener uses 80 as the default port if no port number is specified

MAC003218

ORCL00941894

Max Connect Count

The Maximum Connect Count is the maximum number of simultaneous connections to the Web Listener that the Web Listener will accept. After the maximum count is reached, the Web Listener ignores requests for new connections. The default value is 50. The maximum and optimal value is 338.

DNS Resolution

Domain Name Service Resolution indicates whether and when the Web Listener will resolve hostnames from IP addresses. If the value of this parameter is set to ALWAYS, then the Web Listener always resolves the name. If it is set to LAZY, then the Web Listener resolves the name only on demand. If it is set to LAZY_WITH_CGI, then the Web Listener also resolves the name on demand for CGI applications. If it is set to NEVER, then the Web Listener never resolves the name, even on demand. The default is NEVER.

Web Listener PID

The Web Listener PID file is a one–line file that contains the Web Listener's process identifier. There is no default. This information can be used from a UNIX shell command line to stop the Web Listener, determine its status, or reinitialize a Web Listener that is already running. Normally, however, you use the Administration Utility itself to do these things.

Log File Parameters

The following parameters are defined in this section:

Log File Directory

The Log File Directory determines the directory where error and information logs are kept. You need to make sure that this directory actually exists, because the Web Listener will not create one if it does not. The Web Listener needs write permission on this directory and on the files it contains. There is no default.

Info File Name

The Log Information File contains periodic and event–driven messages that the Web Listener generates to help WebServer Administrators monitor their web site. The Oracle Web Listener records connections in the industry–standard Common Log Format. This format includes the

MAC003219

ORCL00941895

first line of the client's request, which specifies the action, URL, and protocol version. It also includes such information as the client's host address, whether the request was successful, and how many bytes were transmitted. This information may be used later to analyze what types of accesses are performed by which users, and to check for errors.

Error File Name

The Log Error File contains event-driven error messages that the Web Listener generates as it encounters problems with clients, resource access, and various other internal problems. There is no default.

Miscellaneous Web Listener Parameters

The following parameters are defined in this section:

Configuration Directory

This is the directory where the configuration file for the Web Listener is to be located. The current value is shown. If you change this value to a new directory, the file containing the changes you currently are making to all Web Listener parameters will be written to this new directory.

Initial File

The Initial File specifies the file that the Web Listener looks for when it is sent a directory URL. For example, if the Initial file is welcome, a request for http:/ / www/ stuff causes the Web Listener to return the file / stuff/ welcome. The default is initial.

User Directory

The User Directory is applicable only to Web Listeners running under the UNIX Operating System. It specifies the subdirectory in a user's directory the Web Listener searches when it receives a URL that starts with / ~username/ . For example, if at www, this value is set to public_html, then a request for http:/ / www/ ~charles/ initial.htm causes the Web Listener to return the file ~charles/ public_html/ initial.htm. There is no default.

User Directory Initial File

The User Directory Initial File is a special case. For directories accessed with a URL with / ~username/ , the Oracle Web Listener will not use HTTP incomplete file negotiation and so the Web Listener needs a

MAC003220

ORCL00941896

complete filename in the URL. The default is initial.htm.

Default MIME Type

The default MIME (Multimedia Internet Mail Extension) Type is the MIME type for the Web Listener to use if a URL uses a file name extension to request a MIME type the Web Listener does not support. MIME types specify the kind of file, such as image, audio, or video, and the subtype gives the precise file format. The default for this parameter is application/ octet–stream, which means the Web Listener will regard any unsupported MIME type as a binary file. If you would like the Web Listener to regard unsupported MIME types as anything other than application/ octet–stream, you need to set this parameter.

Default Character Set

This is the character set that the Web Listener uses if it does not recognize the character set specified in the URL's file extension. For more information, see "Language Extensions", later in this chapter.

Preferred Language

This is the language that the Web Listener uses if the specified or defaulted character set can apply to multiple languages, and the exact language is not specified. For more information, see "Language Extensions," later in this chapter.

Image Map

The contents of this field denote the extension that the Web Listener expects image maps to have. Thus, if your image maps have .map as their extension, set this parameter to map.

An image map is a graphic that is specially configured to act as a selector for one or more hyperlinks, based on where the user clicks on the graphic as displayed on the browser. Oracle WebServer processes image maps internally, without creating a separate process to handle them. An example of an image map would be a map of the world that allows the user to bring up an HTML document on a specific country by clicking on the country on the map. The Oracle Web Listener supports the same image map file formats as are read by the NCSA server.

MAC003221

ORCL00941897

Secure Information Parameters

The following parameters are applicable only to Web Listeners running on the UNIX Operating System:

User ID

The User ID specifies the user (or user identification number) under whose privileges the Web Listener executes. To acquire these privileges, the Web Listener must be owned by this user or by root with "Set user ID" set. The User ID can be either a user name or a userid number. The default is the effective ID.

Group ID

The Group ID specifies the group (or group identification number) under whose privileges the Web Listener executes. To acquire these privileges, the Web Listener must be owned by this group or by root with "Set group ID" set. The Group ID can be either a group name or a GID number. The default is the effective GID. For further information on the use of Group ID for security, see "Basic Configuration Parameters," earlier in this chapter.

Directory Mappings

The Directory Mappings control the mapping of directories from the file system into the Web Listener's virtual file system. The Oracle Web Listener supports the concept of a "virtual" or mapped filesystem. This feature enables the URL used to reference a set of files (directory) to be kept constant even if the files are moved. The translation of the network visible "virtual" path to an actual path is performed at runtime based on the mappings set in this section.

You specify one mapping per line, putting the actual path on the left, the virtual path on the right, and selecting from the pop–up menu one of the following:

- 'C', which indicates that the directory contains CGI or other scripts. Although the directory can contain multiple CGI scripts, it cannot be recursive. That is to say, none of the CGI scripts can be contained in subdirectories.
- 'R', which indicates that the subdirectories should be mapped recursively. That is to say, subdirectories of the current directory are included.

MAC003222

ORCL00941898

- 'N', which indicates that the subdirectories should not be mapped recursively. That is to say, subdirectories of the current directory are not included.

Although you can list parameters in most sections of the configuration file in any order, you must order the parameters in this section so that parent virtual directories are listed before the virtual subdirectories they contain. Hence, the first directory mapping must reference the virtual root directory. The following is an example for UNIX systems:

File-System Directory	R/N/C	Virtual Directory
/html	R	/
/prodinfo	N	/products
/final/xr25	R	/products/xr25
/misc/people	R	/employees
/misc/routines	C	/scripts

In the example above, /html is the virtual root directory. This means that any URL that does not contain a directory gets mapped to /html. Thus, the URL http://www/foo.html will cause the Web Listener to look for foo.html in the /html directory. Since this particular mapping is set to be recursive, any URL which contains a directory other than /products, /employees, and /scripts will get mapped to /html/ directory. Thus, the URL http://www/foo/bar.html will cause the Web Listener to look for bar.html in the /html/foo directory. Since /prodinfo is set to non-recursive, any URL that contains a subdirectory of /products causes an error. The exception would be a URL that contains /products/xr25, because the third line maps this virtual directory to /final/xr25. The last line is an example of a mapping to a directory that contains CGI and other scripts. Thus, the URL http://www/scripts/baz will cause the Web Listener to execute /misc/routines/baz.

> Note: If you change the directory mappings, you must reload the Web Listener to put the new mappings into effect. To do this, use the WLCTL utility, as described under "WLCTL" in Chapter 3 of this document.

File Cache Definitions

This section designates cached files. Files listed here are kept open for the life of the Web Listener. Files not listed here are not cached, and the Web Listener releases the resources required to access them once there are no outstanding client references to them—that is, once their reference counts drop to zero.

MAC003223

ORCL00941899

File caching decreases the access time required for commonly–used files, but at the cost of increasing the memory that the WebServer process uses. For this reason, you should not cache very large files unless absolutely necessary.

You can specify files for caching in any of the following three ways:

- by giving an explicit filename
- by giving an incomplete filename that ends in an asterisk to indicate that all files matching the portion before the asterisk are affected
- by giving a directory, in which case all files in that directory and all of its subdirectories are affected

File references must use virtual rather than actual paths. For more on virtual paths, see "Directory Mappings", earlier in this chapter.

Language Extensions

The Web Listener can recognize various character sets, which are associated with various languages. The character set and language to be associated with a file is indicated with an extension to the filename. This is the character set and language to be used for communication between the Web Listener and web browsers. It is independent of the character set used for the data itself, which is indicated by the Web Agent parameter NLS_LANG.

This section defines the filename extensions that the Web Listener recognizes and the languages and character sets to which those extensions map. Extensions are case–sensitive. Each line in this section of the configuration file contains a language identifier as described in RFC 1766, followed by a character set, such as those described in RFC 1521, followed by one or more extensions that represent this combination of language and character set to the Web Listener. Multiple extensions are separated by blanks. For more information on Oracle Web Listener's support for character sets, see "Oracle Web Listener Language Extensions" in Chapter 3 of this manual.

An example follows:

Language Type	Character Set	File Extension(s)
en	iso–8859–1	eng
en	unicode–1–1	engUuc
fr–CA	iso–8859–1	frc
fr–FR	iso–8859–1	fr
jp–JP	iso–2022–jp	jp
jp–JP	unicode–1–1–utf–8	jpU

MAC003224

ORCL00941900

Since character sets are used only on textual files, language extensions differing only in the character set are equivalent for non–textual files. For example, for a Web Listener configured with the type extensions shown in the example, the file image.eng.jpg and the file image.uc.jpg both indicate an International English version of an image.

You can find more information and further examples under "Oracle Web Listener Authentication Configuration" in Chapter 3 of this manual.

MIME Types

The Multipurpose Internet Mail Extension protocol (MIME) defines a number of content types and subtypes that allow programs to recognize different kinds of files and deal with them appropriately. The MIME type specifies what kind of file it is, such as image, audio, or video, and the subtype gives the precise file format. For more information on MIME, you can look over the RFC documents that describe MIME at the following URL:

http://www.oac.uci.edu/indiv/ehood/MIME/MIME.html

This section defines MIME types known to the Web Listener by their file name extensions. Each line in this section names a MIME type and then lists extensions that map to that MIME type. Extensions are case–sensitive, and multiple extensions are separated by blanks.

Encoding Extensions

This section defines the Web Listener's known encoding types and their expected extensions. When the Web Listener encounters files with extensions defined here, it recognizes the file to be of the corresponding type. Each line in this section names an encoding type and the extensions that map to this type. Extensions are case–sensitive, and multiple extensions are separated by blanks.

Configure Security Form

Use this form to define the security protections for the Web Listener to use on various pages. The top of the form is shown in Figure 7 – 8. The rest of the form provides text fields in which to enter the security parameters and values of your choice.

MAC003225

ORCL00941901

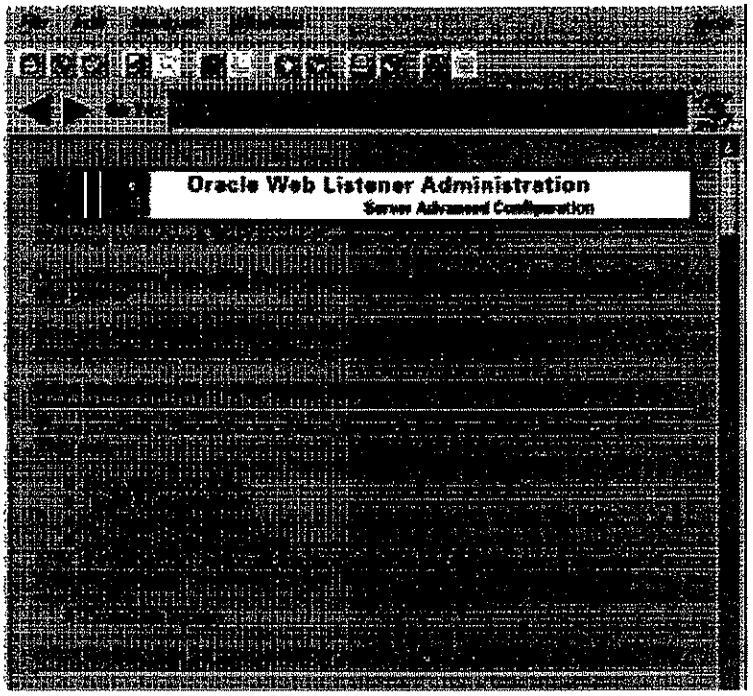


Figure 7–8 Web Listener Configure Security Form

There are two considerations:

- The security scheme. That is to say, how the Web Listener determines whether an access request comes from a sufficiently–privileged user. There are four possibilities, and you can impose more than one on the same material for extra security.
- Which users can access which files. You can define this both in terms of sets of users and sets of files. Specify files using virtual, rather than actual, paths. Virtual paths are explained under "Directory Mappings," earlier in this chapter.

The first of the following sections describes the security schemes in the abstract, the second describes the forms used to specify the users, and the third describes the form used to associate the users with the protected files or sets of files.

Click on a security scheme to navigate to the form which specifies the users in the manner relevant to that scheme, as described in the second section below. When you have completed these definitions, click

MAC003226

ORCL00941902

Protection Section to navigate to the form where you apply the security schemes by associating the specified users with the protected files. Alternatively, you can scroll down the Configure Security page to reach all these forms in sequence.

The Security Schemes

The four security schemes fall into two categories: authorization schemes and restriction schemes. Authorization schemes work on the principle of password protection—the user provides a password to verify his or her identity. Restriction schemes use the Internet itself to determine who the user is. Basic and Digest authentication are authorization schemes, while IP–based and Domain–based authentication are restriction schemes. All the schemes are explained in the paragraphs that follow.

Basic Authentication

Basic authentication employs usernames and passwords to restrict access to certain files on the Web Listener. Because Basic authentication sends passwords in the clear across the network, it is not recommended for extremely sensitive information.

Digest Authentication

Digest authentication works essentially the same way as Basic authentication, except that it does not send passwords in the clear. Instead, it uses a cryptographic checksum, often called a digest, to encode the password. This scheme is a significant improvement over Basic Authentication and should be used whenever possible. Unfortunately, this scheme depends on the cooperation of the browser to encrypt the password, and therefore does not work with older browsers that don't support this feature.

IP–based Restriction

IP–based restriction uses the network Internet Protocol (IP) address of the client to determine whether to allow or deny access to the protected page. Individual IP addresses or entire IP subnetworks may be specified. IP–based restriction is significantly more convenient than Basic and Digest authentication because it avoids the problems associated with passwords, such as keeping the password file secure and users forgetting their passwords. However, it is also less secure than Digest authentication because it is possible for a clever intruder to arrange to falsify his or her network address. Furthermore, many users need proxy servers because they are behind firewalls. In order to allow

MAC003227

ORCL00941903

access to such a user, the IP address of the proxy server needs to be specified. However, specifying the IP address of the proxy server provides access to the protected page to all users employing that proxy server.

Domain–based Restriction

Domain–based restriction is similar to IP–based restriction, except that it uses symbolic hostnames rather than network addresses. The use of symbolic hostnames is preferable because it is immune to changes in the underlying network architecture, which can result in changes to network addresses. Domain–based restriction, like IP–based restriction, is more convenient than Basic and Digest authentication, but also less secure than Digest authentication. As in IP–based restriction, it suffers from the problem with proxy servers.

Specifying The Privileged Users

After you have selected the security scheme or schemes to use, you need to specify the members of the privileged groups. The exact information that needs to be provided depends on the security scheme used.

Basic and Digest Authentication

For Basic and Digest Authentication, the information consists of a list of users, then groups, and then realms. Users and groups refer to the standard UNIX notion of users and groups. Realms are collections of groups that are assigned the same security privileges. In other words, they constitute a new (higher) level of the Unix grouping hierarchy. For example:

MAC003228

ORCL00941904

User Name	Password
mcarey:	achtung
mfoaf:	anything
chines:	pretend
pgabriel:	shockmonk

Group Name	User(s)
temps:	mcarey chines
sales:	chines mfoaf
analysts:	mfoaf pgabriel

Realms	Group(s)
red:	temps
blue:	sales temps
green:	analysts

In the above example, chines, whose password is pretend, belongs to the sales group, which is part of the blue realm. Thus, any files that are set to be accessible to the blue realm are accessible to chines as long as he types in pretend when prompted for his password. When specifying multiple users or groups, please make sure that they are separated by at least one space. Realms are non–exclusive, which is to say, users and groups can belong to multiple realms. A given realm can contain a mixture of users and groups.

IP–based Restriction

For IP–based restriction, you define groups of IP addresses that do or do not have access to a set of documents. For example:

IP Group	IP Address(es)
spyonly	+192.246.238.* +204.4.214.*
notspy	–192.246.238.* –204.4.214.* +*
friends	+10.0.0.11 +10.0.0.83 +10.0.0.117

A '+' before an IP address denotes that that particular IP address or subnet should be given access. A '–' before an IP address denotes that that particular IP address or subnet should be denied access. An asterisk, as usual, is a wildcard indicating that the entire subnet—that is, all IP addresses that begin with the preceding numbers—is included. You can later use the group names given here to refer to the entire complex of associated IP addresses, with their permissions or lack of same, and assign file permissions to these groups.

MAC003229

ORCL00941905

In the above example, documents in the spyonly group are accessible only from sytems within the two specified subnets. Documents in not spy are accessible only from systems outside the two specified subnets. Documents in friends are accessible only from the three specified systems.

Domain–based Restriction

Domain–based restriction is similar to IP–based restriction. A list of groups and hostnames are specified listing which hosts should or should not have access to a set of documents. The following is an example of domain–based restriction:

Domain Group	Domain Name(s)
oracle:	+*.oracle.com
misc:	+dilbert.redbull.com –*.redbull.com
	–liz.lotion.com +*

You later can use the group names given here to refer to the entire complex of associated domain names, just as in IP–based restriction.

In the above example, documents in the group oracle are accessible only to systems within the oracle domain. Documents in misc are accessible to connections from the specific system dilbert.redbull.com, but not any other system in the redbull.com domain. Documents in misc are also accessible to anyone outside of the redbull.com domain as long as they are not connecting from liz.lotion.com

Performing the Security Assignments – The Protection Section

Each line in the Protection Section of the form contains a file specification followed by one or two security schemes to protect that group of one or more files.

You can specify files in any of the following three ways:

- by giving an explicit filename
- by giving an incomplete filename that ends in an asterisk to indicate that all files matching the portion before the asterisk are affected
- by giving a directory, in which case all files in that directory and all of its subdirectories are affected

Fill in the file specification according to these criteria and use the pop–up menus to apply the security schemes. All files are specified

MAC003230

ORCL00941906

using virtual, rather than actual, paths. Virtual paths are explained under "Directory Mappings," earlier in this chapter.

You can apply one authentication and/ or one restriction scheme for each file specification. If you choose to use both, you must choose '&' or '|' from the '&|' pop–up menu to decide whether the user must satisfy both ('&') or only must satisfy either ('|') of the two schemes. Restriction schemes, naturally, are applied without user input.

To specify an authentication scheme, choose Basic or Digest from the pop–up menu. Then specify in the realm column the realm to which the scheme is to apply.

To specify a restriction scheme, choose IP or Domain from the pop–up menu. Then specify in the group column the group of IP addresses or Domain names, as appropriate, to which the scheme is to apply.

Here is an example:

Files	Basic/ Digest	Realm	&/|	IP/ Domain	Group
/secret/	Basic	red	|	IP	spyonly
/misc/ hot*	Basic	green	&	IP	spyonly
/company/*				IP	spyonly
/updat e. ht m	Basic	blue			

In this example, all of the files in /super secr et/, but not its subdirectories, can be accessed by any user who is either coming from the correct IP address or has a username and password for the red realm. In the second case, all files in /misc/ that start with "hot" can be accessed only by a user who has both the correct IP address and authentication for the green realm. Files in /company/, including subdirectories (as indicated by the asterisk (*)), are protected only by IP address. The single file updat e. ht m is protected only by authorization in the blue realm.

MAC003231

ORCL00941907

MAC003232

ORCL00941908

CHAPTER

7

The Oracle WebServer Administration Utility

This chapter shows you how to use the Administration Utility of the Oracle WebServer. The Administration Utility enables you to configure the components of the Oracle WebServer from any web browser, so you do not have to manually edit files to perform common operations.

To navigate to the Administration Utility, click Administration from the Oracle WebServer home page. The utility is divided into the following three areas:

- The Oracle7 Database Server. This enables you to start up or shut down an Oracle7 database.
- The Oracle Web Listener. This enables you to create an instance of the Oracle Web Listener or to modify the parameters of an existing instance.
- The Oracle Web Agent. This enables you to create or modify web agent services, including the specification of password–protected security.

The Administration page is shown in Figure 7 – 1. Note: If you are viewing this document online, there are no screen shots provided.

MAC003233

ORCL00941909

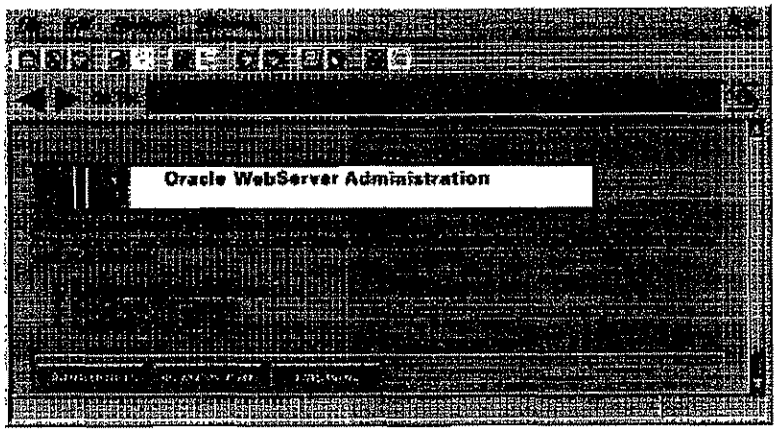


Figure 7 – 1 Administration Utility Form

Choose the area you want to use by clicking the appropriate button. This chapter contains a section for each administration area.

Database Administration

Use this page, shown in Figure 7 – 2, to start up or shut down any Oracle7 database running on the same machine as the Oracle Web Listener.

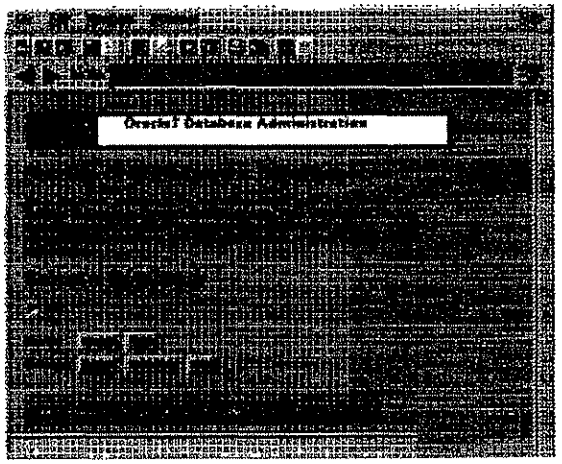


Figure 7 – 2 Database Administration Page

MAC003234

ORCL00941910

The page lists the available databases. This list comes from a file on the machine on which the Web Listener resides. On UNIX machines, this is known as the ORATAB file. See your platform–specific documentation for the location and format of this file. If no databases are listed, then the file contains no database descriptions.

The procedure is as follows:

1. Choose the database to which you want to apply the operation by clicking the radio button beside its name in the list.
2. Choose the operation you wish to perform and the mode by clicking the appropriate button. The modes are explained in the following sections.

For more information on database startup and shutdown, see the Oracle7 Server Administrator's Guide.

Startup Modes

You can perform database startup in either of the following two modes:

Normal

Starts up the database normally. This assumes that the database currently is shut down. If there are any errors during startup, they are reported, and the database is left shut down.

Force

Shuts down the database in the Abort mode (as explained under Shutdown Modes below), and then starts it up in the Normal mode.

Shutdown Modes

You can perform database shutdown in any of the following three modes:

Normal

Shuts down the database cleanly, waiting for current session to disconnect normally. New connections are not allowed while shutdown is in progress.

MAC003235

ORCL00941911

Immediate

Attempts to shut the database down immediately and cleanly. Disconnect any existing user sessions, but wait for active calls to complete before disconnecting. Roll back all uncommitted transactions

.

Abort

Shuts the database down as quickly as possible, killing all active Oracle7 processes and freeing the memory used. This leaves current transactions unresolved. When you start up the database again, instance recovery is performed automatically. This option is recommended only for emergency use.

For more information on Oracle7 databases including startup and shutdown, see the Oracle7 Server Concepts Manual and the Oracle7 Server Administrator's Guide.

Web Agent Administration

The Web Agent Administration page enables you to create, delete, or modify Web Agent Services, which Web Agents use to connect to the database. An Oracle7 Web Agent Service specifies values needed by the Web Agent every time it makes a database connection. For example, an Oracle7 username and password are needed, as well as an Oracle7 SID or SQL*Net V2 service name. For more information on Web Agent Services, see "Web Agent Service" in Chapter 5.

Creating or Modifying Web Agent Services

Use the Web Agent section of the Administration Utility to create a new Web Agent Service or to copy, delete, or modify an existing one. The initial Web Agent Administratin page is shown in Figure 7 – 3.

MAC003236

ORCL00941912

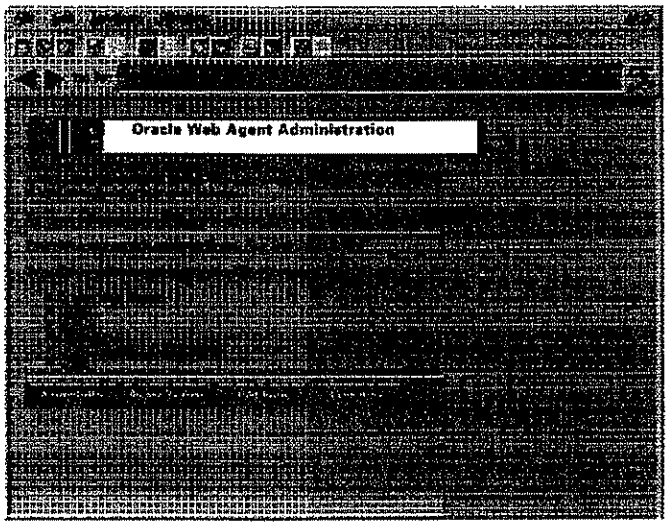


Figure 7–3 Initial Web Agent Administration Page

To create a new Oracle Web Agent Service, click Create New Service. The Oracle Web Agent Administration Creation form appears.

To modify, copy, or delete an existing Oracle Web Agent Service, click the name of that service in the list provided. The Oracle Web Agent Administration Modification form appears.

The Oracle Web Agent Administration Service Creation Form

Use this fill-out form, shown in Figure 7–4 to enter the parameters for the service you want to create. The parameters are explained below.

MAC003237

ORCL00941913

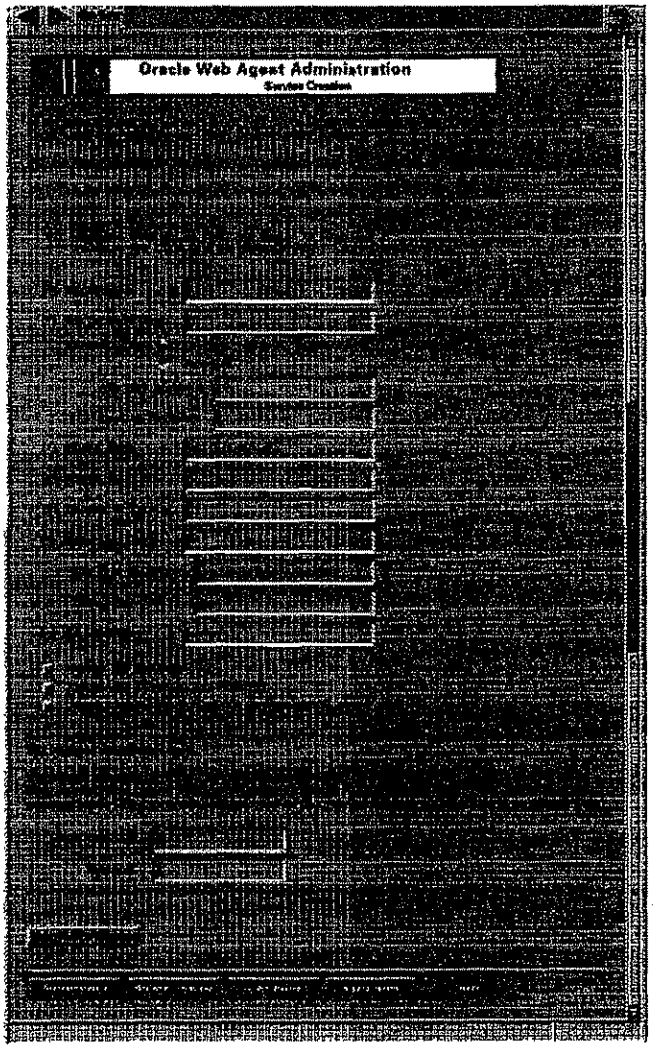


Figure 7–4 Create Web Agent Service Form

OWA Service:

The name of the service you want to create. The name is not case sensitive and must be unique.

MAC003238

ORCL00941914

OWA Database User:

The name of the Oracle7 user that will be used to access the database for this service. If operating system (OS) authentication is used, then the Web Listener must be run with the same effective user as the Oracle7 user. For more information on Oracle7 users, see "CREATE USER" in the Oracle7 Server SQL Reference.

Identified By

A radio button that determines whether Oracle7 or the Operating System performs authentication.

OWA User Password:

The Oracle7 password for the associated OWA Database User. If OS authentication is be used for the OWA Database User, as determined by the Identified By radio button, you can leave this field blank. Otherwise, the username and password for the OWA Database User are verified by Oracle7 upon submission of a Web Agent service.

If an error occurs when you attempt to submit this form, the password fields are blanked out and must be filled in again. This is to provide the tightest security currently available for your passwords. As a short-cut, you can select the Back button on your Web Browser to return to, and correct, the original submission form.

Confirm Password:

This field is used to verify the OWA user password you entered above before recording it.

ORACLE_HOME:

The location of the Oracle7 code tree in the file system. This should be the ORACLE_HOME for the database to which this Web Agent service connects, unless the Web Agent service is set up to connect to a remote database (over SQL*Net). In that case, specify the ORACLE_HOME where the Web Agent is installed.

ORACLE_SID:

The system ID of the database. You only need this if the Web Agent is to connect to a local database. For more on Oracle7 SIDs, see the Oracle7 Server Administrator's Guide.

MAC003239

ORCL00941915

SQL*Net V2 Service:

The SQL*Net service name that indicates which database to connect to. If the database is on the same machine as the Web Listener, then the SQL*Net V2 Service field can be left empty, and an Oracle7 SID specified instead. If both parameters are specified, the SQL*Net V2 Service takes precedence. If SQL*Net V1 is installed on this machine, then a SQL*Net V1 Connect String may also be specified here. For more on the SQL*Net V2 service name, see Understanding SQL*Net.

Authorized Ports:

To provide security for your Web Agent service, use the Authorized Ports field to list (space-delimited) the valid Web Listener network ports for the Web Agent to service. This field is required, and the network ports must be listed explicitly.

Log File Directory:

The directory where the Oracle Web Agent writes its error file. The error file will have the name service_name.err.

If the Web Agent is unable to resolve a service name, and no service called OWA_DEFAULT_SERVICE exists, then it will write error and logging information to the file owa.err.

The default value for the Log File Directory is the log directory under the OWS directory within ORACLE_HOME.

Note that these log files are different from those used by the Oracle7 Server for database recovery. The latter are explained in the Oracle7 Server Concepts Manual.

HTML Error Page :

If the Oracle Web Agent is unable to process a request, it returns this HTML page to the user making the request, and at the same time writes detailed errors and debugging information to the error message file in the Log File Directory. If this field is left blank, a default error message is returned to the client.

NLS Language :

This field specifies the NLS_LANG parameter, which specifies the character set to be used. If this field is left blank, then the Web Agent administration program looks it up in the configuration file for the

MAC003240

ORCL00941916

database upon service submission.

Create OWA Database User :

Create an Oracle7 user using the OWA user name and OWA user password provided. If the Identified by ... Operating System radio button is selected, then the user is created as identified externally, which means that the operating system, rather than Oracle, performs user authentication. In that case, this value is ignored.

Change OWA Database User Password:

Change the Oracle7 Database user's existing password to that specified by the OWA User Password. If the Identified by ... Operating System radio button is selected, then the user will be altered to be identified externally, which means that the operating system, rather than Oracle, performs user authentication.

Install Oracle WebServer Developer's Toolkit PL/SQL Packages

Create the Oracle WebServer Developer's Toolkit PL/ SQL packages—OWA, HTP, HTF, and OWA_UTIL—under the schema of the OWA Database User.

After filling in the above fields, click the Submit New Service button to create the service. To activate your new service, go to the Oracle WebServer Administration form and click Configure for the Web Listener you want to associate with this agent. Then go to "Directory Mappings" to create a virtual path, with the service name as the last part of the path, to the OWS/ bin directory. Virtual paths are discussed under "Directory Mappings", later in this chapter.

The Oracle Web Agent Administration Service Modification Form

This form contains the same fields as the Oracle Web Agent Administration Creation Form. The fields show their current values except for the password fields, which are left blank for security reasons. You can leave these fields blank unless you want to change the password.

MAC003241

ORCL00941917

At the bottom of the form are the following three buttons:

Copy Service

This creates a copy of the service, leaving the original intact. To do this, simply change the service name (the OWA service field) and any other parameters you wish to alter for the new copy and click this button. If you specify no new password, the original password is copied. After clicking this button, you have two services.

Modify Service

This changes the existing service. You can change the name or leave it intact as desired. Make the desired changes and click the button. After clicking this button, you have one service.

Delete Service

This destroys the service. For this purpose, it does not matter whether you change any parameters. After clicking this button, you have no service. Calls to the deleted service in existing Web pages or PL/ SQL blocks do not produce errors, however. Instead, they are reinterpreted as calls to OWA_DEFAULT_SERVICE.

Securing a Web Agent Service

The mechanism of "services" provides a security framework such that individual Web Agent services can be password protected by the Web Listener. For example, to password protect the hr service, you would do the following:

1. Create a virtual path for the hr service by specifying a Web Listener directory mapping. This mapping must use a value of 'C' to indicate an executable. For more information on setting the directory mappings, see Directory Mappings later in this chapter.
2. Create a realm consisting of those users and groups (UNIX only) who are to have access to the service. For instruction on how to do this, see Specifying the Privileged Users, later in this chapter.
3. Assign the realm to the service and specify either Basic or Digest Authentication. To learn how to perform the assignment, see Performing the Security Assignments, later in this chapter. To find out about Basic and Digest Authentication, see Security Schemes, later in this chapter.

MAC003242

ORCL00941918

For more information of Web Listener security in general, see "Security Definitions", later in this chapter, and "Client to Listener Security" in Chapter 3.

Web Listener Administration

The Oracle Web Listener is a reliable, high–performance HTTP server suitable for high–traffic web applications where quick response is crucial.

From the Web Listener Administration page, shown in Figure 7–5, you can move to other pages that provide a convenient way for you to do the following:

- Create a new Web Listener
- Change or copy an existing Web Listener
- Start a Web Listener
- Stop a Web Listener
- See whether an Oracle Web Listener is currently running.

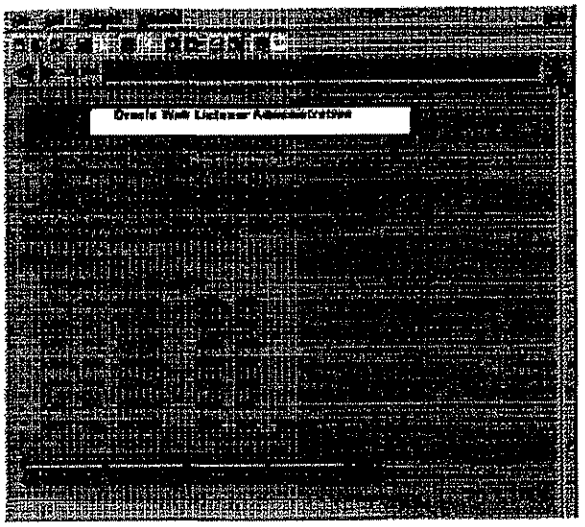


Figure 7–5 Web Listener Administration Page

To create a new Oracle Web Listener, click Create New Oracle Web Listener. This takes you to the Listener Creation form.

MAC003243

ORCL00941919

To perform any of the other operations, choose the appropriate existing Web Listener from the list provided. This list uniquely identifies each Web Listener by the network port on which it is configured to listen. For your convenience, the directory in which the configuration file resides is specified beside the port number. Unlike the port number, this directory need not be unique.

If a Web Listener is running, the word "running" appears beside it in the list.

To modify an existing Oracle Web Listener, select it in the list and click Configure. This takes you to the Configure Extended Parameters form, with the current parameter values filled in. Effect the changes by writing over the current values. For information on the parameters themselves, see "Configure Extended Parameters," later in this chapter.

To start a currently–stopped Oracle Web Listener, select it in the list and click Start.

To stop a currently–running Oracle Web Listener, select it in the list and click Stop.

> Note: You might need to perform a manual startup of an existing Web Listener if you inadvertently shut down the Web Listener that the Administration Utility itself is using. To find out how to do this, see "The WLCTL" in Chapter 3.

The Oracle Web Listener Creation Form

Use this form, shown in the next two figures, to create a new Oracle Web Listener by filling in the listed parameters. These are the parameters the Web Listener requires in order to run. After filling in all required parameters, press the appropriate button to choose a Basic Configuration or an Advanced Configuration. In either case, the Web Listener is created. If you choose Basic Configuration, the parameters you fill in here are all that are needed, and you can start the Web Listener you have created. If you choose Advanced Configuration, you go to the Configure Extended Parameters form to provide optional information. This is equivalent to selecting an existing Web Listener from the list in the initial Web Listener Administration page and clicking Configure. For more information on the Configure Extended Parameters form, see "The Configure Extended Parameters Form" later in this chapter.

MAC003244

ORCL00941920

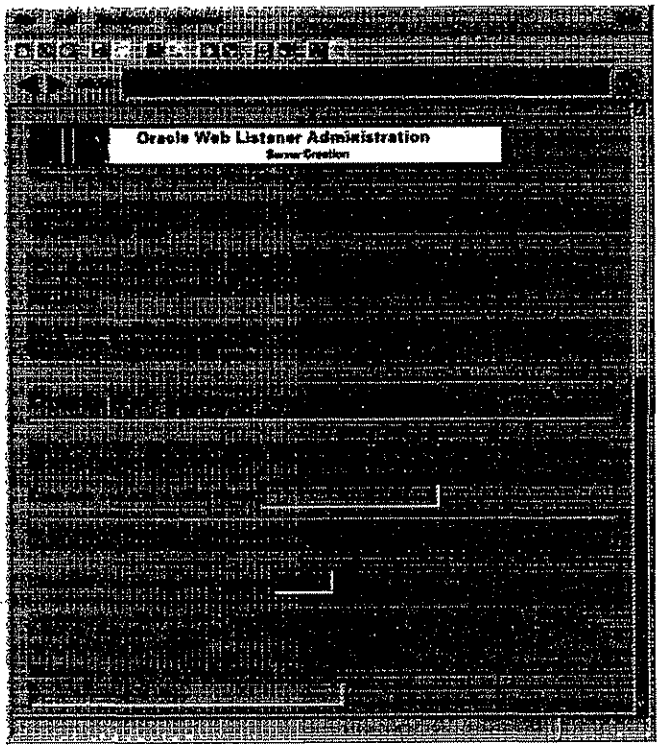


Figure 7–6 Listener Creation Form, Part 1

The Basic Configuration Parameters

The parameters required for a basic configuration are explained below:

Host Name

This is the name of the machine on which the Oracle Web Listener is to reside. It can either be an actual machine name or an alias in the form of a domain name, such as:

www. or acl e. com

Port Number

The Oracle Web Listener listens for network messages on a particular port. Use this parameter to specify that port. Numbers can range from 0 to 65535 with numbers below 1024 requiring superuser privileges on UNIX systems.

MAC003245

ORCL00941921

Configuration Directory

This is the directory where the configuration file for the Web Listener is to be located. A default is provided.

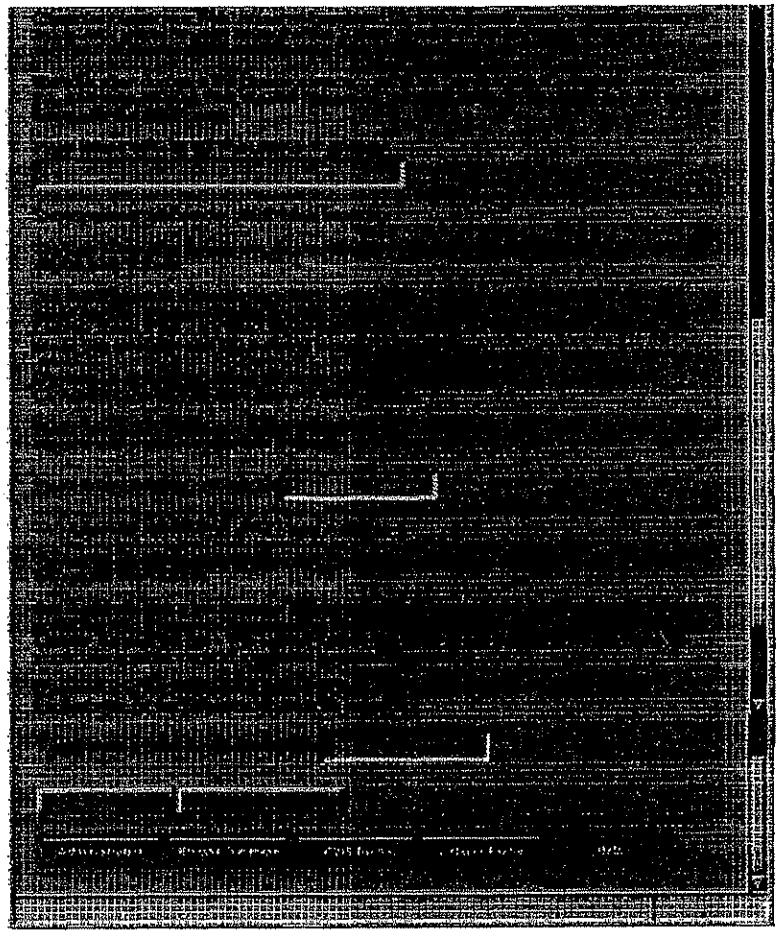


Figure 7 – 7 Listener Creation Form, Part 2

Web Listener Root (/) Directory

This is the directory that serves as the root (/) directory for the Web Listener. This is often called the "Document Root". It is used to resolve client calls that do not specify a directory. Thus a URL that does not contain a directory, such as http:/ / www.oracle.com/ index.html,

MAC003246

ORCL00941922

causes the Web Listener to retrieve the document "index.html" from the directory specified here.

User ID (UNIX only)

The User ID specifies the user (or user identification number) under whose privileges the Web Listener executes. To acquire these privileges, the Web Listener must be owned by this user or by root with "Set user ID" set. The User ID can be either a username or a user ID number. The default is the effective ID. Specifying the Oracle owner or an Oracle user with the DBA role is not advised, as this compromises database security. For more information, see "roles" and "Database Administrators" in the Oracle7 Server Concepts Manual.

Group ID (UNIX only)

The Group ID specifies the group (or group identification number) under whose privileges the Web Listener executes. To acquire these privileges, the Web Listener must be owned by this group or by root with "Set group ID" set. The Group ID can be either a group name or a GID number. The default is the effective GID.

Note that any Oracle Web Listener that invokes the Oracle Web Agent must have read access to the OWA configuration file (owa.cfg). However, since this file contains Oracle usernames and their corresponding passwords, it is recommended that read permissions only be given to the oracle owner and to one privileged group. This is the group that a Web Listener should run under if it is going to run the Oracle Web Agent.

A convenient group to use for this is the OracleDBA group. However, if you run an Oracle Web Listener under the DBA group, you will want to place strict controls on who can create CGI programs that will be invoked by that Web Listener. This is because the CGI program will run with group DBA privileges, and someone with malicious intent could potentially startup and shutdown your Oracle database, or even remove Oracle files. If you wish to avoid this potential security risk, then you need to do the following:

1. Create a new operating system group.
2. Make the Oracle owner a member of the group created in step 1.
3. Log in as the Oracle owner and change the group ownership of owa.cfg to the group created in step 1.

   Note: If at any future point in time you remove the owa.cfg file, you must reset its group ownership when it is recreated.

MAC003247

ORCL00941923

4. Configure the Group ID parameter of your Oracle Web Listener to be the name of the group created in step 1.
5. Restart your Administration Server (the Web Listener that the Administration Utility is using). You must do this to ensure that the Web Agent Administration program inherits the group created in step 1. Do not change the Group ID for the administration server. It must continue to run as the OracleDBA group.

Starting a New Oracle Web Listener

Once you have create a Web Listener, you are shown a Success! message. At this point, you are ready to start your new Oracle Web Listener. Ordinarily, you do this by navigating to the Oracle Web Listener Administration page and clicking start. Note that most web browsers buffer visited pages, and therefore do not reload them when you return to them using something like a Back button. For the new Web Listener to be listed as available for startup, you must navigate to it in such a way that the page is reloaded, for example by reentering the URL.

Reinitializing the Oracle Web Listener

If you selected Advanced Configuration and all the parameters you entered were valid, you will be shown a Success! message. You will then be presented with the Web Listener Configure Extended Parameters form which allows you to modify any of the Oracle Web Listener configuration parameters. It is important to remember that once the Web Listener is started, however, it does not automatically recognize new files added to its virtual file system or parameters changed in its configuration file. If you would like to add files to the Web Listener, you should navigate to the Web Listener Administration page and click Configure next to the Web Listener you want to modify.

The Configure Extended Parameters Form

This form provides a convenient way to customize an existing Oracle Web Listener for your particular environment. By using this form, you can avoid having to manually edit the Web Listener configuration file, thus eliminating the need to be familiar with the exact syntax for each parameter.

From this form, you will be able to customize the following:

- Network Parameters
- Log File Parameters

MAC003248

ORCL00941924

- Miscellaneous Web Listener Parameters
- User and Group Parameters
- Directory Mappings
- File Cache Definitions
- Language Extensions
- MIME Types
- Encoding Extensions
- Security Definitions for files and directories.

To go to the form for any of these sets of parameters, click it in the list. Alternatively, you can scroll through the Configure Extended Parameters Form to visit each parameter set in sequence except Security Definitions, which you can reach only by clicking.

These parameter sets are explained later in this chapter. For more information, see "Configuration Parameters" in Chapter 3.

Below each parameter set are three buttons. Click one to apply the changes you have made. The buttons are as follows:

Copy Web Listener

If you select this button, you must first change the Port Number listed in the Network Parameters section on this form. Then, the entire contents of this form will be used to create a new configuration file and thus a new Oracle Web Listener, leaving the existing Web Listener's configuration file intact. If you reached this form as part of a "create" operation, two Web Listeners are created.

Modify Web Listener

Pushing this button will overwrite the existing Oracle Web Listener's configuration file. This operation creates a new Web Listener and deletes the existing one. If you reached this form as part of a "create" operation, one Web Listener is created.

Delete Web Listener

Pushing this button will bring up a dialog for you to confirm the deletion. Confirming will do the following:

- Shut down the specified Oracle Web Listener if is currently running.

MAC003249

ORCL00941925

- Delete the specified Oracle Web Listener's configuration entry.
- Delete the specified Oracle Web Listener's configuration file.
- If you reached this form as part of a "create" operation, abort the create.

Network Parameters

This section defines the following parameters:

Host Name

The Host Name is the name of the machine on which the Web Listener executes. You must provide this parameter if the Web Listener cannot determine the host name or if the Web Listener executes under an alias. The Web Listener needs this information for logging purposes and to answer SNMP (Simple Network Management Protocol, an Internet standard protocol for checking the status of network components) requests for status. If no host name is specified here, the Web Listener uses the primary Internet address as the default.

Host Address

The Host Address is the IP address on which the Web Listener accepts connections. Specifying this address enables a Web Listener residing on a multi–homed machine to designate explicitly the direction from which it receives connections. Multi–homed machines are machines that are configured to have more than one IP address. Specifying ANY, which is the default, indicates that the Web Listener is willing to accept connections arriving on any of its available IP addresses. Leave the parameter set to ANY unless you have a specific reason for changing it.

Port Number

The Port Number is the advertised port on which the Web Listener listens for connections. Except for the value Port Number = 0, any port number is valid provided the Web Listener has the authority to use the requested port. Port numbers between 0 and 1023 are reserved ports and require root (or special) privileges in order to be used. Non–privileged processes can use any port greater than or equal to 1024 unless those ports are reserved for some other use. Ordinarily, the Web Listener uses 80 as the default port if no port number is specified

MAC003250

ORCL00941926

Max Connect Count

The Maximum Connect Count is the maximum number of simultaneous connections to the Web Listener that the Web Listener will accept. After the maximum count is reached, the Web Listener ignores requests for new connections. The default value is 50. The maximum and optimal value is 338.

DNS Resolution

Domain Name Service Resolution indicates whether and when the Web Listener will resolve hostnames from IP addresses. If the value of this parameter is set to ALWAYS, then the Web Listener always resolves the name. If it is set to LAZY, then the Web Listener resolves the name only on demand. If it is set to LAZY_WITH_CGI, then the Web Listener also resolves the name on demand for CGI applications. If it is set to NEVER, then the Web Listener never resolves the name, even on demand. The default is NEVER.

Web Listener PID

The Web Listener PID file is a one–line file that contains the Web Listener's process identifier. There is no default. This information can be used from a UNIX shell command line to stop the Web Listener, determine its status, or reinitialize a Web Listener that is already running. Normally, however, you use the Administration Utility itself to do these things.

Log File Parameters

The following parameters are defined in this section:

Log File Directory

The Log File Directory determines the directory where error and information logs are kept. You need to make sure that this directory actually exists, because the Web Listener will not create one if it does not. The Web Listener needs write permission on this directory and on the files it contains. There is no default.

Info File Name

The Log Information File contains periodic and event–driven messages that the Web Listener generates to help WebServer Administrators monitor their web site. The Oracle Web Listener records connections in the industry–standard Common Log Format. This format includes the

MAC003251

ORCL00941927

first line of the client's request, which specifies the action, URL, and protocol version. It also includes such information as the client's host address, whether the request was successful, and how many bytes were transmitted. This information may be used later to analyze what types of accesses are performed by which users, and to check for errors.

Error File Name

The Log Error File contains event–driven error messages that the Web Listener generates as it encounters problems with clients, resource access, and various other internal problems. There is no default.

Miscellaneous Web Listener Parameters

The following parameters are defined in this section:

Configuration Directory

This is the directory where the configuration file for the Web Listener is to be located. The current value is shown. If you change this value to a new directory, the file containing the changes you currently are making to all Web Listener parameters will be written to this new directory.

Initial File

The Initial File specifies the file that the Web Listener looks for when it is sent a directory URL. For example, if the Initial file is welcome, a request for http:/ / www/ stuff causes the Web Listener to return the file / stuff/ welcome. The default is initial.

User Directory

The User Directory is applicable only to Web Listeners running under the UNIX Operating System. It specifies the subdirectory in a user's directory the Web Listener searches when it receives a URL that starts with / ~username/ . For example, if at www, this value is set to public_html, then a request for http:/ / www/ ~charles/ initial.htm causes the Web Listener to return the file ~charles/ public_html/ initial.htm. There is no default.

User Directory Initial File

The User Directory Initial File is a special case. For directories accessed with a URL with / ~username/ , the Oracle Web Listener will not use HTTP incomplete file negotiation and so the Web Listener needs a

MAC003252

ORCL00941928

complete filename in the URL. The default is initial.htm.

Default MIME Type

The default MIME (Multimedia Internet Mail Extension) Type is the MIME type for the Web Listener to use if a URL uses a file name extension to request a MIME type the Web Listener does not support. MIME types specify the kind of file, such as image, audio, or video, and the subtype gives the precise file format. The default for this parameter is application/ octet–stream, which means the Web Listener will regard any unsupported MIME type as a binary file. If you would like the Web Listener to regard unsupported MIME types as anything other than application/ octet–stream, you need to set this parameter.

Default Character Set

This is the character set that the Web Listener uses if it does not recognize the character set specified in the URL's file extension. For more information, see "Language Extensions", later in this chapter.

Preferred Language

This is the language that the Web Listener uses if the specified or defaulted character set can apply to multiple languages, and the exact language is not specified. For more information, see "Language Extensions," later in this chapter.

Image Map

The contents of this field denote the extension that the Web Listener expects image maps to have. Thus, if your image maps have .map as their extension, set this parameter to map.

An image map is a graphic that is specially configured to act as a selector for one or more hyperlinks, based on where the user clicks on the graphic as displayed on the browser. Oracle WebServer processes image maps internally, without creating a separate process to handle them. An example of an image map would be a map of the world that allows the user to bring up an HTML document on a specific country by clicking on the country on the map. The Oracle Web Listener supports the same image map file formats as are read by the NCSA server.

MAC003253

ORCL00941929

Secure Information Parameters

The following parameters are applicable only to Web Listeners running on the UNIX Operating System:

User ID

The User ID specifies the user (or user identification number) under whose privileges the Web Listener executes. To acquire these privileges, the Web Listener must be owned by this user or by root with "Set user ID" set. The User ID can be either a user name or a userid number. The default is the effective ID.

Group ID

The Group ID specifies the group (or group identification number) under whose privileges the Web Listener executes. To acquire these privileges, the Web Listener must be owned by this group or by root with "Set group ID" set. The Group ID can be either a group name or a GID number. The default is the effective GID. For further information on the use of Group ID for security, see "Basic Configuration Parameters," earlier in this chapter.

Directory Mappings

The Directory Mappings control the mapping of directories from the file system into the Web Listener's virtual file system. The Oracle Web Listener supports the concept of a "virtual" or mapped filesystem. This feature enables the URL used to reference a set of files (directory) to be kept constant even if the files are moved. The translation of the network visible "virtual" path to an actual path is performed at runtime based on the mappings set in this section.

You specify one mapping per line, putting the actual path on the left, the virtual path on the right, and selecting from the pop–up menu one of the following:

- 'C', which indicates that the directory contains CGI or other scripts. Although the directory can contain multiple CGI scripts, it cannot be recursive. That is to say, none of the CGI scripts can be contained in subdirectories.
- 'R', which indicates that the subdirectories should be mapped recursively. That is to say, subdirectories of the current directory are included.

MAC003254

ORCL00941930

- 'N', which indicates that the subdirectories should not be mapped recursively. That is to say, subdirectories of the current directory are not included.

Although you can list parameters in most sections of the configuration file in any order, you must order the parameters in this section so that parent virtual directories are listed before the virtual subdirectories they contain. Hence, the first directory mapping must reference the virtual root directory. The following is an example for UNIX systems:

File-System Directory	R/N/C	Virtual Directory
/html	R	/
/prodinfo	N	/products
/final/xr25	R	/products/xr25
/misc/people	R	/employees
/misc/routines	C	/scripts

In the example above, /html is the virtual root directory. This means that any URL that does not contain a directory gets mapped to /html. Thus, the URL http://www/foo.html will cause the Web Listener to look for foo.html in the /html directory. Since this particular mapping is set to be recursive, any URL which contains a directory other than /products, /employees, and /scripts will get mapped to /html/ directory. Thus, the URL http://www/foo/bar.html will cause the Web Listener to look for bar.html in the /html/foo directory. Since /prodinfo is set to non-recursive, any URL that contains a subdirectory of /products causes an error. The exception would be a URL that contains /products/xr25, because the third line maps this virtual directory to /final/xr25. The last line is an example of a mapping to a directory that contains CGI and other scripts. Thus, the URL http://www/scripts/baz will cause the Web Listener to execute /misc/routines/baz.

> Note: If you change the directory mappings, you must reload the Web Listener to put the new mappings into effect. To do this, use the WLCTL utility, as described under "WLCTL" in Chapter 3 of this document.

File Cache Definitions

This section designates cached files. Files listed here are kept open for the life of the Web Listener. Files not listed here are not cached, and the Web Listener releases the resources required to access them once there are no outstanding client references to them—that is, once their reference counts drop to zero.

MAC003255

ORCL00941931

File caching decreases the access time required for commonly–used files, but at the cost of increasing the memory that the WebServer process uses. For this reason, you should not cache very large files unless absolutely necessary.

You can specify files for caching in any of the following three ways:

- by giving an explicit filename
- by giving an incomplete filename that ends in an asterisk to indicate that all files matching the portion before the asterisk are affected
- by giving a directory, in which case all files in that directory and all of its subdirectories are affected

File references must use virtual rather than actual paths. For more on virtual paths, see "Directory Mappings", earlier in this chapter.

Language Extensions

The Web Listener can recognize various character sets, which are associated with various languages. The character set and language to be associated with a file is indicated with an extension to the filename. This is the character set and language to be used for communication between the Web Listener and web browsers. It is independent of the character set used for the data itself, which is indicated by the Web Agent parameter NLS_LANG.

This section defines the filename extensions that the Web Listener recognizes and the languages and character sets to which those extensions map. Extensions are case–sensitive. Each line in this section of the configuration file contains a language identifier as described in RFC 1766, followed by a character set, such as those described in RFC 1521, followed by one or more extensions that represent this combination of language and character set to the Web Listener. Multiple extensions are separated by blanks. For more information on Oracle Web Listener's support for character sets, see "Oracle Web Listener Language Extensions" in Chapter 3 of this manual.

An example follows:

Language Type	Character Set	File Extension(s)
en	iso–8859–1	eng
en	unicode–1–1	engUuc
fr–CA	iso–8859–1	frc
fr–FR	iso–8859–1	fr
jp–JP	iso–2022–jp	jp
jp–JP	unicode–1–1–utf–8	jpU

MAC003256

ORCL00941932

Since character sets are used only on textual files, language extensions differing only in the character set are equivalent for non–textual files. For example, for a Web Listener configured with the type extensions shown in the example, the file image.eng.jpg and the file image.uc.jpg both indicate an International English version of an image.

You can find more information and further examples under "Oracle Web Listener Authentication Configuration" in Chapter 3 of this manual.

MIME Types

The Multipurpose Internet Mail Extension protocol (MIME) defines a number of content types and subtypes that allow programs to recognize different kinds of files and deal with them appropriately. The MIME type specifies what kind of file it is, such as image, audio, or video, and the subtype gives the precise file format. For more information on MIME, you can look over the RFC documents that describe MIME at the following URL:

http://www.oac.uci.edu/indiv/ehood/MIME/MIME.html

This section defines MIME types known to the Web Listener by their file name extensions. Each line in this section names a MIME type and then lists extensions that map to that MIME type. Extensions are case–sensitive, and multiple extensions are separated by blanks.

Encoding Extensions

This section defines the Web Listener's known encoding types and their expected extensions. When the Web Listener encounters files with extensions defined here, it recognizes the file to be of the corresponding type. Each line in this section names an encoding type and the extensions that map to this type. Extensions are case–sensitive, and multiple extensions are separated by blanks.

Configure Security Form

Use this form to define the security protections for the Web Listener to use on various pages. The top of the form is shown in Figure 7–8. The rest of the form provides text fields in which to enter the security parameters and values of your choice.

MAC003257

ORCL00941933

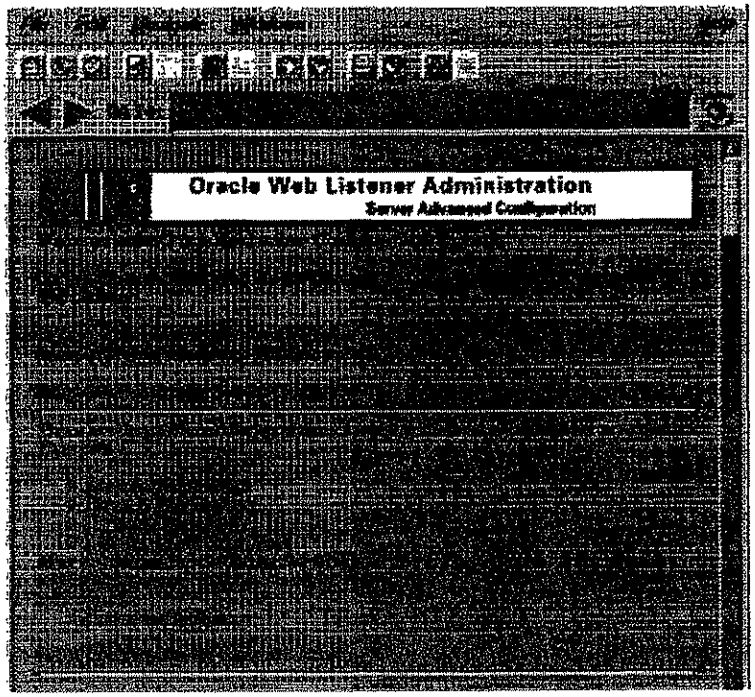


Figure 7–8 Web Listener Configure Security Form

There are two considerations:

- The security scheme. That is to say, how the Web Listener determines whether an access request comes from a sufficiently–privileged user. There are four possibilities, and you can impose more than one on the same material for extra security.
- Which users can access which files. You can define this both in terms of sets of users and sets of files. Specify files using virtual, rather than actual, paths. Virtual paths are explained under "Directory Mappings," earlier in this chapter.

The first of the following sections describes the security schemes in the abstract, the second describes the forms used to specify the users, and the third describes the form used to associate the users with the protected files or sets of files.

Click on a security scheme to navigate to the form which specifies the users in the manner relevant to that scheme, as described in the second section below. When you have completed these definitions, click

MAC003258

ORCL00941934

Protection Section to navigate to the form where you apply the security schemes by associating the specified users with the protected files. Alternatively, you can scroll down the Configure Security page to reach all these forms in sequence.

The Security Schemes

The four security schemes fall into two categories: authorization schemes and restriction schemes. Authorization schemes work on the principle of password protection—the user provides a password to verify his or her identity. Restriction schemes use the Internet itself to determine who the user is. Basic and Digest authentication are authorization schemes, while IP–based and Domain–based authentication are restriction schemes. All the schemes are explained in the paragraphs that follow.

Basic Authentication

Basic authentication employs usernames and passwords to restrict access to certain files on the Web Listener. Because Basic authentication sends passwords in the clear across the network, it is not recommended for extremely sensitive information.

Digest Authentication

Digest authentication works essentially the same way as Basic authentication, except that it does not send passwords in the clear. Instead, it uses a cryptographic checksum, often called a digest, to encode the password. This scheme is a significant improvement over Basic Authentication and should be used whenever possible. Unfortunately, this scheme depends on the cooperation of the browser to encrypt the password, and therefore does not work with older browsers that don't support this feature.

IP–based Restriction

IP–based restriction uses the network Internet Protocol (IP) address of the client to determine whether to allow or deny access to the protected page. Individual IP addresses or entire IP subnetworks may be specified. IP–based restriction is significantly more convenient than Basic and Digest authentication because it avoids the problems associated with passwords, such as keeping the password file secure and users forgetting their passwords. However, it is also less secure than Digest authentication because it is possible for a clever intruder to arrange to falsify his or her network address. Furthermore, many users need proxy servers because they are behind firewalls. In order to allow

MAC003259

ORCL00941935

access to such a user, the IP address of the proxy server needs to be specified. However, specifying the IP address of the proxy server provides access to the protected page to all users employing that proxy server.

Domain–based Restriction

Domain–based restriction is similar to IP–based restriction, except that it uses symbolic hostnames rather than network addresses. The use of symbolic hostnames is preferable because it is immune to changes in the underlying network architecture, which can result in changes to network addresses. Domain–based restriction, like IP–based restriction, is more convenient than Basic and Digest authentication, but also less secure than Digest authentication. As in IP–based restriction, it suffers from the problem with proxy servers.

Specifying The Privileged Users

After you have selected the security scheme or schemes to use, you need to specify the members of the privileged groups. The exact information that needs to be provided depends on the security scheme used.

Basic and Digest Authentication

For Basic and Digest Authentication, the information consists of a list of users, then groups, and then realms. Users and groups refer to the standard UNIX notion of users and groups. Realms are collections of groups that are assigned the same security privileges. In other words, they constitute a new (higher) level of the Unix grouping hierarchy. For example:

MAC003260

ORCL00941936

```
User Name    Password
---------    --------
mcarey:      achtung
mfoaf:       anything
chines:      pretend
pgabriel:    shockmonk

Group Name   User(s)
----------   -------
temps:       mcarey chines
sales:       chines mfoaf
analysts:    mfoaf pgabriel

Realms       Group(s)
------       --------
red:         temps
blue:        sales temps
green:       analysts
```

In the above example, chines, whose password is pretend, belongs to the sales group, which is part of the blue realm. Thus, any files that are set to be accessible to the blue realm are accessible to chines as long as he types in pretend when prompted for his password. When specifying multiple users or groups, please make sure that they are separated by at least one space. Realms are non–exclusive, which is to say, users and groups can belong to multiple realms. A given realm can contain a mixture of users and groups.

IP–based Restriction

For IP–based restriction, you define groups of IP addresses that do or do not have access to a set of documents. For example:

```
IP Group     IP Address(es)
--------     --------------
spyonly      +192.246.238.*  +204.4.214.*
notspy       –192.246.238.*  –204.4.214.*  +*
friends      +10.0.0.11  +10.0.0.83  +10.0.0.117
```

A '+' before an IP address denotes that that particular IP address or subnet should be given access. A '–' before an IP address denotes that that particular IP address or subnet should be denied access. An asterisk, as usual, is a wildcard indicating that the entire subnet—that is, all IP addresses that begin with the preceding numbers—is included. You can later use the group names given here to refer to the entire complex of associated IP addresses, with their permissions or lack of same, and assign file permissions to these groups.

MAC003261

ORCL00941937

In the above example, documents in the spyonly group are accessible only from sytems within the two specified subnets. Documents in not spy are accessible only from systems outside the two specified subnets. Documents in friends are accessible only from the three specified systems.

Domain–based Restriction

Domain–based restriction is similar to IP–based restriction. A list of groups and hostnames are specified listing which hosts should or should not have access to a set of documents. The following is an example of domain–based restriction:

Domain Group	Domain Name(s)
oracle:	+*.oracle.com
misc:	+dilbert.redbull.com –*.redbull.com –liz.lotion.com +*

You later can use the group names given here to refer to the entire complex of associated domain names, just as in IP–based restriction.

In the above example, documents in the group oracle are accessible only to systems within the oradedomain. Documents in misc are accessible to connections from the specific system dilbert.redbull.com, but not any other system in the redbull.com domain. Documents in misc are also accessible to anyone outside of the redbull.comdomain as long as they are not connecting from liz.lotion.com

Performing the Security Assignments — The Protection Section

Each line in the Protection Section of the form contains a file specification followed by one or two security schemes to protect that group of one or more files.

You can specify files in any of the following three ways:

- by giving an explicit filename
- by giving an incomplete filename that ends in an asterisk to indicate that all files matching the portion before the asterisk are affected
- by giving a directory, in which case all files in that directory and all of its subdirectories are affected

Fill in the file specification according to these criteria and use the pop–up menus to apply the security schemes. All files are specified

MAC003262

ORCL00941938

using virtual, rather than actual, paths. Virtual paths are explained under "Directory Mappings," earlier in this chapter.

You can apply one authentication and/ or one restriction scheme for each file specification. If you choose to use both, you must choose '&' or '| ' from the '&| ' pop–up menu to decide whether the user must satisfy both ('&') or only must satisfy either ('| ') of the two schemes. Restriction schemes, naturally, are applied without user input.

To specify an authentication scheme, choose Basic or Digest from the pop–up menu. Then specify in the realm column the realm to which the scheme is to apply.

To specify a restriction scheme, choose IP or Domain from the pop–up menu. Then specify in the group column the group of IP addresses or Domain names, as appropriate, to which the scheme is to apply.

Here is an example:

Files	Basic/Digest	Realm	&/|	IP/Domain	Group
/secret/	Basic	red	|	IP	spyonly
/misc/hot*	Basic	green	&	IP	spyonly
/company/*				IP	spyonly
/update.htm	Basic	blue			

In this example, all of the files in /supersecret/, but not its subdirectories, can be accessed by any user who is either coming from the correct IP address or has a username and password for the red realm. In the second case, all files in /misc/ that start with "hot" can be accessed only by a user who has both the correct IP address and authentication for the green realm. Files in /company/, including subdirectories (as indicated by the asterisk (*)), are protected only by IP address. The single file update.ht mis protected only by authorization in the blue realm.

MAC003263

ORCL00941939

MAC003264

ORCL00941940

APPENDIX

A

Oracle WebServer Messages

This appendix contains messages you may receive if you encounter errors while using Oracle WebServer. If you get an error message, look in this appendix to find the probable cause and suggestions of what to do.

MAC003265

ORCL00941941

00001 – 00600 Generic Oracle WebServer Configuration Messages

OWS—00001 component: unable to open file name

Cause The file may not exist or may not be readable to the signalling program.

Action Verify the file's existence in the file–system. Check file and directory permissions and ownership, as well as the ownership of the signalling process.

OWS—00002 component: error reading file name

Cause Usually an operating system error. If the problem had been file permissions or non–existence, error 1 would have been signalled.

Action Check your operating system log for errors.

OWS—00003 component: error writing file name

Cause Usually an operating system error. If the problem had been file permissions or non–existence, error 1 would have been signalled.

Action Check your operating system log for errors.

OWS—00010 component: file format error when reading name

Cause The signalling program expected a specific file format which it did not find.

Action Analyze the file for incorrect formatting.

OWS—00011 Installation not completed. Root.sh must be run

Cause The signalling program checked if root.sh completed successfully.

Action Run root.sh as indicated by your installation guide.

OWS—00020 component out of memory when requesting number bytes

Cause The operating system was not able to supply the requested amount of memory.

Action Reduce the number of running programs on the machine and retry the operation.

OWS—00600 component internal error, arguments [arg1][arg2][arg3][arg4][arg5]

Cause This is the generic internal error number for Oracle WebServer program exceptions. This indicates that a process has encountered an exceptional condition.

Action Report as a bug. The first argument is the internal error number

MAC003266

ORCL00941942

05000 – 05499 Oracle Web Agent Messages

OWS—05100 Agent : unable to connect due to Oracle error number

Cause The Oracle Web Agent was unable to connect to the specified Database due to an Oracle error.

Action See the corresponding Oracle error in the Oracle7 Server Messages. Common errors are: invalid username/ password – check the Web Agent configuration file; Oracle not available – start up the database.

OWS—05101 Agent : execution failed due to Oracle error number

Action See the associated Oracle error.

OWS—05110 Agent : no stored procedure specified to call

Cause OWA requires the PATH_INFO environment variable to be set. PATH_INFO is set based on information passed through the URL after the OWA executable name and before the "?" or the end of the URL.

Action Specify a stored procedure name after the OWA executable name in the URL.

OWS—05111 Agent : no procedure matches this call with the arguments passed

Cause The procedure and parameters passed to OWA don't match any existing stored procedures.

Action Verify that there is a stored procedure (not function)with no RECORD parameters which matches the parameters passed. If no value was passed for a parameter, be sure that the stored procedure was declared to have a default value for that parameter. Check for typographical errors.

OWS—05112 Agent : too many procedures matches this call

Cause The procedure and parameters passed to OWA match too many existing stored procedures.

Action This should only happen if there are overloaded stored procedures, and that in two different versions of the procedure, a given parameter is declared in one as a scalar and the other as a PL/ SQL table.

MAC003267

ORCL00941943

OWS—05150 Agent : Web Server Error – environment variable name is NULL or non–existent

Cause The Web Server or other invoking application did not set the listed environment variable.

Action Contact your Web Server or other invoking application vendor.

OWS—05151 Agent : Port number not in list of valid ports : number list

Cause The network port that the invoking an HTTP server is listening on is not a valid port for this OWA Service.

Action Add this port to the list of valid ports for this Service, or access this service through an HTTP server listening on a valid port.

OWS—05200 Agent : service name not found in configuration file

Cause The piece of the URL just before the Web Agent executable name did not correspond to a Web Agent Service in the OWA configuration file.

Action Correct the URL, or create the desired service.

OWS—05201 Agent : service parameter name not found for service name

Cause A parameter required by the Oracle Web Agent was not found in the configuration file along with the specified service.

Action Modify the Web Agent service configuration using the Web Agent administration form.

MAC003268

ORCL00941944

05500 – 05599 Oracle Web Agent Administration Messages

OWS—05500 Service name successfully written

OWS—05501 Service name successfully modified

OWS—05502 Service name successfully deleted

OWS—05503 Password for service name successfully modified

OWS—05504 Service name successfully written. Verify the PL/SQL output in filename

Cause The Web Agent Administration program successfully wrote the specified service and performed all actions requested. The program, however, cannot detect errors which may have occurred while installing the PL/ SQL.

Action Check the output in the file specified and verify that there are not unexpected error messages.

OWS—05520 Service name not deleted because it does not exist

OWS—05521 Service name submission failed because the passwords submitted do not match

Cause The password and the confirm password field do not match.

Action Retype both passwords and re–submit the form.

OWS—05522 Service name submission failed because password identification was selected and the submitted password was NULL

Cause The form submitted indicated that the Web Agent service should use a password to connect to the database, however no password was provided.

Action Select operating system authentication for this service, or provide a password for the given OWA Database User.

OWS—05523 Service name submission failed because value for mandatory parameter name is NULL

Action Fill in the specified field and re–submit the form.

MAC003269

ORCL00941945

OWS—05524 Service name submission failed because both parameter 1 and parameter2 are NULL

Cause Both of the listed fields were NULL.

Action Fill in one of the listed fields and re-submit the form.

OWS—05525 Service name submission failed because a service with the same name already exists

Action Choose another name for the service you are attempting to create, or delete the existing service.

OWS—05526 Service name submission failed due to error number

Action See the associated error and take the appropriate action.

OWS—05527 Service name submission failed because the user type Username and Password are invalid

Cause Either the OWA Database User name and password, or the DBA username and password, is not a valid username/ password combination for the specified database.

Action Correct the username and password and re-submit the form.

OWS—05528 Service name submission failed because the DBA User submitted does not have privileges

Action Choose a different Oracle username and password for the DBA.

OWS—05529 Service name to modify does not exist

Cause The user chose to modify an existing Web Agent service, however that service does not exist. This error should not occur unless the service to modify was deleted from a different form after this Service Modification Form was generated.

Action Create a new service.

OWS—05530 Service name submission failed because the NLS Language could not be determined

Cause The Web Agent Administration package was unable to lookup the NLS Language for the specified database.

Action Specify the value explicitly or submit a valid dba username and password.

MAC003270

ORCL00941946

OWS—05531 Service name submission failed because neither Server Manager, nor SQL*DBA could be run

Cause The Web Agent Administration package was unable to install the WebServer Developer's Toolkit which must be done from Server Manager (line mode) or SQL*DBA.

Action Install either Server Manager or SQL*DBA in the ORACLE_HOME where the Web Agent administration is installed.

05600 – 05699 Oracle Web Database Administration Messages

OWS—05600 Startup of database name successful

OWS—05601 Shutdown of database name successful

OWS—05610 Startup of database name failed due to error number

Action See the associated error and take the appropriate action.

OWS—05611 Shutdown of database name failed due to error number

OWS—05620 DB Admin : no database selected

Action Return to the administration screen and select a database.

OWS—05621 DB Admin : name is already running.

Cause A request was made to start up a database, but that database is already running.

Action If you would like to restart the database, shut it down first or choose Startup Force.

OWS—05622 DB Admin : name is already shut down.

Cause A request was made to shut down a database, but that database is already shut down.

OWS—05623 DB Admin : parameter file name is larger than maximum size number bytes

Cause The initialization file listed was larger than the maximum allowable size.

Action Reduce the size of the initialization file.

MAC003271

ORCL00941947

05700 – 05799 Oracle Web Listener Configuration Messages

OWS—05700 Configuration for Web Listener on port number successfully written

OWS—05701 Configuration for Web Listener on port number successfully deleted

OWS—05710 Value submitted for parameter must not be NULL

Action Type a value for the specified parameter and re-submit the form.

OWS—05711 Port number is already in use by the Oracle Web Listener Administration Utility

Cause The port specified for Web Listener creation or modification is already in use by the Oracle WebServer Administration Utility.

Action Choose another port to run one of the Web Listeners on.

OWS—05712 An Oracle Web Listener is already configured to run on port number

Cause The port specified for service creation or modification is already in use by another Oracle Web Listener.

Action Choose another port to run one of the Web Listeners on.

OWS—05713 Value for parameter must be between minval and maxval

Action Specify a value between the two limits and resubmit the form.

OWS—05714 For ports less than number, the effective userid must be name

Cause Ports below that specified require superuser privileges to run a Web Listener on it.

Action Configure the server to run on a unrestricted port, or set the effective userid for this Web Listener the superuser.

MAC003272

ORCL00941948

OWS—05720 Web Listener on port number successfully started

OWS—05721 Web Listener on port number successfully stopped

OWS—05722 Web Listener on port number successfully stopped

OWS—05725 Web Listener on port number failed to start

Action See the associated error message.

OWS—05726 Web Listener on port number failed to reload

Action See the associated error message.

OWS—05727 Web Listener on port number was not stopped

Action See the associated error message.

05900 – 05999 Oracle Web Server Configuration Messages

OWS—05900 Web Listener on port number successfully stopped

OWS—05901 Please enter a value for field name.

Cause The specified field is a required field for registration.

Action Type a value into the specified field and re–submit the form.

MAC003273

ORCL00941949

Index

Symbols

[ ], (symbols explained), 4 – 35
& (symbol explained), 7 – 31
%ROWTYPE attribute, 4 – 19
%TYPE attribute, 4 – 15, 4 – 18
| (symbol explained), 4 – 35, 7 – 31

A

Administration Utility
  navigating to, 7 – 1
  structure of, 7 – 1
  to edit Web Listener configuration files, 3 – 11
  Web Listener security assignments using, 7 – 30
Advanced Configuration button, 7 – 12
aliases
  See also correlation variables; range variables; synonyms
  domain names for machine names, 7 – 13
  for database objects, 4 – 8
  for database tables, 4 – 10
  for paths, 7 – 22 to 7 – 27
ALTER TRIGGER statement (SQL), 4 – 37
anchors
  structure of, 2 – 13
  used in hypertext linking, 2 – 11
applications programming, PL/ SQL used for, 4 – 12 to 4 – 13
arrays, implemented as tables in PL/ SQL, 4 – 17
assignments
  initial, 4 – 15
  values to parameters (PL/ SQL), 4 – 20
  values to variables (PL/ SQL), 4 – 21
authorization schemes, 7 – 27 to 7 – 31
automatic recompilation (PL/ SQL), 4 – 31

B

Basic Authentication, 3 – 8, 3 – 17, 7 – 27, 7 – 28 to 7 – 31
Basic Configuration button, 7 – 12
blocks (PL/ SQL), 4 – 13
  See also subprograms (PL/ SQL)
Boolean logic
  IF statements and (PL/ SQL), 4 – 22
  Three-Valued in SQL, 4 – 7

C

cattributes, 6 – 5
cattributes parameter, use in passing exact text, 6 – 5
CGI, 3 – 4
CGI Application Execution, 3 – 5
CGI scripts, how flagged, 7 – 22 to 7 – 23
Change OWA Database User Password, 7 – 9

MAC003274

ORCL00941950

changes, viewing in HTML document, 2 – 16
character sets, 7 – 8 to 7 – 9
  default, 7 – 21
  international, defining, 7 – 24
columns
  See also tables; databases
  of database tables, 4 – 2
Common Gateway Interface. See CGI
compiler directives (PL/ SQL), 4 – 27
Configuration Directory, 7 – 14
  changing, 7 – 20
configuration files, Web Listener. See Web Listener, configuration file
Configure button, 7 – 12
Configure Extended Parameters Form, 7 – 16 to 7 – 31
Confirm Password, 7 – 7
connections, to the database, 4 – 4
constants, declaring (PL/ SQL), 4 – 16
constraints (SQL), 4 – 7
Copy Service button, 7 – 10
correlation variables
  in SQL statements, 4 – 10
  in triggers, 4 – 36
CREATE FUNCTION statement (SQL), 4 – 28
Create OWA Database User, 7 – 9
CREATE PACKAGE BODY statement (SQL), 4 – 32 to 4 – 33
CREATE PACKAGE statement (SQL), 4 – 31 to 4 – 32
CREATE PROCEDURE privilege (SQL), 4 – 29
  See also privileges; procedures
CREATE PROCEDURE statement (SQL), 4 – 28
CREATE TABLE statement (SQL), 4 – 7
cursor variables (PL/ SQL), 4 – 19
  See also cursors; variables
cursors (PL/ SQL), 4 – 19
  in package specifications, 4 – 31

D

Data Dictionary, 4 – 2
data structures, in PL/ SQL, 4 – 17 to 4 – 19
database links. See databases, distributed
databases
  adminstering from Administration Utility, 7 – 2
  changing data content of, 4 – 8 to 4 – 9
  connecting multiple, 4 – 5
  connecting to, 4 – 4, 7 – 4
  constraining, 4 – 7
  creating objects in using PL/ SQL, 4 – 13
  creating tables in, 4 – 7
  database triggers, 4 – 34 to 4 – 37
  distributed, 4 – 5
  interfacing to, 4 – 5
  keys used in, 4 – 2
  local, 4 – 5, 7 – 7
  missing data in, 4 – 7
  nulls used in, 4 – 7
  referencing objects in, 4 – 8
  relational, 4 – 2
  remote, 4 – 5, 7 – 7
  removing data from, 4 – 9
  retrieving data from, 4 – 6
  shutting down, 7 – 3
  specifying location of. See ORACLE_HOME
  starting up, 7 – 3
  storing code in, 4 – 13, 4 – 28 to 4 – 37
    See also stored procedures; packages; triggers
  structure of, 4 – 2 to 4 – 4
  unique identifiers in, 4 – 2
  users of, 4 – 4
datatypes
  composite, 4 – 14
  converting, 4 – 15
  inheritance of (PL/ SQL), 4 – 15
  PL/ SQL, 4 – 14
  pointer, 4 – 14
  RECORD (PL/ SQL), 4 – 18
  scalar, 4 – 14
  SQL, 4 – 7
    complex data, 4 – 4
    LONG RAW, 4 – 4
  structured, 4 – 14
  TABLE (PL/ SQL), 4 – 17
  user-defined, 4 – 16
    PL/ SQL records, 4 – 18 to 4 – 19
    PL/ SQL tables, 4 – 18

MAC003275

ORCL00941951

dblinks. See databases, distributed
DBMS. See RDBMS, databases
DBMS_SQL package, 4 – 13
DBMSSTDX.SQL script, 4 – 31
declarations
  constants (PL/ SQL), 4 – 16
  subprograms (PL/ SQL), 4 – 20 to 4 – 21
  types (PL/ SQL), 4 – 16
  variable (PL/ SQL), 4 – 15
DECLARE section (PL/ SQL), 4 – 14 to 4 – 21
Delete Service button, 7 – 10
DELETE statement (SQL), 4 – 9
Developer's Toolkit, 6 – 1
  installing, 7 – 9 to 7 – 10
  procedures and functions, 6 – 1 to 6 – 37
Digest Authentication, 3 – 8, 3 – 17, 7 – 27, 7 – 28 to 7 – 31
Directory Mappings, 7 – 22 to 7 – 23
distributed databases, 4 – 5
  See also databases
DNS Resolution, 7 – 19
Domain Name Based Restriction, 3 – 8
domain names
  as aliases for machine names, 7 – 13
  resolved to IP addresses, 7 – 19
  security based on. See Domain–based restriction
Domain–based restriction, 7 – 28, 7 – 30 to 7 – 31
dot notation
  for database objects, 4 – 8
  for PL/ SQL objects and subprograms, 4 – 16, 4 – 17

E

encoding, how indicated, 7 – 25 to 7 – 27
Error File Name, 7 – 20
error handling
  HTML, 7 – 8
  location of files, 7 – 19
  log file name, 7 – 20
  log files used for, 7 – 8
  Oracle7, 4 – 26
  PL/ SQL, 4 – 25 to 4 – 28
  return codes, 5 – 18
  system errors, 5 – 17 to 5 – 18
  triggers and, 4 – 36
exceptions (PL/ SQL), 4 – 25 to 4 – 28
  See also error handling
  declared in DECLARE section, 4 – 19
  declaring, 4 – 26
  predefined, 4 – 26
  user–defined, 4 – 27
EXECUTABLE section (PL/ SQL), 4 – 21 to 4 – 25
EXECUTE privilege (SQL), 4 – 29
  See also privileges
EXIT statement (PL/ SQL), 4 – 25

F

fields
  alternate name for database columns. See columns; databases
  in PL/ SQL records, 4 – 18
File Cache, 7 – 23 to 7 – 24
files
  caching of, 7 – 23
  encoded, 7 – 25 to 7 – 27
flow control (PL/ SQL), 4 – 21 to 4 – 25
FOR loops (PL/ SQL), 4 – 24
FOR UPDATE clause (PL/ SQL), 4 – 19
forced line breaks, 2 – 6
foreign keys, 4 – 2
  See also databases
  joins using, 4 – 10
Form tags, 2 – 22
  Input, 2 – 22
  TextArea, 2 – 22
Forms, 2 – 20
  selection menus, 2 – 24 to 2 – 26
  syntax, 2 – 21
FTP, 2 – 12
functions, 4 – 20 to 4 – 21
  See also subprograms (PL/ SQL)
  grouped into packages, 4 – 30
  return values of (PL/ SQL), 4 – 20 to 4 – 21

MAC003276

ORCL00941952

storing in database, 4 – 28 to 4 – 37

G

GOTO statement (PL/ SQL), 4 – 25
  error handling and, 4 – 28
graphics
  as hypertext buttons, 7 – 21 to 7 – 22
  within HTML documents, 2 – 14 to 2 – 16
Group ID, 7 – 15, 7 – 22

H

Host Address, 7 – 18
Host Name, 7 – 13, 7 – 18
HTF. See HyperText Functions
HTML
  basic concepts, 2 – 2
  case sensitivity, 2 – 3
  error handling, 7 – 8
  introduction, 2 – 1
  multi–button graphics, 7 – 21 to 7 – 22
HTML error page, 7 – 8
HTML Tags
  ADDRESS, 2 – 17
  BLOCKQUOTE, 2 – 7
  BODY, 2 – 4
  BOLD, 2 – 16
  CAPTION, 2 – 19
  definition list, 2 – 10
  escape sequence, 2 – 17
  getting started, 2 – 3 to 2 – 6
  HEAD, 2 – 4
  heading level, 2 – 5 to 2 – 10
  I, 2 – 16
  lists, 2 – 8
  mono spaced, 2 – 16
  nested list, 2 – 9
  ordered list, 2 – 8
  P, 2 – 5
  PRE, 2 – 5
  TABLE, 2 – 18
  TD, 2 – 18
  TH, 2 – 19
  TITLE, 2 – 4
  TR, 2 – 18
  U, 2 – 16
  unordered list, 2 – 9
HTP. See HyperText Procedures
HTTP, 2 – 12
HyperText Functions, description of, 5 – 2
HyperText Markup Language, 2 – 1
HyperText Procedures
  body tags, 6 – 11 to 6 – 16
  character format tags, 6 – 19
  definition of, 6 – 1
  description of, 5 – 2
  form tags, 6 – 23 to 6 – 30
  head–related tags, 6 – 8
  list tags, 6 – 15 to 6 – 20
  physical format tags, 6 – 22
  printing procedures, 6 – 5 to 6 – 7
  structural tags (HTML), 6 – 7 to 6 – 9
  table tags, 6 – 29 to 6 – 33

I

Identified by
  defined, 7 – 9
  radio button, 7 – 7
IF statement (PL/ SQL), 4 – 22
Image Map, 7 – 21 to 7 – 22
imagemap, 3 – 7
Info File Name, 7 – 19
Initial File, 7 – 20
initialization section (PL/ SQL packages), 4 – 33
input parameters. See parameters, modes of (PL/ SQL)
INSERT statement (SQL), 4 – 9
Internet Protocol based Restriction, 3 – 8
IP addresses
  Domain Name Resolution of, 7 – 19
  security based on. See IP–based restriction
  Web Listener, 7 – 18
IP–based restriction, 3 – 18, 7 – 27 to 7 – 28, 7 – 29 to 7 – 31
ISINDEX, 3 – 5, 3 – 6
ISMAP attribute, 2 – 14

MAC003277

ORCL00941953

J

joins, 4 – 10 to 4 – 11
  natural, 4 – 10
  outer, 4 – 10

L

labels, PL/ SQL, 4 – 25
languages
  See also character sets; SQL; PL/ SQL
  associating with character sets, 7 – 24 to 7 – 25
  default, 7 – 21
links
  to other documents, 2 – 11
  to other sections, 2 – 15
list tags
  definition, 2 – 10
  nested, 2 – 9
  ordered, 2 – 8
  types of, 2 – 8
  unordered, 2 – 9
Log File Directory, 7 – 8, 7 – 19
log files
  default, 7 – 8
  Web Agent Service, 7 – 8
loops (PL/ SQL), 4 – 23
  exiting nested, 4 – 25
  FOR, 4 – 24
  WHILE, 4 – 24

M

mailto, 2 – 12
Max Connect Count, 7 – 19
MIME (Multimedia Internet Mail Extension) types
  default, 7 – 21
  defining, 7 – 25
  further information, 7 – 25
Modify Service button, 7 – 10
multimedia data
  file types of, 7 – 25
  in Oracle7 database, 4 – 4
  MIME types, 7 – 21

N

naming conventions
  PL/ SQL, 4 – 17
    tables, 4 – 18
  SQL, 4 – 8
natural joins, 4 – 10
network communications, 4 – 5
  port selection, 7 – 8
NLS Language, 7 – 8 to 7 – 9
NOT NULL constraint (SQL), 4 – 7
  See also nulls; constraints
  PL/ SQL use of, 4 – 15
NULL statement (PL/ SQL), 4 – 23
nulls
  as Boolean values, 4 – 7
  database contains, 4 – 7
  differentiated from NULL statement, 4 – 23
  outer joins generate, 4 – 10
  prohibiting, 4 – 7, 4 – 15
  SQL handling of, 4 – 7
  UPDATE statement (SQL) generates, 4 – 9
  variables initialized to (PL/ SQL), 4 – 15

O

objects
  aliases for database objects. See correlation variables; synonyms
  ownership of database, 4 – 8
    See also databases
Oracle Web Agent, description of, 5 – 2 to 5 – 4
ORACLE_HOME
  default log file directory in, 7 – 8
  Web Agent Service uses, 7 – 7
ORACLE_SID, 7 – 7
Oracle7 Server, 4 – 1
  See also databases
  stores multimedia data, 4 – 4
outer joins, 4 – 10
output parameters. See parameters, modes of (PL/ SQL)

MAC003278

ORCL00941954

owa, calling procedures with, 7 – 4
OWA Database User, 7 – 7
OWA Log Directory, defines Web Agent error log, 5 – 19
OWA Service, 7 – 6
OWA Service Name, defines Web Agent error log, 5 – 19
OWA User Passord, 7 – 7
OWA Utilities package, description of, 6 – 2
owa.cfg
  See also Web Listener configuration file
  sample with parameters, 5 – 6 to 5 – 8
  Web Agent uses, 5 – 4
owa.err, 7 – 8
  See also error handling
OWA_DEFAULT_SERVICE, 7 – 8
  See also Web Agent Services
  deleted services and, 7 – 10
OWA_UTIL Package, 6 – 31 to 6 – 36
OWA_UTIL package, definition of, 5 – 2
OWAINS.SQL sql script, using to install Developer's Toolkit, 6 – 3

P

packages, 4 – 30 to 4 – 34
  bodies, 4 – 32
  creating, 4 – 31
  specification, 4 – 31
parameters
  actual (PL/ SQL), 4 – 20
  formal (PL/ SQL), 4 – 20
  in PL/ SQL, 4 – 20
  in PL/ SQL packages, 4 – 33
  log file, 7 – 19 to 7 – 20
  modes of (PL/ SQL), 4 – 20
  network, 7 – 18 to 7 – 19
  Web Agent Service, 7 – 5 to 7 – 9
parent keys, 4 – 3
  See also primary keys; foreign keys; databases
passwords
  changing, 7 – 9
  database, 4 – 4
  Web Agent Service uses, 7 – 7, 7 – 10 to 7 – 11
  Web Listener, 7 – 28
PID (Process Identifier), Web Listener, 7 – 19
PL/ SQL, 4 – 12 to 4 – 37
  blocks, 4 – 13
  declarations, 4 – 14 to 4 – 21
  DECLARE section, 4 – 14 to 4 – 21
  EXCEPTION section, 4 – 25 to 4 – 28
  EXECUTABLE section, 4 – 21 to 4 – 25
  flow control in, 4 – 21 to 4 – 25
  general discussion of, 4 – 12 to 4 – 13
  generating dynamically, 4 – 13
  GOTO restrictions, 4 – 25
  overloading subprograms, 4 – 33
  scope of objects and subprograms, 4 – 16
  sections of, 4 – 13
  subprograms, 4 – 13
  user–defined datatypes in, 4 – 16
Port Number, 7 – 13, 7 – 18
ports
  changing to copy Web Listeners, 7 – 17
  reserved, 7 – 18
  selecting, 7 – 8
  Web Listeners associated with, 7 – 13, 7 – 18
  Web Listeners identified by, 7 – 11
pragmas (PL/ SQL), 4 – 27
predicates
  in IF statements (PL/ SQL), 4 – 22
  in SQL, 4 – 6
    See also Three–Valued Logic
primary keys, 4 – 2
  enforcing, 4 – 7
private synonyms, 4 – 8
  See also synonyms
privileges
  See also roles; profiles; users
  object, 4 – 4
  required to create triggers, 4 – 36
  stored procedures require, 4 – 29
  stored procedures use, 4 – 29
  system, 4 – 4
procedures
  See also subprograms (PL/ SQL)
  declaring, 4 – 20 to 4 – 21
  grouped into packages, 4 – 30
  storing in database, 4 – 28 to 4 – 37
procedures and functions, getting started, 6 – 4

MAC003279

ORCL00941955

profiles, 4 – 5
  See also users, database
public synonyms, 4 – 8
  See also synonyms
  for WebServer paths. See virtual paths

Q

queries (SQL), 4 – 6
  multiple tables used in, 4 – 10 to 4 – 11
  stored in cursors (PL/ SQL), 4 – 19
  within other queries (subqueries), 4 – 11
query string, 3 – 5
QUERY_STRING environmental variable,
    passing parameters with, 5 – 10

R

RAISE statement (PL/ SQL), 4 – 27
  See also error handling
range variables, 4 – 10
  See also correlation variables
RDBMS, 4 – 1
  See also Databases
realms
  assigning, 7 – 31
  defined, 7 – 28 to 7 – 29
records
  alternate name for database rows. See rows;
      databases
  PL/ SQL datatype, 4 – 18
ref cursors (PL/ SQL), 4 – 14
  See also cursors
referenced keys, 4 – 3
referential integrity, 4 – 3
relational databases, 4 – 2
  See also databases
reload, 2 – 16
REQUEST_METHOD environmental variable,
    passing parameters with, 5 – 9
restriction schemes, 7 – 27 to 7 – 31
RETURN statement (PL/ SQL), 4 – 20 to 4 – 21
return values
  cursor (PL/ SQL packages), 4 – 31
  functions (PL/ SQL), 4 – 20 to 4 – 21
roles, 4 – 5
  See also privileges
rows
  See also tables; databases
  of database tables, 4 – 2

S

schemas, 4 – 8
  See also databases; users
scope
  of PL/ SQL objects and subprograms, 4 – 16
  PL/ SQL packages and, 4 – 33
SCRIPT_NAME variable, how Web agent uses,
    5 – 4
scripts, 3 – 5
security
  types of, 7 – 27 to 7 – 28
  Web Listener, 7 – 25 to 7 – 31
    assignments, 7 – 30 to 7 – 31
SELECT statement (SQL), 4 – 6
  See also queries (SQL)
servers
  database, 4 – 1
    See also databases; Oracle7 server
  web
    generic. See HTTP servers
    Oracle. See Web Listeners
service_name.err, Web Agent error log, 5 – 19
services
  SQL*Net V2, 7 – 8
  Web Agent. See Web Agent Services
shutdown, Web Listener, 7 – 12
shutdown mode (database), 7 – 3
SQL, 4 – 5 to 4 – 12
  building applications using, 4 – 12 to 4 – 13
  dynamic, 4 – 13
  predicates, 4 – 6
  standards, 4 – 5
  Three–Valued Logic (TRUE, FALSE, NULL)
      in, 4 – 7
SQL*Net, 4 – 5
  V2 service specfication, 7 – 8
SQLCODE, 4 – 26

MAC003280

ORCL00941956

SQLERRM, 4 – 26
STANDARD package (PL/ SQL), 4 – 26
Start button, 7 – 12
startup, Web Listener, 7 – 12
startup mode (database), 7 – 3
Stop button, 7 – 12
stored procedures, 4 – 28 to 4 – 34
  invoking in URL, 7 – 4
Structured Query Language. See SQL
Submit New Service button, 7 – 9
subprograms (PL/ SQL), 4 – 13
  declaring, 4 – 20 to 4 – 21
  error handling and, 4 – 27
  GOTO restrictions, 4 – 25
  overloading, 4 – 33
  parameters of, 4 – 17
  resolving ambiguous references, 4 – 17
  scope and visibility of, 4 – 16
subqueries (SQL), 4 – 11
subtypes (PL/ SQL), 4 – 16
  See also datatypes
  using to create PL/ SQL records, 4 – 18 to 4 – 19
  using to create PL/ SQL tables, 4 – 18
synonyms, for database objects, 4 – 8

T

Table attributes
  Align, 2 – 19
  Border, 2 – 19
  Colspan, 2 – 19
  Colspec, 2 – 20
  Nowrap, 2 – 19
  Rowspan, 2 – 20
  Valign, 2 – 19
Tables, in HTML, 2 – 18 to 2 – 20
tables
  database, 4 – 2, 4 – 4
    See also databases
    aliases for, 4 – 8
      See also range variables; synonyms
    creating, 4 – 7
    stored in cursors (PL/ SQL), 4 – 19
  HTML, 4 – 4
  ownership of, 4 – 8
  passing as parameters in PL/ SQL, 4 – 17
  PL/ SQL, referencing, 4 – 18
  PL/ SQL datatype, 4 – 17 to 4 – 18
text–only browsers, 2 – 14
Three–Valued Logic (TRUE, FALSE, NULL), 4 – 7
triggers, 4 – 34 to 4 – 37
  altering, 4 – 35 to 4 – 36
  creating, 4 – 35 to 4 – 36
  enabling and disabling, 4 – 37

U

Uniform Resource Locator (URL), 2 – 11, 2 – 13, 2 – 14, 3 – 5
unique identifiers
  See also primary keys
  database use of, 4 – 2
  in database. See naming conventions
unique keys, 4 – 3
UPDATE statement (SQL), 4 – 9
URL, 2 – 11
  See also Uniform Resource Locator (URL)
  Web Agent Service specified in, 7 – 4
URL encoding, 3 – 6
User Directory, 7 – 20
User Directory Initial File, 7 – 20
User ID, 7 – 15, 7 – 22
  Web Listener, 7 – 15, 7 – 22
users
  assigning to realms, 7 – 28
  assigning Web Listener security schemes to, 7 – 30
  authentication used for, 7 – 28
  database, 4 – 4
    controlling actions of, 4 – 4
    creating, 7 – 9
    own objects, 4 – 4
    own schemas, 4 – 8
    privileges and, 4 – 4
    privileges of, 7 – 8
      See also privileges
    use of resources, 4 – 5
    use roles, 4 – 5

MAC003281

ORCL00941957

directories used for, 7 – 20
domain name identification of, 7 – 30
IP identification of, 7 – 29 to 7 – 30
Oracle7, 4 – 4
OS authentication of, 7 – 7
Web Agent Services emulate, 7 – 7
Web Listener security assignments, 7 – 30 to 7 – 31

V

variables
assigning values to (PL/ SQL), 4 – 21
cursor, 4 – 14, 4 – 19
datatypes of. See datatypes
declaring (PL/ SQL), 4 – 15
in triggers, 4 – 36
initializing, in PL/ SQL packages, 4 – 33
instantiation in PL/ SQL packages, 4 – 31
virtual paths, 7 – 22 to 7 – 23
file caching uses, 7 – 24
security schemes use, 7 – 26
Web Agent Services use, 7 – 9
visibility
of PL/ SQL objects and subprograms, 4 – 16
PL/ SQL packages and, 4 – 33

W

Web Agent
CGI environment variables, 5 – 16
components of, 5 – 2
creating customized error pages, 5 – 17
Creation form (HTML), 5 – 6
error handling, 5 – 17 to 5 – 19
error log, 5 – 19
how it works, 5 – 3 to 5 – 7
optimizing multiple services, 6 – 3
overview, 5 – 1
request methods, 5 – 9
users of, 5 – 2
version number, 5 – 19
Web Agent error handling, application errors, 5 – 17
Web Agent Modification form, 7 – 9 to 7 – 10
Web Agent Service, 5 – 5 to 5 – 36
Web Agent Services
activating, 7 – 9
administering, 7 – 4 to 7 – 11
creating, 7 – 5, 7 – 9
deleting, 7 – 9
modifying, 7 – 5, 7 – 9
securing, 7 – 10 to 7 – 11
URL specifies, 7 – 4
Web Listener configuration file, 3 – 3
parameters in, 3 – 11 to 3 – 20
Web Listener Control Utility, 3 – 10 to 3 – 12
Web Listener Creation Form, 7 – 12
Web Listener PID, 7 – 19
Web Listener root directory, 7 – 14
Web Listeners
administering, 7 – 11 to 7 – 31
ascertaining state of, 7 – 12
associating with machines, 7 – 13, 7 – 18
associating with Web Agent Services, 7 – 9
configuration files, 7 – 16
configuration parameters for, 3 – 10
configuring, 7 – 16 to 7 – 31
connections to, maximum, 7 – 19
copying, 7 – 17
creating, 7 – 11
default directories, 7 – 20
deleting, 7 – 17 to 7 – 18
directory mapping, 3 – 3
directory mapping parameters, 3 – 13
DNS resolution, 3 – 7
encoding definitions, 3 – 14
error handling, 7 – 19 to 7 – 20
features, 3 – 2
file cache definitions, 3 – 13
file caching, 3 – 2, 7 – 23
group ID, 7 – 15, 7 – 22
HTTP protocol, 3 – 4
imagemap support, 3 – 7
international characters sets, 7 – 24 to 7 – 25
IP addresses, 7 – 18
language extensions, 3 – 3, 3 – 15
logging, 3 – 9
logging parameters, 3 – 12
memory mapping, 3 – 2
mime type definitions, 3 – 15
modifying, 7 – 12, 7 – 17
multimedia data, 7 – 25

MAC003282

ORCL00941958

overview, 3 – 1
parameters, 7 – 20 to 7 – 21
password protect Web Agent Services, 7 – 10
process identifiers (PIDs), 7 – 19
reinitializing, 7 – 16
reloading with WLCTL, 3 – 10
root directory of, 7 – 14 to 7 – 15
security, 3 – 8 to 3 – 10, 7 – 22, 7 – 25 to 7 – 31
  basic authentication, 7 – 28 to 7 – 31
  digest authentication, 7 – 28 to 7 – 31
  domain–based restriction, 7 – 30 to 7 – 31
  encoding, 7 – 25 to 7 – 27
  IP–based restriction, 7 – 29 to 7 – 31
  types of, 7 – 27 to 7 – 28
security for individual files, 3 – 18
  starting, 7 – 16
  starting with WLCTL, 3 – 10
  stopping with WLCTL, 3 – 10
  tracking information, 7 – 19
web servers
  generic. See HTTP servers
  Oracle. See Web Listeners
WebServer, administration, 7 – 19 to 7 – 20
WebServer Developer's Toolkit
  installing, 6 – 2
  security considerations, 6 – 4
  using OWAINS.SQL to install, 6 – 3
WHILE loops (PL/ SQL), 4 – 24
WLCTL. See Web Listener Control Utility

MAC003283

ORCL00941959

Reader's Comment Form

Name of Document: Oracle® WebServer User's Guide
Part No. A34986–2

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 topic, chapter, and page number below:

_____

_____

_____

_____

_____

_____

_____

_____

Please send your comments to:

Oracle Internet Products Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood City, CA 94065 U.S.A.
Fax: (415) 506–7200

If you would like a reply, please give your name, address, and telephone number below:

_____

_____

_____

Thank you for helping us improve our documentation.

MAC003284

Exhibit 6

The Oracle WebServer: A Technical Discussion

Katrina Montinola, Oracle Corporation

Although the Internet has existed for over 30 years, it was generally regarded as the exclusive playground of the military, research and academic worlds until very recently, when the World Wide Web and NCSA Mosaic emerged. The World Wide Web is a set of techniques, protocols, technology and software that greatly simplify access to the vast amount of information available on the Internet. Together with NCSA Mosaic, the first graphical interface to the World Wide Web, it sparked the incredible growth of the Internet because it made navigating through the Internet as simple as pointing and clicking with a mouse.

The Challenge of Managing Web-based Information

The exponential growth of the World Wide Web has not gone unnoticed in the corporate world. Corporations have started to realize its potential for information dissemination and electronic commerce and are scrambling to set up Web servers. Unfortunately, while putting up a pilot Web server is a relatively simple task, a number of challenges arise as these sites grow and start to be deployed in large-scale commercial projects. For instance, how does one store, manipulate, manage and retrieve large volumes of data? How does one provide secure access to information without compromise when there are 20 million potential users of this information? As corporations conduct electronic commerce over the Internet, how does one perform secure transactions and store results? As Web sites expand, how does one keep up with this growth and prevent their site from becoming an administrative nightmare?

The Oracle WebServer

Oracle is the world's largest information management company. We have been solving similar problems for corporate data since the early 1980s, before the World Wide Web was even conceived. Since Web-based data has a lot in common with corporate data, we feel that the Oracle database can be just as effective at solving these problems. To address this need, we have developed the Oracle WebServer.

The Oracle WebServer is a World Wide Web server with a tightly-integrated Oracle7 database. It is targeted at users whose Web sites have grown too large and too complex to be managed effectively using traditional servers such as the NCSA and CERN servers. It consists of the following components: the Oracle Web Listener, the Oracle Web Agent, and the Oracle7 database. The Oracle WebServer provides a number of benefits beyond those of the traditional Web servers in many areas, the most important of which are:

Easy Access to Oracle Data - There is a large amount of data stored in Oracle databases that customers want to make available via the World Wide Web. Realizing this, we have provided a mechanism by which data stored in an Oracle7 database can be easily and efficiently transformed into HTML documents for distribution via the World Wide Web. This feature, called the Oracle Web Agent, will be available in Release 1.0 of the Oracle WebServer. High-Performance, Scalable, Portable, Commercial-quality Web Server - The Oracle WebServer will include the Oracle Web Listener, a full-fledged HTTP server engine that implements version 1.0 of the protocol as described in the RFC from the IETF HTTP Working Group. The server engine is designed to be a very reliable and high-performance solution suitable for high-traffic and mission-critical applications. Utilization of the Database for All Aspects of Web Server Management - In Release 2.0, which we have been developing in parallel with Release 1.0, a key differentiating factor between the Oracle WebServer and traditional Web servers will be the ability to store Web objects, such as HTML documents and GIF files, or pointers to Web objects, in the

ORCL01042968

database. Besides storing Web objects, we will also store the access log information and error log information in the database. Once the Web objects and other Web-related information are inside the database, Oracle's expertise in managing information can be put to work.

Although this paper will focus on providing technical information on the Oracle Web Listener and the Oracle Web Agent, two components of the Oracle WebServer Release 1.0, it will also touch briefly on Release 2.0 to give the reader a peek into the future of Web technology as we at Oracle see it.

The Oracle Web Listener

Why use a commercial Web server when free ones are available from CERN and NCSA? This is a question that anyone planning to put up a Web site is sure to ask at some stage in the planning process. The most obvious reason: a commercial site that is running a mission-critical application needs a commercially-supported Web server because any bugs encountered in the server need to be fixed in a reasonable amount of time.

Of course, the decision to go with a commercial Web server begs the question: which commercial Web server is most suitable to the application? In most cases, this can only be answered by looking closely at the architecture of the Web servers under consideration. This is exactly what we did when we were shopping around for a third-party commercial Web server to include in the Oracle WebServer. The server we ultimately chose met our criteria for high-performance, scalability, portability and manageability.

For maximum performance and scalability, the Oracle Web Listener is designed to run as a single process with a single thread, as opposed to most servers which start a new thread or process whenever a new connection is made. This design improves performance in two ways: First, there is less overhead at the operating system level because the need for task switching and memory swapping is minimized. Second, the act of starting a new process or thread is time consuming and can significantly affect the latency between when a connection is established and when the server can begin to manipulate the data.

Oracle has always been known for the portability of its products, so this was a key feature for us. The Web server we ultimately chose was designed with portability in mind. Over 75% of the source code in the Oracle Web Listener is shared between the Windows NT and UNIX versions. The code was also designed to be easily portable to Windows95, should we decide to release a version on that platform in the future.

Since we are targeting large, mission-critical Web sites, manageability was also an important factor. The Oracle Web Listener will include Simple Network Management Protocol (SNMP) support. In the first release, this will be limited to basic "I'm alive" messages. However, we will implement the different portions of the Management Image Base (MIB) for HTTP servers as it gets standardized.

The Oracle Web Agent

With approximately 35% of the relational database market, there is a huge amount of data stored in Oracle databases today. A significant percentage of this data would be even more useful and valuable if it could be accessed via the World Wide Web. To assist our customers with this task, we have developed the Oracle Web Agent. The Web Agent is a component of the Oracle WebServer that facilitates the creation of dynamic HTML documents via a language already familiar to most Oracle users: PL/SQL.

In addition to existing corporate data, the Oracle database could also be used to store data meant

ORCL01042969

exclusively for the Web. If this data resides in tables in a relational database and simply gets transformed into dynamic HTML documents when accessed, it will be much simpler to manage and manipulate. Furthermore, dynamic documents provide a greater degree of customization, making it possible to tailor a document to the specific user requesting it. The Oracle Web Agent can be used to create these dynamic HTML documents quickly and easily in PL/SQL.

The Oracle Web Agent consists of the following:

A "C" program named owa which serves as the actual CGI program that the Web Listener executes when a POST or GET request is received for a dynamic document. This program makes OCI calls to log into the database and execute the user's PL/SQL stored procedures.

A PL/SQL package called wow which contains utility functions to set up the CGI environment variables, obtain the value of a specific environment variable, and extract the output generated by the user's PL/SQL code from a PL/SQL table.

An optional, but strongly recommended set of PL/SQL packages known as the WebServer Developer's Toolkit that ease the generation of HTML tags from within the user's PL/SQL code. HTP and HTF, which are two of the packages in this toolkit, automatically generate the HTML tags and put the document into the PL/SQL table where it gets retrieved by the OCI program called owa.

A set of HTML forms and corresponding code to ease the task of configuring the CGI applications written in PL/SQL with the Web Agent.

A PL/SQL development environment which allows the programmer to compile and submit code into the database directly from within an HTML form. This is a Release 2.0 feature.

The Oracle Web Agent provides an easy-to-use environment for building CGI applications in PL/SQL, which are then stored in the Oracle7 database. A programmer who has chosen PL/SQL as the language in which to implement a CGI application need only worry about implementing the logic specific to the application itself, because the Oracle Web Agent provides utilities which take care of the repetitive tasks associated with deploying a CGI application.

Why chose PL/SQL in the first place? Two reasons: First, PL/SQL is compiled and stored inside the database, thus PL/SQL code tends to execute faster than regular SQL code that needs to be parsed and interpreted before executing. Second, since PL/SQL code resides inside the database, it is completely portable to any platform that Oracle7 runs on. Porting PL/SQL code is simply a matter of moving the PL/SQL packages from one database to another in the same way that moving regular database data is accomplished. Step 2 Step 3 Step 5 Step 4 Step 1 Step 6

Web client

The figure above illustrates the sequence of events that take place in the Oracle WebServer when a typical GET request is received for a dynamic HTML document produced via the Oracle Web Agent. .The Web client issues a GET request for an HTML document which is actually a dynamic document produced by PL/SQL code residing in the database. .The Web Listener spawns a "C" stub program called owa. .owa makes OCI calls that log into the database and execute the PL/SQL procedure which produces the HTML document. It knows which PL/SQL procedure to execute because the name of the PL/SQL procedure is embedded in the URL. .owa takes the HTML document produced by the PL/SQL procedure from a PL/SQL table where the document has been placed by the HTP/HTF calls made by the

PL/SQL procedure. .owa passes it back to the Web Listener via standard output. .The Web Listener passes it back to the client via the HTTP protocol.

To get a better technical understanding of the Oracle Web Agent, let's examine a number of it's key features in greater detail.

I. CGI Stub

Since PL/SQL procedures are stored inside the database, there is no way for a Web server to directly execute them when the request from the browser is received. Furthermore, there is no way for the PL/SQL code to send the HTML document generated to standard output, which is where the Web server is expecting it to be. Thus, there is a need for a program which handles these tasks. The Oracle Web Agent includes a "C" program called owa which does the following using OCI calls : .Extracts the database service name embedded in the URL and logs on to the database using the userid/password defined by the service. Database services are defined in a file called owa.ora. .Sets up the parameters passed from the Web server so that they are accessible from within the database through a PL/SQL function call similar to getenv() in UNIX. .Executes the user's PL/SQL function which extracts the relevant data from the database and formats it into an HTML document that is stored in a PL/SQL table. .Checks the return code of the user's PL/SQL function. If it is zero, it takes the formatted HTML document from the PL/SQL table and passes it back to the web server via standard output. If it is non-zero, it returns the standard error HTML document supplied by the user or a default if the user doesn't supply one.

Thus, it is not necessary for the PL/SQL programmer to be intimately familiar with how CGI works. owa takes care of all those details and allows the programmer to concentrate on developing the logic to extract the proper data from the database. The user's PL/SQL code need only make function calls to extract the values of the necessary CGI environment variables, query the database, and format the results returned from the database into an HTML document. Everything else is taken care of by owa.

Both the database service name and the user's PL/SQL function name are embedded in the URL, which is of the following format:

http://myhostname:port/.../myservicename/owa/myplsqlname?parameterlist

URL section Description

http	denotes the protocol used to obtain the document from the WebServer
myhost	denotes the hostname or IP address of the machine the WebServer resides in
port	an optional parameter, this specifies the TCP port number that the Web Listener is listening on. If none is specified, it defaults to 80
myservicename	denotes the database service name to use when logging on to Oracle7. The database service name immediately precedes 'owa' but doesn't have to directly follow the hostname/port section of the URL.
owa	must always exist. This instructs the Web Listener to spawn the owa executable
myplsqlname	denotes the name of the PL/SQL procedure that gets spawned when owa logs on to Oracle7
parameterlist	If a '?' exists, any text following it will be passed by owa to the PL/SQL procedure it just invoked. This will only exist if the GET method is used.

II. HTML tags

HTML documents are generated by inserting the necessary HTML tags around the text output which Web browsers can interpret. The HTP/HTF PL/SQL packages which are part of the Oracle WebServer Developer's Toolkit greatly simplify this task. The PL/SQL programmer need only call these functions at the appropriate place in the code with the actual text as a parameter. Then, the HTP (HyperText Procedure) and corresponding HTF (HyperText Function) take care of generating the correct tags around the text and putting it into a PL/SQL table.

It is important to note that the HTP/HTF packages do not eliminate the need for the PL/SQL programmer to know HTML syntax. The programmer must still realize that an anchor tag is needed to create a hyperlink, for example. What the HTP/HTF packages do is automate the process of creating these tags and putting the document in the PL/SQL table.

III. Development Environment

PL/SQL code is typically developed using SQL*Plus or an embedded development environment in tools such as Oracle Forms 4.5. Since CGI application programmers may not have access to these tools or may not be familiar with them, the Oracle Web Agent will provide a development environment accessible via a Web browser. The Oracle Web Agent will include an HTML form which will allow programmers to type in PL/SQL code. The form will have a push button to compile the code and submit it into the database. If there are errors in the compilation phase, these will be returned to the browser as an HTML document. This is a Release 2.0 feature.

IV. Error Handling

A CGI application written in PL/SQL can generate two broad types of errors: application errors and system errors. Application errors are specific to the application, and are generally meaningful to the end user. For example, if the query returns no rows, the PL/SQL programmer could add code to generate an HTML document that said, "The item you ordered is temporarily out of stock. Please try again later." These errors are transparent to owa because owa does not read HTML documents to determine their content. As far as owa is concerned, if the user's PL/SQL code generates an HTML document, the operation was successful. owa determines whether an HTML document has been generated by checking the return code from the original call to execute the user's PL/SQL program. If the return code is zero, an HTML document has been generated in the PL/SQL table.

System errors are errors encountered by owa itself. These are errors that prevent owa from executing the user's PL/SQL code, or failure of the user's PL/SQL code to generate an HTML document. A user's PL/SQL code signals it's failure to generate an HTML document by returning an error to owa.

An example of the first type of system error would be the failure of owa to log on to the database. An example of the second type of system error would be an error in the SQL query because the table does not exist. In the latter case, the programmer could write code to format an HTML document with the specific reason for the error, but it would be more convenient for the programmer if there were a standard HTML document that could be returned for errors such as these. The Oracle Web Agent provides a mechanism for doing just that. To avail himself or herself of this mechanism, the programmer returns an error to owa (see Section I, step 4). owa will either return a default error message, or return a predefined error message when it gets the non-zero return code.

The Oracle WebServer: The Future

ORCL01042972

Although the Oracle Web Agent greatly simplifies the development of CGI applications that need access to data stored in an Oracle7 database, we feel that Web servers can benefit from even tighter integration with the database. By this we mean the following:

Direct access to PL/SQL stored procedures

Although CGI works pretty well for small Web servers with relatively light loads, it doesn't provide adequate performance and doesn't scale well for Web servers that are heavily accessed. This is because CGI requires the Web server to spawn a separate process for each document requested. When Web servers start experiencing hundreds of simultaneous requests, the number of spawned processes can easily bog down the machine. Realizing this, in Release 2.0 of the Oracle WebServer, we will remove the Oracle Web Agent's reliance on CGI. owa will no longer be a separate "C" executable that gets spawned by the Web server. It will be linked directly into the Web server and will pass on the name of the PL/SQL procedure to be executed to a configurable number of processes that are always connected to the Oracle7 database. Besides providing a much more scaleable solution, this method will also enjoy increased performance, because spawning a process and logging on to the Oracle7 database are some of the slowest operations in a typical transaction.

It is important to note, however, that we are not removing the CGI interface altogether. We realize that it is important to continue to support CGI because it is an open standard that will allow our Web server to support any application written for any other Web server that supports this standard. We are merely eliminating the need to go through CGI for applications written with the Oracle Web Agent to provide increased performance and scalability.

Replacing the File System Itself with the Database

Anything that a traditional Web server would use the file system for, the Oracle WebServer Release 2.0 will use the database. This will include storage of static HTML documents and other Web objects, storage of access log information, and storage of error log information. The benefits of storing logging information in the database are immediately apparent: it will be possible to formulate much more sophisticated queries using SQL to generate reports on the actual usage and hit rate of the Web server. However, the benefits of storing actual Web objects in the database are not as obvious. Thus, I'd like to focus on this area for the rest of this section.

Although the Oracle Web Listener, which is a component of the Oracle WebServer, is a full-fledged, industrial-strength HTTP server that can serve HTML documents off the file system like any other Web server, most of the benefits of the Oracle WebServer will not be realized until the database is utilized to store Web objects. Storing Web objects in the database has the following advantages:

Since a list of objects or the objects themselves reside in the database, it will be possible to use the database to enforce additional security using the concept of database roles. The web server administrator will be able to restrict groups of objects to certain users and have these users administer their own groups. The designated administrator for a particular group will be able to add users and objects to the group, but will not be able to tamper with objects in a group with a higher privilege. This allows a finer granularity of security than is available on traditional web servers.

It will be possible to write SQL and PL/SQL code to retrieve information about the objects in the Web server. This opens up a whole range of possibilities such as the ability to maintain different versions of objects much more easily, the ability to locate specific objects more easily, etc.

ORCL01042973

Searching and retrieving Web objects based on their content will be possible. This is slow and quite cumbersome to do with flat files.

Realizing that for large Web sites it will be a big task to load all existing documents into the database, we are providing the option to leave the documents in the file system and simply create pointers to them in the database. Creating pointers in the database will still bring you the first two advantages described above. Furthermore, for extremely large documents, storing them in the file system with pointers in the database may be preferable from a performance standpoint. Fetching a 10 megabyte document from the database, for example, could be significantly slower than fetching it from the file system. For smaller documents, this performance hit is small enough that the ability to do sophisticated searches based on the content of the document and increased security would outweigh the performance degradation.

Integration with Oracle ConText

The World Wide Web's greatest asset is the wealth of information available to anyone at the click of a mouse button. However, the Web may be the classic case of "too much of a good thing" because it is becoming increasingly difficult to find the information needed due to the sheer volume of information available. Oracle ConText, which will be integrated into the Oracle WebServer, is the language-processing core of Oracle's solution to this information overload.

ConText is a natural language technology that uses the linguistic information contained in every word, sentence, and paragraph to identify the themes and content of any text. With this knowledge, ConText can facilitate automatic classification, visual browsing, and easy management of ever-increasing volumes of electronic information. Its analysis delivers insights into documents that statistical methods cannot:

Main themes representing a selection of text form a "document fingerprint" for classifying or retrieving.

Prominent ideas can be highlighted while details are hidden, for summaries or abstracts.

Areas of text discussing similar themes can be identified, collapsed, or indexed, for hyperlink navigation or finding information quickly.

ConText is unique in that it appears to understand the English language: it can distinguish between a document that merely contains the term "Computer Science" 100 times in the document, and a document that has Computer Science as its main theme, for example. The actual words, "Computer Science", may not even appear anywhere in the latter document.

This capability is extremely useful for large Web sites that want to be able to automatically classify their documents to assist users in finding what they need quickly and easily. After all, information is only valuable if one can find it when it is needed.

Conclusion

We at Oracle believe that in the future, it will be difficult to envision a commercial World Wide Web site without a backend database to manage Web content as well as other information about the Web site, such as access statistics. The Oracle WebServer is only the first product from Oracle aimed at addressing this need. Stay tuned for more!

ORCL01042974

Exhibit 7

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 8

Oracle WebServer 2.0

Technical Note
March 1996

ORACLE®
Enabling the Information Age™

Oracle WebServer 2.0

This Technical Note is one of a series of publications from Oracle Internet Server Products, a product group within the Server Technologies Division of Oracle Corporation. This document is written for web software developers and other audiences familiar with basic Web technology. Specifically, no introduction to the Internet or Web will be given here. By reading this document, you will gain a technical understanding of how Oracle WebServer 2.0 is designed, and how you can use it to write high performance applications for the web. Many acronyms and technical terms are used in this document. For your convenience, we have included a glossary in appendix A.

Overview

Oracle WebServer 2.0 lets you publish hypermedia documents and deploy media-rich applications on the World Wide Web and corporate intranets using the industry standard HTTP protocol. Unlike mere HTTP servers, Oracle WebServer 2.0 offers a fully integrated platform for application development, mass-storage, and system and site-management, all using fully integrated state-of-the-art relational database technology and established industry standards such as HTTP, SSL, and HTML. Oracle WebServer 2.0 offers an open, extensible application development platform for the Web, enabling software developers to safely and easily add high-speed, custom server extensions to take full advantage of any and all modern technologies such as Java, distributed objects and virtual reality to name a few, without having to resort to the low-level network interfaces employed by most other HTTP servers today.

An Application Platform for the Web

Oracle WebServer 2.0 provides an open platform for developing web applications. For customers with heavy investments in the Oracle7 RDBMS, we offer the integrated PL/SQL Agent, an extension to Oracle WebServer 2.0, which allows you to develop complete web applications using Oracle7 stored procedures. For customers who wish to use Java, we offer a fully integrated Java Interpreter, enabling you to develop complete web applications using server-side Java classes. The Java Interpreter includes classes for native Oracle7 access via PL/SQL and dynamic creation of HTML documents. For customers who wish to employ their own programming language of choice, Oracle WebServer 2.0 offers an easy-to-use API allowing you to build your own

server-side plugins ranging from existing legacy applications written in COBOL to modern CORBA or COM/OLE based object-oriented systems written in for instance C++ or Smalltalk. Oracle WebServer 2.0 allows developers to easily integrate their back-end applications using our open API to the Oracle Web Request Broker.

Architecture

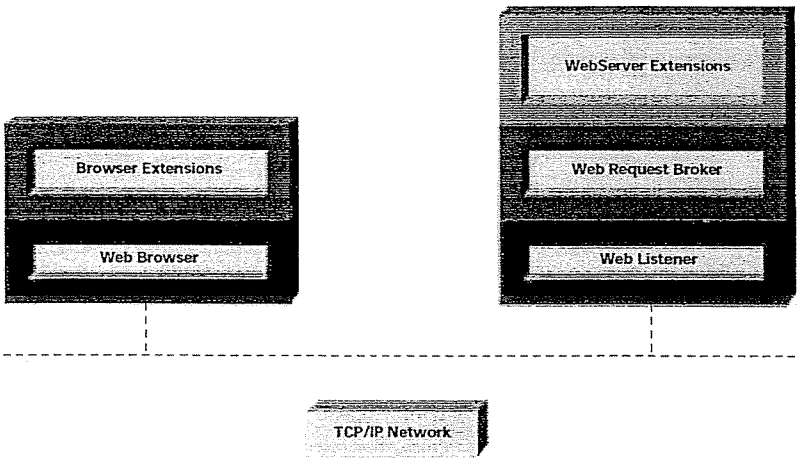
From a web browsers point of view, Oracle WebServer 2.0 looks just like any other web server. The industry standard HTTP 1.0 protocol with optional SSL based data-encryption is used to establish a short-term connection and request an object from the server. When the object is received by the client, the MIME type of the object determines how it is handled: either processed natively by the client, or handed off to a helper application or client-side plugin (extension) for further processing. This defines a complete HTTP transaction and is most often (but certainly not limited to) a request for and delivery of an HTML page, which is displayed in the browsers main viewing area.

Underneath the covers of the Oracle WebServer's HTTP server component, the Oracle Web Listener, interesting things happen to fulfill a request from a web browser. The following sections provides an in-depth description of the three major components of Oracle WebServer 2.0: the *Web Listener, Web Request Broker and the WebServer SDK.*

Oracle Web Listener

The Web Listener is responsible for delivering incoming requests to the Web Request Broker, where they are processed or dispatched to services built with the WebServer SDK.

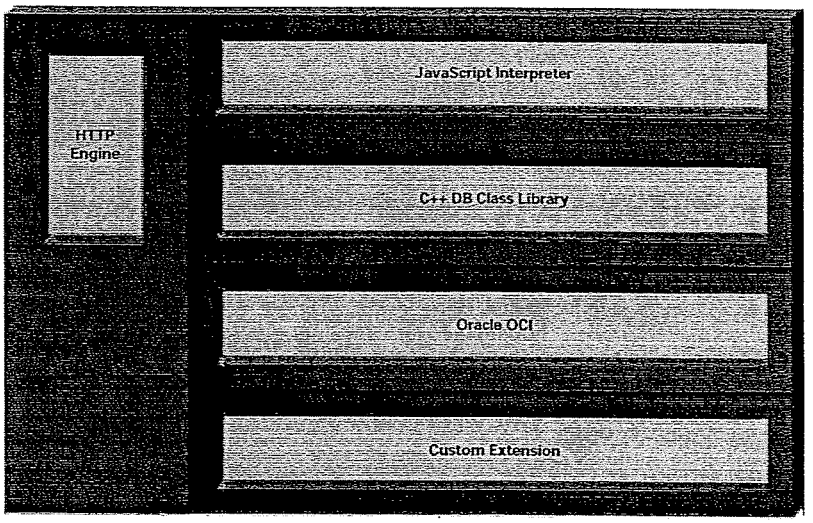
Oracle WebServer 2.0 Architecture



Oracle's HTTP Server is designed to be a very scalable, high-performance network daemon. It is implemented as a multi-threaded, single-process asynchronous engine, allowing multiple concurrent requests to be processed at the same time using standard HTTP or HTTP over SSL. The HTTP Server is the network layer component of the Oracle WebServer. It listens for incoming HTTP requests, delivers static files and runs simple CGI programs, and hands off everything else to the Web Request Broker (WRB), where a request is handled by a server extension. The first generation of websites on the Internet simply offered static HTML files to their visitors. To enable dynamic content, web designers are mainly using CGI to extend the capabilities of their web server. Although easy to use, CGI has some serious limitations, primarily the high overhead of spawning a new process for each incoming request, which causes severe degradation in scalability and throughput.

Today, several webservers have defined an API for implementing extensions to the basic HTTP engine. The typical HTTP API enables a custom extension to be linked into the HTTP server process and allows it to intercept and fulfill a client request. Examples of such APIs are Netscape NSAPI, Spyglass ADI, and Microsoft IS-API. These APIs are becoming increasingly popular because they address the serious shortcomings of the original CGI protocol.

The problem with the HTTP API approach is that all extensions are linked into the same executable and thus share the same memory, process, and signal handling space. Extensions are free to clobber each other, and a poorly designed extension can bring the entire system down. Furthermore, technical support immediately becomes a nightmare: who are you going to call when you have linked together components from 5 different vendors ? What if the extension from vendor A is multi-threaded, and the extension from vendor B isn't thread-safe ?



Typical HTTP server extended through API

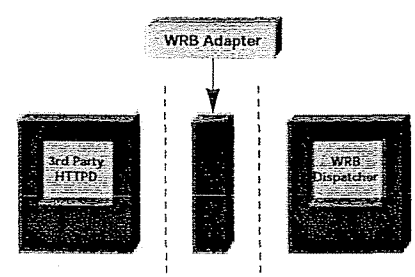
Oracle's answer to these issues is a new and improved architecture for the server, which effectively layers Oracle WebServer 2.0 into two fundamental components: a HTTP engine (the Oracle Web Listener) and a high-speed dispatch mechanism (the Web Request Broker) which routes requests to server extensions running in separate processes.

Being the industry leader in server and networking technology, Oracle fully understands how to optimize performance and throughput in an environment like the World Wide Web. Our experience with products like SQL*Net and Oracle7 show that network service daemons and information servers benefit from different internal architectures. Most important, since the network itself is the largest bottle-neck, the network listener must be lean and fast, and not carry any extra overhead to its main function.

Today, several hundred web server implementations are in use on the Internet. Ranging from a few lines of Perl code to full-blown commercial products on just about every imaginable operating system, this abundance of available HTTP servers demonstrates two things: first, that it is relatively easy to build a HTTP server, and second, that they will become a commodity very soon. Oracle expects HTTP to become an integral part of all new operating system releases, much like standard FTP or TELNET daemons.

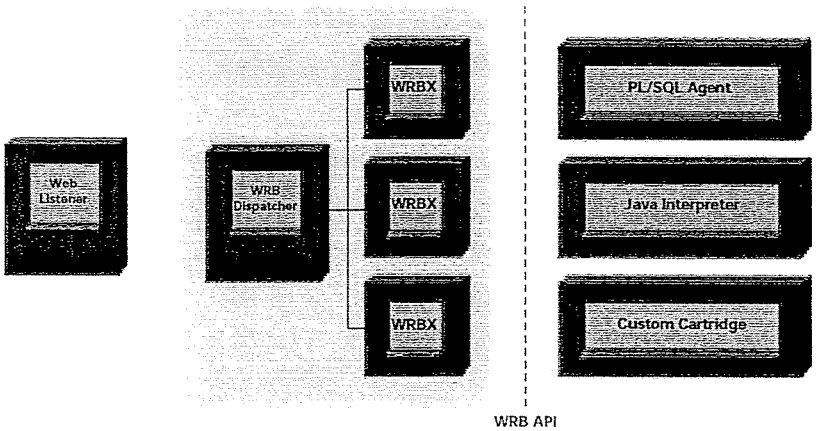
Oracle's mission is to provide an operating system independent development and runtime environment for business applications. As a direct consequence of this objective, we intend to provide support for an increasing number of HTTP engines. Initially, we will provide WRB adapters for the very popular Apache and NCSA servers, but also for other servers such as the Netscape Commerce Server and Open Market's WebServer. Note that these adapters are not provided in the initial release of WebServer 2.0, which only supports the Oracle Web Listener.

Integrating with any HTTP server



The Web Request Broker

Web clients connect to Oracle WebServer 2.0 using HTTP just like any other web server. A request is delivered in the form of a URL, which conceptually is an identifier for a web object (normally an HTML page). The HTTP engine does not make any attempt at interpreting the URL, instead it is immediately handed off to the Web Request Broker for further processing.



The Web Request Broker

The WRB Dispatcher (WRBD)

The WRB dispatcher must decide what type of object is being requested. To do this, it examines the WRB configuration file, which maps virtual directories to WRB Services. For instance, the Web Request Broker may be configured to dispatch all URL requests beginning with `/java` to the integrated Java Interpreter. However, if no suitable WRB Service is found, the request is passed back to the Web Listener and standard processing continues. The WRB Dispatcher performs dynamic load balancing between multiple instances of a WRB Service, and the webmaster can configure each WRB Service to have a minimum and maximum number of instances. An instance corresponds to a process in a UNIX environment.

All inter-process communication is handled by the transport independent WRB protocol. WebServer 2.0 supports standard IPC mechanisms, and future releases will also support other mechanisms, such as for instance OMX, Oracle's CORBA 2.0 compliant Object Request Broker. The advantage of this is clear: WRB Services may be implemented as industry standard distributed objects, enabling true distribution and scalability across multiple nodes.

The WRB Execution Engine (WRBX)

Each WRB Service consists of a common WRB Execution Engine (WRBX) and a shared library, which is dynamically loaded at runtime. By providing an open API to WRBX, Oracle is encouraging partners and customers to integrate their own extensions. Integration is done at a much higher level than CGI or various HTTP server APIs.

The WRB API

The WRB API allows a developer to register three basic callback functions with WRBX: an initialization function, a request handler, and a shutdown function. The WRB API also contains numerous utility functions which may be invoked directly from the three basic callbacks.

The WRB API is being developed in cooperation with Oracle partners and customers, and we are using their feedback to finalize the specification and provide maximum openness and inter-operability between Oracle WebServer and third-party applications.

Building a Cartridge

This section will take you through the steps needed to build your own custom cartridge, or extension, to Oracle WebServer 2.0. Keep in mind that this application development platform is language independent—so you can use any programming language that your operating system supports, as long as your linker can build a shared library from the compiled code. This is a very powerful capability, which allows you to integrate existing legacy applications written in for instance COBOL, as well as develop new cartridges using C or C++. The steps involved in creating a new cartridge are as follows:

1. Create three functions: one to initialize the cartridge, one to shutdown, and one to handle a request.
2. Create an entry point function which takes one parameter of type pointer to a `WRBCallbacks` structure. Fill in the WRBCallbacks structure with your three functions.
3. Link your cartridge with the WRB library to produce a shared library (dynamic link library).
4. Register your library and entry point function in the WRB configuration file using the WebServer Manager.
5. Define one or more virtual directory mappings for your cartridge.

The full specification of the WRB API is available as part of the distribution. Please review it carefully before you attempt to build your first cartridge.

Oracle WebServer 2.0 ships with a sample cartridge called "Hello World!". On UNIX system, you can find the source code in the `ows2/sample/wrbsdk` directory. The file `hello.c` defines the three mandatory functions as follows:

```
WRBReturnCode test_init( WRBCtx, clientcxp )
void  *WRBCtx;
void  **clientcxp;
{
        return (WRB_DONE);
}

WRBReturnCode test_exec( WRBCtx, clientcxp )
void   *WRBCtx;
void  **clientcxp;
{
        WRBClientWrite(WRBCtx, "Content-type: text/html\n\nHello World\n", 40);
        return (WRB_DONE);
}

WRBReturnCode test_shut( WRBCtx, clientcxp )
void   *WRBCtx;
void  **clientcxp;
{
                return (WRB_DONE);
}
```

The first function, **test_init()**, will be our initialization function, which is called once when the cartridge is started. The third function, **test_shut()**, is the shutdown function, which is called once just before the cartridge is stopped. The final function, **test_exec()**, is the guts of the cartridge. It is invoked each time an incoming request is dispatched to this cartridge. As you have probably already guessed, this is a very simple cartridge: nothing is done in the initialization or shutdown functions, and the exec function simply prints a MIME header for an HTML document followed by the words "Hello World".

The next step is to register these three functions as callback function with the WRB Execution Engine (WRBX), which is the program that actually dynamically loads your cartridge at runtime. This is done in the function **testentry()**:

```
WRBReturnCode testentry (WRBCalls)
WRBCallbacks *WRBCalls;
{
        WRBCalls->init_WRBCallback = test_init;
        WRBCalls->exec_WRBCallback = test_exec;
        WRBCalls->shut_WRBCallback = test_shut;

        return (WRB_DONE);
}
```

Finally, you must compile and link your program to produce a shared library which can be loaded by the WRBX at runtime to instantiate your cartridge. The following shows how the shared library is built on Sun Solaris 2.4, but it will differ on just about every operating system:

```
$ cc -c -o helloworld.o -g -I${ORACLE_HOME}/ows2/wrbsdk/inc helloworld.c

$ cc -g -xs -L${ORACLE_HOME}/ows2/wrbsdk/lib -o helloworld.so -G helloworld.o
  -lnsl -lm -lsocket -ldl -laio
```

Please use the Makefile included in the **hello** directory as a template for your operating system. Once the shared library has been successfully built, the next step is to register the cartridge with the Web Request Broker. You can do this with the WebServer Manager HTML user interface. The configuration file in the distribution already has the Hello World cartridge registered as follows (note that **/ora/7.3.2/** will be replaced by whatever your **ORACLE_HOME** is):

```
[Apps]
;
; APP   Object Path                                  Entry Point  Min   Max
; ===   ===========                                  ===========  ===   ===
OWA     /ora/7.3.2/ows2/lib/libndwoa.so              ndwoadinit   0     100
SSI     /ora/7.3.2/ows2/lib/ndwussi.so               ndwussinit   0     100
JAVA    /ora/7.3.2/ows2/lib/libjava.so               ojsdinit     0     100
HELLO   /ora/7.3.2/ows2/sample/wrbsdk/ helloworld.so testentry     0     100

[AppDirs]
;
; Virtual Path          APP     Physical Path
; ============          ===     =============
/ssi                    SSI     /ora/7.3.2/ows2/sample/ssi
/hr/owa                 OWA     /ora/7.3.2/ows2/bin
/ir/owa                 OWA     /ora/7.3.2/ows2/bin
/owa_dba/owa            OWA     /ora/7.3.2/ows2/bin
/java                   JAVA    /ora/7.3.2/ows2/java
/sample/wrbsdk/hello    HELLO   /ora/7.3.2/ows2
/sample/ssi             SSI     /ora/7.3.2/ows2/sample/ssi
/sample/java/run        JAVA    /ora/7.3.2/ows2/sample/java

[AppProtection]
/owa_dba/owa            Basic(Admin Server)

[SSI]
EnableLiveHTML   = TRUE
ParseHTMLExtn    = FALSE
EnableExecTag    = TRUE
ExtensionList    = html shtml lhtml
[JAVA]

CLASSPATH        = /ora/7.3.2/ows2/java/classes.zip:/ora/7.3.2/ows2/java/oracle.zip
                   JavaCacheTimeout = 86400
```

Each line in the [Apps] section is used to define one cartridge. The columns are:

- Cartridge name. A unique name assigned to the cartridge.
- Shared library name. The full path to your application.
- Entry-point. The name of the function that the WRBX will call to register your callbacks.
- The last two columns contain the minimum and maximum number of cartridge instances (threads or processes) that the Web Request Broker will start.

Each line in the `[AppDirs]` section defines a virtual directory path and a working directory for the cartridge. The columns are:

- Virtual directory. When the Web Request Broker sees a HTTP request for an object in this directory path, the request will be dispatched asynchronously to the corresponding cartridge (next column). Note that multiple paths may be defined for the same cartridge.
- Cartridge name. This must correspond to one of the cartridges defined in the `[Apps]` section.
- Working directory for the cartridge.

Each cartridge may optionally have its own section containing parameters, which are accessible through the WRB API. For instance, the Java Interpreter uses the `CLASSPATH` variable to determine where to look for Java classes.

That's all there is to it. This is the way Oracle provides cartridges to extend the base functionality of Oracle WebServer 2.0, and this is the way that 3rd parties and customers can do the same. You will notice that the Java Interpreter, PL/SQL Agent, and LiveHTML are implemented this way. Thus nothing prevents you from integrating any information management system, be it relational databases or legacy systems running on mainframes. If it can be accessed through any programming language on the WebServer node, it can be fully integrated through a custom cartridge and run as a WRB Service.

WebServer SDK

Oracle WebServer 2.0 includes two types of cartridges: system cartridges that perform a well defined function, and programmable cartridges that act as interpreters or runtime environments for applications. An example of a system cartridge is the Verifone VPOS (Virtual Point Of Sale) cartridge, which allows web applications to perform electronic payment transactions using a vast number of payment protocols over the Internet (Note: the VPOS cartridge is not included with Oracle WebServer 2.0, but you can get more information about it at `http://www.verifone.com`). Examples of programmable cartridges are the PL/SQL Agent, Java Interpreter, and LiveHTML, which are all covered in the following section.

ORCL000770

The PL/SQL Agent

The PL/SQL Agent enables web applications to be developed using Oracle7 stored procedures written in PL/SQL. This is a tremendous advantage if you are building a data-driven application and your data happens to be stored in an Oracle7 database. Interacting with database objects is much easier in PL/SQL than any other language. It also has the benefit of being very scalable and completely portable between all operating systems supported by Oracle.

From a developer's perspective, the PL/SQL Agent is no different than the popular "Web Agent", which shipped with Oracle WebServer 1.0. Both enable developers to build data-driven, dynamic HTML pages with personalized content using Oracle7 stored procedures. Due to the new architecture in Oracle WebServer 2.0 with the Web Request Broker, the PL/SQL Agent is roughly an order of magnitude faster than the previous version. This is mainly due to the fact that each instance of the PL/SQL Agent stays connected to Oracle7 between requests, and thus does not need to establish a new database connection each time a stored procedure needs to be executed. Note that the Web Agent implemented through CGI is still bundled with WebServer 2.0 for backwards compatibility purposes, but its use is not recommended since the new architecture is so much faster.

The PL/SQL Agent performs two major tasks which enable a stored procedure to generate a dynamic HTML page. The first is to launch the procedure and pass in any parameters that were part of the HTTP request. The translation of a HTTP request to PL/SQL is not obvious, and we will therefore explore this in greater detail in a moment. The second task performed by the PL/SQL Agent is to provide a method for any stored procedure to actually deliver "output". In addition to performing these tasks, the PL/SQL Agent includes the PL/SQL Web Toolkit: a collection of packaged procedures, functions, and utilities to enhance productivity and ensure valid HTML output.

Launching a Procedure

For a request to be handed off to the PL/SQL Agent, the leading directory path must be found in the WRB configuration file. When you install WebServer 2.0, the following directory mappings are pre-defined:

```
/hr/owa                 CWA     /ora/7.3.2/ows2/bin
/tr/owa                 CWA     /ora/7.3.2/ows2/bin
/owa_dba/owa            CWA     /ora/7.3.2/ows2/bin
```

So any request starting with for instance the directory path `/tr/owa` will be recognized and handed off to the PL/SQL agent. Lets look at a sample URL:

```
http://www.oracle.com/tr/owa/tr.splash
```

This URL consists of three different parts: the first part is simply a pointer to the webserver (`http://www.oracle.com`). The second part is the directory path (`/tr/owa`) which has an entry in the WRB configuration file. At this point we know that we will be handing off the request to the PL/SQL Agent. What follows after the directory path (`/tr.splash`) is interpreted as the name of a

stored procedure (the leading slash is ignored), so in this example the PL/SQL Agent will attempt to run a procedure named `tr.splash`. The actual procedure is named `splash` and is part of the public interface to a package named tr, hence the dot-notation.

Connecting to the Database

The Oracle7 server has a higher level of security than flat files managed by an operating system. Users are authenticated using a wide range of mechanisms ranging from simple passwords to fingerprint scanners, smart-cards, and distributed protocols such as DCE and Kerberos. The PL/SQL Agent also needs to authenticate itself to gain access to a distinct schema of objects and storage space within the database. It also needs to have access to all the details required to actually establish a permanent connection: typically a set of environment variables (`ORACLE_HOME` and `ORACLE_SID` for UNIX systems) and perhaps a SQL*Net connect descriptor. All of this information is referred to as a DCD (Database Connection Descriptor), and all of the DCDs known to the PL/SQL Agent are stored in a configuration file named `owa.cfg` (DCDs where known as Web Agent Services in WebServer 1.0). Care should be taken to prevent unauthorized access to this file, since it contains sufficient information to gain access to the database. DCDs are defined in the PL/SQL Agent's configuration file, but they are activated by creating a directory mapping in the Web Request Broker's configuration file.

At the lowest level, Oracle's internal API allows us to establish a connection to an Oracle7 Server using SQL*Net in one step, and actually logging on to the database in a separate step. Since each HTTP request needs a new session, we are technically logging on and off between requests, but this is still extremely fast since the Oracle7 Server connection is maintained between requests.

The token immediately preceding the keyword owa in the directory path is interpreted as the name of a DCD. For instance, `/tr/owa` would cause the PL/SQL Agent to look for a DCD named tr. If such a DCD does not exist, an attempt is made to use the default DCD. If no procedure is found or if the default DCD has not been created, the request will fail.

In Oracle WebServer 2.0 one or more instances of the PL/SQL Agent will be running for each active DCD. Incoming requests will share the same DCD if they are using the same directory path, and authentication of the user will be handled at the HTTP level by the Oracle Web Listener. You will typically configure the Oracle WebServer to have at least one public DCD and several protected DCDs requiring user authentication. When a user has authenticated with a username and password, the username is available to the stored procedure being run, but it is up to the application developer to implement security.

Passing Parameters

The HTTP protocol supports several methods of passing information from the client to the server along with a request. The POST method passes parameters in a stream (standard input) directly to the Web Request Broker (or a CGI program). The GET method tags parameters onto the end of a request. For instance, if two parameters where to be passed to the **tr.splash** procedure, the complete URL would look like this:

```
http://www.oracle.com/tr/owa/tr.splash?name1=value1&name2=value2.
```

The parameters are encoded and passed as name/value pairs with & as a delimiter. If you were developing a web application from scratch, you would have to parse this "query string" and decode the parameters yourself. Fortunately, the Web Request Broker does this automatically for all cartridges, and the PL/SQL Agent converts the parameters to PL/SQL procedure parameters. Furthermore, you don't have to care about what method is used to pass the parameters, they are simply delivered as parameters to your procedure. To complete this example, the URL above with the two parameters would be translated into a PL/SQL procedure call like this:

```
begin
    tr.splash ( name1 => 'value1', name2 => 'value2' );
end;
```

This parameter passing notation may not be familiar to you even if you are an experienced PL/SQL developer, but naming each parameter this way essentially enables us to not worry about the order they appear in. For instance, this procedure could actually have been defined like this:

```
create package tr as
    procedure splash ( name2 in varchar2 := null, name1 in varchar2 := null);
end;
```

Note that all parameters have default values (null). This ensures that the procedure will be found even if one or more parameters are missing. One additional feature is worth mentioning: if you have a large or variable number of parameters, the PL/SQL Agent allows you to use PL/SQL tables of varchar2 as input parameters. Consider an electronic storefront. The user will first search through the product catalog "putting" items into an electronic shopping cart. When the order is ready to be processed, an HTML purchase form with a variable number of line-items needs to be submitted from the browser. If the input fields for each line-item have identical names in the HTML form (e.g. re-occurring **item_partno** and **item_quantity**), the following procedure could handle the request:

```
create package shop as
    type vc2arr is table of varchar2(255) index by binary_integer;
    procedure buy ( item_partno in vc2arr, item_quantity in vc2arr);
end;
```

We have now covered the basics of how an Oracle7 stored procedure is launched by the PL/SQL Agent and how input parameters are passed from the client browser to the procedure. The next section explains how to generate the resulting dynamic HTML using the PL/SQL Web Toolkit.

Generating HTML with the PL/SQL Web Toolkit

The toolkit consists of a number of packages that are installed into each schema corresponding to a DCD when the DCD first is created. The two packages named **HTP** and **HTF** (HyperText Procedures and HyperText Functions) make up the core of the PL/SQL Web Toolkit and provide a low-level one-to-one mapping between HTML 2.0 and PL/SQL procedures and function. At the lowest level, the procedure **htp.print** plays a central role, in that it is ultimately used either directly by the developer or indirectly by a higher-level procedure or function to put something into the result, which is transmitted back to the client. Although the result will usually be a HTML document, any MIME type is supported, including user-defined types. Here is an example of a simple procedure:

```
create procedure hello is
begin
    htp.print('Hello World');
end;
```

The **htp.print** procedure simply puts whatever parameter you pass to it on a page. A shortcut procedure is available with the name **htp.p**. You can of course add HTML tags manually, or you can use the functions and procedures in the PL/SQL web toolkit that encapsulate HTML for you. The following two procedures generate the exact same result, **hello1** simply prints the result exactly as desired, whereas **hello2** uses the toolkit procedures and functions:

```
create procedure hello1 is
begin
    htp.p('<title>Hello Sample</title>');
    htp.p('<h1>Hello Sample</h1>');
    htp.p('Hello World, this is <b>easy</b>');
    htp.p('<hr>');
    htp.p('<a href="http://www.oracle.com">Visit Oracle on the Web!</a>');
end;

create procedure hello2 is
begin
    htp.htitle('Hello Sample');
    htp.p('Hello World, this is ' || htf.bold('easy'));
    htp.line;
    htp.anchor('http://www.oracle.com','Visit Oracle on the Web!');
end;
```

The second example demonstrates a powerful way of bundling multiple HTML tags into widgets. The **htp.htitle** procedure generates both a title and a header tag. This concept can easily be expanded to higher levels, so you could

ORCL000774

for instance define a common procedure to set your document title and header, background bitmap, text and link colors, and navigation toolbar. In addition to increasing productivity, this technique automatically brings a consistent look and feel to your applications.

Java Interpreter

For a general introduction to Java, please visit `http://java.sun.com`. This section assumes general knowledge of Java as a technology, although a very brief introduction follows. Java is an objective-oriented language strongly resembling C++ but with a few restrictions that strongly improve application robustness, the most important being no explicit pointer manipulation and no multiple class-inheritance.

Java allows you to develop three types of classes: normal Java classes which are used as building blocks or abstract classes, runnable classes, known as Java applets, which can be downloaded by a Java-enabled web browser, and Java applications, which are standalone programs. Oracle WebServer 2.0 allows you to integrate the last type, standalone applications, into your website using the Oracle Java Interpreter. Oracle WebServer 2.0 (and 1.0 for that matter) can also act as a repository for Java applets to be downloaded to clients, but that is a different topic not covered here.

The distinguishing attribute of a Java application compared to an applet or other class, is that it contains a public method (function) called `main`. This method acts as the entry-point to the application. Oracle WebServer 2.0 ships with a few sample Java applications, of which the simplest looks like this:

```
class HelloWorld {
    public static void main (String args[ ]) {
            System.out.println("Content-type: text/html\n\n");
            System.out.println("Hello World!");
            System.out.println("Hello World!");
    }
}
```

Anything printed to the standard output stream from a Java application is simply delivered back to the client.

Extending Java

Oracle WebServer also contains the Oracle Java Web Toolkit, three Java packages to extend the capabilities of the standard Java interpreter. These are `oracle.html.*`, which contains objects for dynamic HTML generation, `oracle.rdbms.*`, which contains objects for database access, and `oracle.plsql.*`, which contains objects for PL/SQL access. These packages must be explicitly imported by any application that intends to use them.

Executing PL/SQL from Java

The Java Interpreter interfaces to the Oracle7 Server by running PL/SQL packages or standalone PL/SQL procedures and functions. Each package or procedure must have a Java package wrapper, which is a Java class containing methods to call that package's procedures and functions. Standalone procedures and functions are all wrapped in a single wrapper. Once you have identified or created the PL/SQL packages your applications needs, you can create the package wrappers automatically by running the `pl2java` utility as follows:

```
pl2java [flags] username/password[@connect-string] packagename...
```

This utility creates a Java wrapper class for each package or procedure given as an argument to the command. When your application is run, it creates an instance of this class to interface to the package. The following is a summary of steps required to write a Java application which interacts with an Oracle7 Server through PL/SQL:

1. Create an object of type Session to handle the database connection. All of the operations performed during this session must be called from within this object.
2. For each PL/SQL package used in the application, you must create one instance of the packagewrapper subtype created for that PL/SQL package by the `pl2java` utility.
3. For each parameter of a PL/SQL package, you must create an instance of the Java variable that matches that parameter.

The following example uses a packaged function named **`employee.count_employees`** to get a tally of people in a given department. For the sake of simplicity, all error and exception handling has been left out.

```
import oracle.rdbms.*;
import oracle.plsql.*;

public class EmployeeReport {
     public static void main (String args[]) {
               .
               int tally = get_count("SALES");
               .
     }

     int get_count( String deptName ) {

     Session.setProperty("ORACLE_HOME", "/user/oracle");
     Session.setProperty("TNS_ADMIN", "/user/oracle/network/admin");

     Session session;
     session = new Session("scott", "tiger", "HR_DB");

     // Create a new instance of Employee package
     Employee employee = new Employee(session);

     // Create object to encapsulate PL/SQL parameter
     PDouble       pEmployeeCount;
```


ORCL000776

```
    // ADD: call Employee package to count the number of
    // employees in the department
    pEmployeeCount = employee.count_employees(pDeptName);
    return (int)pEmployeeCount.doubleValue();
  }
```

Generating HTML from Java

A Java application can generate a dynamic HTML page. Essentially, anything written to standard output by a Java application run by the Java Interpreter will be returned to the client browser. The "Hello World" example described earlier demonstrates this. Similar to the PL/SQL Web Toolkit, the Oracle Java Web Toolkit allows you to generate HTML without having to hardcode the actual HTML tags.

To generate dynamic HTML from within Java, you create various objects that use the interface **`IHtmlItem`**. All classes that generate dynamic HTML implement this interface, which has two simple methods: **`toHTML`** and **`print`**. Both of these methods produce the content of the object as HTML, but **`toHTML`** returns it as a string, whereas **`print`** sends it to system output. Therefore, you use **`toHTML`** when you want the resulting HTML to be further processed by another method and use **`print`** when you want to output it. In effect, you build your Web page in a buffer with **`toHTML`** and flush the buffer with **`print`**.

The **`oracle.html`** package provides a standard set of classes based on HTML2, HTML3, and popular browser-specific extensions. You are not limited to these, however. You can easily create your own customizable HTML classes by deriving them from the **`CompoundItem`** or **`Container`** classes. The **`oracle.html`** package also has the intelligence to generate output that is optimized for the browser at hand. For example, a browser that does not support tables will get table data in the form of pre-formatted strings. Here is a simple example:

```
import oracle.html.*;

public class HelloWorld {
    public static void main (String args[]) {

            // Create an HtmlHead Object titled "Hello World!"
            HtmlHead hd = new HtmlHead("Hello World!");

            // Create an HtmlBody Object
            HtmlBody bd = new HtmlBody();

            // Create an HtmlPage Object
            HtmlPage hp = new HtmlPage(hd, bd);

            // Adds a simple string "Hello World" in this page
            bd.addItem("Hello World!");

            // Print out the content of this Page
            hp.print();
    }
}
```

LiveHTML

One of the most popular features of the Web is known as *Server Side Includes* (SSI). Certain documents can be parsed by a web server before sending them back to the client, and they may contain "include" directives instructing the web server to insert information into the document on the fly. Examples of use are page-hit counters, timestamps, etc. Server Side Includes literally bring otherwise static HTML pages to life, hence the name LiveHTML. Oracle WebServer 2.0 supports SSI+, a common extension to NCSA's original specification.

LiveHTML code is formatted as SGML comments, so that it is ignored should the file ever find its way to the browser unparsed. The format for LiveHTML codes, therefore, is the following:

```
<!--#command tag1="value1" tag2="value2" -->
```

The tags are arguments to the commands, most of which actually only accept one of the possible tags. The possible commands and their associated tags are as follows:

`config`. This command sets parameters for how the file or script is to be parsed and therefore is normally the first LiveHTML command in a file. The possible tags are:

- `errmsg`. This specifies the error message that is sent to the client if an error occurs while parsing this document. Here is an example:

  ```
  <!--#config errmsg="A parse error occurred in the Hockey_Pass file"-->
  ```

- `timefmt`. This specifies a date format. LiveHTML files frequently include timestamps. The conventions follow the strftime library call supported in most versions of Unix, even if the WebServer is not running on a Unix platform.

- `sizefmt`. This specifies the format used when displaying a file size. The possibilities are bytes, which gives the absolute size in bytes, and abbrev, which gives the size in kilobytes or megabytes as appropriate.

- `cmdecho`. This specifies whether non-CGI scripts subsequently executed have their output incorporated into this HTML page. The possible values are ON and OFF. ON specifies that the output is included. The default is OFF.

- `cmdprefix`. Specifies a string that will be prepended to each line of the script output.

- `cmdpostfix`. Specifies a string that will be appended to each line of the script output.

`include`. This command specifies that a file is to be included in the generated HTML page at this point. The file can any of the following:

- another LiveHTML file like the current one.

ORCL000778

- a regular HTML file.
- an ASCII file. which of these it is determined, as usual, by the extension.

The possible tags are:

- `virtual`. This gives a virtual path to the file. The directory mappings for virtual paths are set by the WebServer administrator using WebServer Manager.
- `file`. This gives a pathname relative to the current directory. References to parent directories or uses of absolute pathnames are forbidden.

`echo`. This gives the value of an environment variable. This variable is either one of the standard CGI environment variables or one of the LiveHTML extensions, which currently are all standard Server Side Include variables. There is only one tag, var, and it must be present. It provides the name of the variable. The LiveHTML environment variables are as follows:

- `DOCUMENT_NAME`. The current filename.
- `DOCUMENT_URL`. The virtual path to this file.
- `QUERY_STRING_UNESCAPED`. If the client sent a query string, this is an unescaped version of it, with all shell-special characters escaped with \. DATE_LOCAL The current date and local time zone in the format specified by the most recent config timefmt command.
- `DATE_GMT`. Same as the above, but in Greenwich mean time.
- `LAST_MODIFIED`. The last modification date of the file, given in the format specified in the last config timefmt command.
- `fsize`. This produces the size of the file in the format specified in the most recent config filesize command. Tags are the same as for include.
- `flastmod`. This produces the last modification date of the file in the format specified in the most recent config timefmt command. Tags are the same as for include.
- `exec`. This is the command to execute a script. The tags specify whether or not the script is CGI.
- `cmd`. This specifies a non-CGI script. Execution is passed to the Operating System, and the given string is parsed as though it were entered at a command-line interface. The full path of the script must be given. The non-CGI environment variables specified under echo above can be referenced. Whether the output of the script is included in the HTML page that the parser outputs, is determined by the most recently executed config cmdecho.
- `cgi`. This specifies a CGI script. The value given will be the virtual path of the script. URL locations are automatically converted into HTML anchors.

Summary

First generation webservers have fundamental shortcomings in the areas of scalability, management, and extensibility. Oracle is leveraging its core competency in these areas to deliver a next-generation product capable of addressing these issues and making extremely large websites with hundreds of thousands of users and terabytes of information a reality. Oracle WebServer 2.0 lays the foundation and provides a first glimpse of new emerging technology, which will be fully developed in forthcoming releases. After all, this is Internet Technology, and we are finding our product life-cycles being reduced from years to months.

Oracle WebServer 3.0 is scheduled for release in late 1996, and we expect the Web Request Broker API to be fully developed by that time. Oracle is actively working with partners and developers to provide a solid platform for application development. Other plans for WebServer 3.0 include making extensive use of Oracle7 for website configuration management, including user authentication, optionally storing content with automatic replication to multiple sites, more cartridges, inter-cartridge processing, and full support for all major HTTP listeners. It is going to be an exciting year—be sure to keep an eye on our public website, `http://www.oracle.com`, for upcoming news, releases, and events.

Magnus Lonnroth
Senior Product Manager
Oracle Internet Server products

Terminology and Acronyms

Term	Description
Apache	An increasingly popular HTTP server on the Internet
API	Application Programming Interface
CGI	Common Gateway Interface
daemon	Detached process waiting for incoming service-requests over the network
FTP	File Transfer Protocol
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
Java	An object-oriented language for the Web
JavaScript	A language developed by Netscape for use in the Netscape Navigator and the Netscape Commerce Server through NSAPI
JDBC	ODBC for Java
LiveHTML	Currently same as Server Side Includes.
MIME	Multi-purpose Internet Mail Extensions
NCSA	National Center for Super-computing Applications
NSAPI	Netscape Server API
PL/SQL	Oracle's procedural language for stored procedures
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
SQL*Net	Oracle's network protocol which runs on top of most networks
SSI	Server Side Includes
SSL	Secure Sockets Layer
TELNET	Terminal Emulation Program
URL	Uniform Resource Location
VRML	Virtual Reality Markup Language
W3C	World Wide Web Consortium
WRB	Web Request Broker
WRB API	An API for developing custom cartridges
WRBD	Web Request Broker Dispatcher
WRBX	Web Request Broker Execution Engine

ORCL000781