Oracle9*i*AS Reports Services

Publishing Reports to the Web

Release 9.0

Part No. A92102-01

February 2002

ORACLE

Oracle9*i*AS Reports Services Publishing Reports to the Web, Release 9.0

Part No. A92102-01

Primary Author: Joan Carter

Contributing Authors: Robin J. Fisher, Frank Rovitto, and Philipp Weckerle

Contributors: Shaun Lin, Vinay Pamadi, Rajesh Ramachandran, Danny Richardson, Jim Safcik, J. Toby Shimizu, Jeff Tang, and Vanessa Wang

Contents

Send Us Your Comments xvii

Preface xix

Intended Audience xix
Documentation Accessibility xx
Accessibility of Code Examples in Documentation xx
Structure xx
Related Documents xxi
Notational Conventions xxii

Part I Preparing Your Environment

1 Oracle9iAS Reports Services Architecture

1.1 Overview of Oracle9*i*AS Reports Services 1-1
1.2 Oracle9*i*AS Reports Services Components 1-4
1.3 Oracle9*i*AS Reports Services Runtime Process 1-6
1.4 Things to Consider When You Set Up Your System 1-8
1.4.1 Choosing the Types of Requests You Will Service 1-9
1.4.2 Choosing Servlet, JSP, or CGI 1-9
1.4.3 Choosing Single- Or Multiple-Machine Configurations 1-10
1.4.4 Choosing Whether to Cluster Multiple Servers 1-11

2 Starting and Stopping Oracle9*i*AS Reports Services

2.1 Starting the Reports Server 2-1
2.1.1 Installing and Starting the Reports Server as a Service (Windows NT/2000) 2-2
2.1.2 Starting the Reports Server as a Servlet (Windows and UNIX) 2-3
2.1.3 Starting the Reports Server from a Command Line (Windows and UNIX) 2-3
2.2 Verifying the Reports Servlet and Server Are Running 2-4
2.3 Verifying that the Oracle HTTP Server Is Running 2-4
2.4 Stopping the Reports Server 2-5

3 Configuring Oracle9*i*AS Reports Services

3.1 Oracle9*i*AS Reports Services Configuration Files 3-1
3.2 Configuring the Oracle9*i*AS Reports Server 3-3
3.2.1 Reports Server Configuration Elements (rwserverconf.dtd) 3-4
3.2.1.1 server 3-5
3.2.1.2 compatible 3-6
3.2.1.3 cache 3-7
3.2.1.4 engine 3-8
3.2.1.5 security 3-11
3.2.1.6 destination 3-13
3.2.1.7 job 3-14
3.2.1.8 notification 3-15
3.2.1.9 log 3-17
3.2.1.10 jobStatusRepository 3-18
3.2.1.11 trace 3-19
3.2.1.12 connection 3-21
3.2.1.13 queue 3-24
3.2.1.14 persistFile 3-25
3.2.1.15 identifier 3-26
3.2.1.16 pluginParam 3-27
3.3 Configuring the Reports Servlet 3-28
3.3.1 Specifying the location of the key map file 3-30
3.3.2 Reloading the Key Map File 3-30
3.3.3 Hiding Verbose Error Messages 3-30
3.3.4 Selecting Login Dialog Boxes 3-31
3.3.5 Setting up Trace Options for the Reports Servlet and JSPs 3-32

3.3.6 Customizing the Appearance of Server Error Messages ... 3-33
3.3.7 Specifying an In-Process Server ... 3-33
3.3.8 Identifying the Default Reports Server ... 3-34
3.3.9 Pointing to Dynamically Generated Images ... 3-34
3.3.10 Setting Expiration for DB Authentication and SYSAUTH Cookies ... 3-35
3.3.11 Setting an Encryption Key for the DB Authentication Cookie ... 3-35
3.3.12 Adding Formatting to Diagnostic/Debugging Output ... 3-35
3.3.13 Specifying an SSL Port Number ... 3-35
3.3.14 Defining the rwservlet Help File ... 3-36
3.3.15 Specifying the Use of Single Sign-On ... 3-36
3.4 Configuring the URL Engine ... 3-36
3.5 Entering Proxy Information ... 3-38
3.6 Configuring the Reports Server for Oracle Enterprise Manager ... 3-39

4 Configuring Destinations for Oracle9*i*AS Reports Services

4.1 Overview of Output Processing ... 4-1
4.2 Registering Destination Types with the Server ... 4-4
4.2.1 Setting Up a Destination Section in the Server Configuration File ... 4-4
4.2.2 Entering Valid Values for a Destination ... 4-5
4.2.2.1 Destination destypes and classes ... 4-5
4.2.2.2 Destination Property name/value Pairs ... 4-6

5 Controlling User Access

5.1 Introduction to Oracle9*i*AS Portal ... 5-2
5.2 Defining Portal-Based Security in the Server Configuration File ... 5-3
5.3 Creating Reports Users and Named Groups in Oracle9iAS Portal ... 5-4
5.3.1 Default Reports-Related Groups ... 5-5
5.3.1.1 RW_BASIC_USER ... 5-5
5.3.1.2 RW_POWER_USER ... 5-5
5.3.1.3 RW_DEVELOPER ... 5-6
5.3.1.4 RW_ADMINISTRATOR ... 5-6
5.3.2 Creating Users and Groups ... 5-6
5.4 Setting Up Access Controls ... 5-6
5.4.1 Creating an Availability Calendar ... 5-7
5.4.1.1 Creating a Simple Availability Calendar ... 5-7

5.4.1.2 Creating a Combined Availability Calendar 5-10
5.4.2 Registering a Printer 5-12
5.4.3 Registering a Reports Server 5-14
5.4.4 Registering a Report 5-16

6 Reports Server Clusters

6.1 Cluster Overview 6-1
6.2 Setting Up a Cluster 6-2
6.2.1 Renaming a Reports Server 6-3
6.2.2 Generating New Public and Private Keys 6-5
6.2.3 Entering Public and Private Keys in the Server Configuration File 6-5
6.2.4 Restarting the Reports Server 6-6
6.2.5 Submitting a Request to a Cluster 6-6

7 Data Source Single Sign-On

7.1 SSO Architecture 7-2
7.1.1 SSO Components 7-2
7.1.2 SSO Transactions 7-4
7.2 Methods for Setting Up User Connection Strings 7-5
7.2.1 Initial Requirements 7-5
7.2.2 Method 1: Giving Users Access to the OID 7-6
7.2.3 Method 2: Assigning Connection Strings and Letting Users Input at Login 7-6

Part II Sending Requests to the Server

8 Running Report Requests

8.1 The Reports URL Syntax 8-1
8.1.1 Servlet 8-2
8.1.2 JSP 8-3
8.1.3 CGI 8-4
8.2 Report Request Methods 8-5
8.3 Publishing a Report Portlet in Oracle9iAS Portal 8-7
8.3.1 Creating a Provider for Your Reports 8-7
8.3.2 Creating the Report Definition File Access 8-7

8.3.3 Adding the Report Portlet to a Page 8-8
8.4 Specifying a Report Request from a Web Browser 8-9
8.5 Sending a Request to the URL Engine 8-10
8.6 Scheduling Reports to Run Automatically 8-10
8.7 Additional Parameters 8-11
8.8 Reusing Report Output from Cache 8-11
8.8.1 Usage Notes 8-12
8.9 Using a Key Map File 8-13
8.9.1 Enabling Key Mapping 8-13
8.9.2 Adding Key Mapping Entries to a Key Map File 8-14
8.9.3 Using a Key with Everything but JSPs 8-15
8.9.4 Using a Key with a Report Run as a JSP 8-15

9 Creating Advanced Distributions

9.1 Distribution Overview 9-1
9.2 Introduction to Distribution XML Files 9-2
9.2.1 The distribution.dtd File 9-2
9.2.2 A Brief Word About Using Variables within Attributes 9-2
9.3 Elements of a Distribution XML File 9-4
9.3.1 destinations 9-4
9.3.2 foreach 9-5
9.3.3 mail 9-7
9.3.4 body 9-10
9.3.5 attach 9-11
9.3.6 include 9-13
9.3.7 file 9-16
9.3.8 printer 9-18
9.3.9 destype 9-19
9.3.10 property 9-22
9.4 Distribution XML File Examples 9-22
9.4.1 foreach examples 9-22
9.4.1.1 Single E-Mail with Report Groups as Separate Attachments 9-23
9.4.1.2 Separate E-Mail for Each Group Instance 9-23
9.4.1.3 Separate E-Mails with Separate Sections as Attachments 9-23
9.4.1.4 Separate File for Each Section 9-24

9.4.1.5 Separate Print Run for Each Report ... 9-24
9.4.1.5.1 Windows ... 9-25
9.4.1.5.2 UNIX ... 9-25
9.4.2 mail examples ... 9-25
9.4.2.1 E-Mail with a Whole Report as the Body ... 9-25
9.4.2.2 E-Mail with a Section of a Report as the Body ... 9-26
9.4.2.3 E-Mail with Two Report Sections as the Body ... 9-26
9.4.2.4 E-Mail with External File as Body and Report as Attachment ... 9-26
9.4.2.4.1 Windows ... 9-26
9.4.2.4.2 UNIX ... 9-27
9.4.2.5 E-Mail with Whole Report and Grouped Sections Attached ... 9-27
9.4.2.6 E-Mail to Relevant Manager and Department ... 9-27
9.4.3 file examples ... 9-28
9.4.3.1 File for Whole Report ... 9-28
9.4.3.1.1 Windows ... 9-28
9.4.3.1.2 UNIX ... 9-28
9.4.3.2 File for Combined Report Sections ... 9-29
9.4.3.3 File for Each Group of Combined Sections ... 9-29
9.4.3.4 File for Each Report Group Instance ... 9-29
9.4.4 printer examples ... 9-29
9.4.4.1 Print Whole Report ... 9-30
9.4.4.1.1 Windows ... 9-30
9.4.4.1.2 UNIX ... 9-30
9.4.4.2 Print Two Sections of a Report ... 9-30
9.4.4.2.1 Windows ... 9-30
9.4.4.2.2 UNIX ... 9-30
9.4.4.3 Print Grouped Report ... 9-31
9.4.4.3.1 Windows ... 9-31
9.4.4.3.2 UNIX ... 9-31
9.4.4.4 Print Combined Sections for Each Group Instance ... 9-31
9.4.4.4.1 Windows ... 9-31
9.4.4.4.2 UNIX ... 9-31
9.4.4.5 Print Relevant Instance of a Report to Its Relevant Printer ... 9-32
9.5 Using a Distribution XML File at Runtime ... 9-32
9.6 XSL Transformation for Custom / Pluggable Destinations ... 9-33

10 Customizing Reports with XML

10.1 Customization Overview 10-2
10.2 Creating XML Customizations 10-3
10.2.1 Required XML Tags 10-4
10.2.2 Changing Styles 10-4
10.2.3 Changing a Format Mask 10-5
10.2.4 Adding Formatting Exceptions 10-5
10.2.5 Adding Program Units and Hyperlinks 10-7
10.2.6 Adding a New Query and Using the Result in a New Header Section 10-8
10.3 Creating XML Data Models 10-9
10.3.1 Creating Multiple Data Sources 10-9
10.3.2 Linking Between Data Sources 10-10
10.3.3 Creating Group Hierarchies within Each Data Source 10-11
10.3.4 Creating Cross-Product (Matrix) Groups 10-12
10.3.5 Creating Formulas, Summaries, and Placeholders at any Level 10-13
10.3.6 Creating Parameters 10-14
10.4 Using XML Files at Runtime 10-16
10.4.1 Applying an XML Report Definition at Runtime 10-16
10.4.1.1 Applying One XML Report Definition 10-16
10.4.1.2 Applying Multiple XML Report Definitions 10-17
10.4.1.3 Applying an XML Report Definition in PL/SQL 10-18
10.4.1.3.1 Applying an XML Definition Stored in a File 10-18
10.4.1.3.2 Applying an XML Definition Stored in Memory 10-18
10.4.2 Running an XML Report Definition by Itself 10-21
10.4.3 Performing Batch Modifications 10-21
10.5 Debugging XML Report Definitions 10-22
10.5.1 XML Parser Error Messages 10-22
10.5.2 Tracing Options 10-23
10.5.3 RWBUILDER 10-23
10.5.4 Writing XML to a File for Debugging 10-24

11 Event-Driven Publishing

11.1 The Event-Driven Publishing API 11-2
11.1.1 Elements of the API 11-2
11.1.2 Creating and Manipulating a Parameter List 11-2

11.1.2.1 Add_Parameter 11-3
11.1.2.2 Remove_Parameter 11-3
11.1.2.3 Clear_Parameter_List 11-4
11.1.3 How to Submit a Job 11-4
11.1.4 How to Check for Status 11-5
11.1.5 Using the Servers' Status Record 11-6
11.2 Debugging Applications That Use the Event-Driven Publishing API 11-7
11.3 Invoking a Report From a Database Event 11-8
11.4 Integrating with Oracle9i Advanced Queuing 11-9
11.4.1 Creating a Queue That Holds Messages of Type SRW_PARAMLIST 11-10
11.4.2 Creating the Enqueuing Procedure 11-11
11.4.3 Creating the Dequeuing Procedure 11-12

Part III National Language Support and Bidirectional Support

12 NLS and Bidirectional Support

12.1 NLS Architecture 12-2
12.1.1 Language-Independent Functions 12-2
12.1.2 Language-Dependent Data 12-2
12.2 NLS Environment Variables 12-2
12.2.1 NLS_LANG Environment Variable 12-3
12.2.1.1 Defining the NLS_LANG Environment Variable 12-6
12.2.1.1.1 Windows 12-6
12.2.1.1.2 UNIX 12-7
12.2.1.2 Character Sets 12-7
12.2.1.2.1 Character Set Design Considerations 12-7
12.2.1.2.2 Font Aliasing on Windows Platforms 12-7
12.2.1.3 Language and Territory 12-8
12.2.2 DEVELOPER_NLS_LANG and USER_NLS_LANG Environment Variables 12-9
12.3 Specifying a Character Set in a JSP or XML File 12-10
12.4 Bidirectional Support 12-12
12.5 Unicode 12-13
12.5.1 Unicode Support 12-13
12.5.2 Unicode Font Support 12-14
12.5.3 Enabling Unicode Support 12-15

12.5.4 Using ALTER SESSION 12-15
12.6 Translating Applications 12-16

Part IV Performance

13 Managing and Monitoring Oracle9*i*AS Reports Services

13.1 Navigating to Reports Services Information in OEM 13-2
13.2 Starting, Stopping, and Restarting Reports Servers 13-2
13.3 Viewing and Managing Reports Job Queues 13-3
13.3.1 Viewing and Managing the Current Jobs Queue 13-4
13.3.1.1 Viewing a Report Server's Current Jobs Queue 13-4
13.3.1.2 Cancelling a Current Job 13-5
13.3.2 Viewing and Managing the Scheduled Jobs Queue 13-5
13.3.2.1 Viewing a Report Server's Scheduled Jobs Queue 13-5
13.3.2.2 Cancelling a Scheduled Job 13-6
13.3.3 Viewing and Managing the Finished Jobs Queue 13-6
13.3.3.1 Viewing a Report Server's Finished Jobs Queue 13-7
13.3.3.2 Viewing a Job's Trace File 13-8
13.3.3.3 Viewing a Result from Cache 13-8
13.3.3.4 Rerunning a Finished Job 13-9
13.3.4 Viewing and Managing the Failed Jobs Queue 13-9
13.3.4.1 Viewing a Report Server's Failed Jobs Queue 13-9
13.3.4.2 Viewing a Failed Job's Trace File 13-10
13.3.4.3 Rerunning a Failed Job 13-11
13.4 Monitoring Server Performance 13-11
13.5 Viewing and Changing Reports Server Configuration Files 13-12
13.6 Viewing and Linking to Server Cluster Members 13-13
13.7 Adding a Reports Server to OEM 13-14

14 Tuning Oracle9*i*AS Reports Services

14.1 Using the In-Process Server 14-1
14.2 Tuning the Reports Engine 14-2
14.2.1 initEngine 14-3
14.2.2 maxEngine 14-3

14.2.3 minEngine 14-4
14.2.4 engLife 14-4
14.2.5 maxIdle 14-4
14.2.6 callBackTimeOut 14-5
14.3 Clustering Multiple Servers 14-5
14.4 Optimizing Cache Strategies 14-6
14.4.1 Setting Up Cache in the Reports Server Configuration File 14-7
14.4.2 Specifying Cache-Related Options in the Command Line 14-8
14.4.2.1 TOLERANCE 14-8
14.4.2.2 EXPIRATION 14-8
14.4.3 Setting Up Caching Options in a JSP 14-9
14.5 Monitoring Performance 14-10
14.5.1 Monitoring Performance with Oracle Trace 14-11
14.5.1.1 Trace Overview 14-11
14.5.1.2 Additional Sources of Trace Information 14-12
14.5.2 The SHOWJOBS Command Keyword 14-13
14.5.3 Accessing the RW_SERVER_QUEUE table 14-14
14.5.4 Updating the Database with Queue Activity 14-17

Part V Appendices

A Command Line Arguments

A.1 Command Overview A-1
A.1.1 rwclient A-2
A.1.2 rwrun A-2
A.1.3 rwbuilder A-3
A.1.4 rwconverter A-3
A.1.5 rwservlet A-4
A.1.6 rwcgi A-5
A.1.7 rwserver A-6
A.2 Command Line Syntax A-6
A.3 General Usage Notes A-7
A.4 Command Line Arguments A-7
A.4.1 ACCESSIBLE A-7
A.4.2 ARRAYSIZE A-8

A.4.3 AUTHID A-8
A.4.4 AUTOCOMMIT A-9
A.4.5 AUTOSTART A-9
A.4.6 BATCH A-10
A.4.7 BCC A-10
A.4.8 BLANKPAGES A-11
A.4.9 BUFFERS A-12
A.4.10 CACHELOB A-12
A.4.11 CC A-13
A.4.12 CELLWRAPPER A-14
A.4.13 CMDFILE A-15
A.4.14 CMDKEY A-16
A.4.15 CONTENTAREA A-16
A.4.16 COPIES A-17
A.4.17 CUSTOMIZE A-18
A.4.18 DATEFORMATMASK A-19
A.4.19 DELAUTH A-19
A.4.20 DELIMITED_HDR A-20
A.4.21 DELIMITER A-20
A.4.22 DESFORMAT A-21
A.4.23 DESNAME A-22
A.4.24 DEST A-23
A.4.25 DESTINATION A-24
A.4.26 DESTYPE A-25
A.4.27 DISTRIBUTE A-26
A.4.28 DTYPE A-27
A.4.29 DUNIT A-28
A.4.30 EXPIRATION A-29
A.4.31 EXPIREDAYS A-30
A.4.32 EXPRESS_SERVER A-30
A.4.33 FORMSIZE A-33
A.4.34 FROM A-33
A.4.35 GETJOBID A-34
A.4.36 GETSERVERINFO A-35
A.4.37 HELP A-35

A.4.38 IGNOREMARGIN A-36
A.4.39 INSTALL A-36
A.4.40 ITEMTITLE A-37
A.4.41 JOBNAME A-37
A.4.42 JOBTYPE A-38
A.4.43 KILLJOBID A-38
A.4.44 LONGCHUNK A-39
A.4.45 MODE A-39
A.4.46 MODULE | REPORT A-40
A.4.47 NONBLOCKSQL A-41
A.4.48 NOTIFYFAILURE A-41
A.4.49 NOTIFYSUCCESS A-42
A.4.50 NUMBERFORMATMASK A-42
A.4.51 ONFAILURE A-43
A.4.52 ONSUCCESS A-44
A.4.53 ORIENTATION A-44
A.4.54 OUTPUTFOLDER A-45
A.4.55 OUTPUTPAGE A-46
A.4.56 OVERWRITE A-47
A.4.57 P_AVAILABILITY A-47
A.4.58 P_DESCRIPTION A-48
A.4.59 P_FORMATS A-48
A.4.60 P_NAME A-49
A.4.61 P_OWNER A-49
A.4.62 P_PFORMTEMPLATE A-50
A.4.63 P_PRINTERS A-50
A.4.64 P_PRIVILEGE A-51
A.4.65 P_SERVERS A-52
A.4.66 P_TRIGGER A-52
A.4.67 P_TYPES A-53
A.4.68 PAGEGROUP A-53
A.4.69 PAGESIZE A-54
A.4.70 PAGESTREAM A-55
A.4.71 PARAMFORM A-56
A.4.72 PARSEQUERY A-56

A.4.73 PDFCOMP ... A-57
A.4.74 PDFEMBED ... A-57
A.4.75 PRINTJOB ... A-58
A.4.76 READONLY ... A-59
A.4.77 REPLACEITEM ... A-59
A.4.78 REPLYTO ... A-60
A.4.79 REPORT|MODULE ... A-61
A.4.80 ROLE ... A-61
A.4.81 RUNDEBUG ... A-61
A.4.82 SAVE_RDF ... A-62
A.4.83 SCHEDULE ... A-62
A.4.84 SERVER ... A-63
A.4.85 SHOWENV ... A-64
A.4.86 SHOWJOBS ... A-65
A.4.87 SHOWMAP ... A-65
A.4.88 SHOWMYJOBS ... A-66
A.4.89 SHUTDOWN ... A-66
A.4.90 SITENAME ... A-67
A.4.91 SOURCE ... A-68
A.4.92 SSOCONN ... A-69
A.4.93 STATUSFORMAT ... A-70
A.4.94 STATUSFOLDER ... A-71
A.4.95 STATUSPAGE ... A-72
A.4.96 STYPE ... A-73
A.4.97 SUBJECT ... A-73
A.4.98 TOLERANCE ... A-74
A.4.99 TRACEFILE ... A-75
A.4.100 TRACEMODE ... A-76
A.4.101 TRACEOPTS ... A-76
A.4.102 UNINSTALL ... A-78
A.4.103 URLPARAMETER ... A-78
A.4.104 USERID ... A-79
A.4.105 WEBSERVER_DEBUG ... A-80
A.4.106 WEBSERVER_DOCROOT ... A-80
A.4.107 WEBSERVER_PORT ... A-81

B Reports-Related Environment Variables

C Batch Registering Reports in Oracle9iAS Portal

C.1 Batch Registering Report Definition Files C-1
C.1.1 Run RWCONVERTER to Generate a SQL Script C-1
C.1.2 Run the Script in SQL*Plus C-4
C.2 Batch Removing Report Packages C-5
C.3 PL/SQL Batch Registering Function C-5

Index

Send Us Your Comments

Oracle9*i*AS Reports Services Publishing Reports to the Web, Release 9.0

Part No. A92102-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. 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?

If you find errors or have suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us via the Reports discussion group forum on the Oracle Technology Network: http://otn.oracle.com

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

Preface

This manual describes the different options available for publishing reports with Oracle9*i*AS Reports Services as well as how to configure the Oracle9*i*AS Reports Services software for publishing reports.

> **Note:** For the latest updates to *Oracle9iAS Reports Services Publishing Reports to the Web*, refer to the Oracle Technology Network (http://otn.oracle.com/products/reports/), then click Getting Started and use the index to navigate to *Oracle9iAS Reports Services Publishing Reports to the Web*.

Intended Audience

This manual is intended for anyone who is interested in publishing reports with Oracle9*i*AS Reports Services. To configure Oracle9*i*AS Reports Services, it will be useful for you to have a solid understanding of the following technologies:

- your operating system
- Java
- databases
- CORBA
- JSP files
- XML and DTD files
- Web server configuration
- HTTP

This manual will guide you through configuring components related to these technologies.

Documentation Accessibility

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

Accessibility of Code Examples in Documentation

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

Structure

This manual contains the following chapters:

Chapter 1	Provides an overview of Oracle9*i*AS Reports Services architecture.
Chapter 2	Tells you how to start and stop Oracle9*i*AS Reports Services.
Chapter 3	Describes how to configure the Oracle9*i*AS Reports Services.
Chapter 4	Explores how Oracle9*i*AS Reports Services handles output processing to default and custom destinations.
Chapter 5	Describes how to control user access to reports with Oracle9*i*AS Portal.
Chapter 6	Describes how to cluster Reports Servers to enhance performance and reliability.

Chapter 7	Provides an overview of single sign-on architecture and process flow and offers tips for getting user resource information into the Oracle Internet Directory.
Chapter 8	Describes the various methods for running reports, in particular, how to construct a runtime URL.
Chapter 9	Describes how set up advanced distributions via a distribution XML file.
Chapter 10	Provides information about customizing reports at runtime with XML.
Chapter 11	Describes how to use the event-driven publishing to invoke reports automatically in response to database triggers.
Chapter 12	Provides NLS information for Oracle9*i*AS Reports Services.
Chapter 13	Describes Oracle9*i*AS Reports Services integration with Oracle Enterprise Manager (OEM) and tells you how you can monitor your Reports Servers with OEM.
Chapter 14	Provides tips for tuning and performance enhancements.
Appendix A	Provides information about Reports-related command line arguments.
Appendix B	Provides information about Oracle9*i*AS Reports Services environment variables.

Related Documents

For more information on building reports, Oracle9*i*AS Portal, or Oracle9*i*AS Reports Services, refer to the following manuals:

- *Oracle9i Reports Developer Tutorial, A90900-01*
- *Oracle9i Reports Developer Building Reports, A92101-01*
- *Oracle9iAS Portal Getting Started*, accessed through the Oracle Technology Network (http://portalcenter.oracle.com)
- *Oracle9i Reports Developer Getting Started*, accessed through the Oracle Technology Network (http://otn.oracle.com/products/reports/)

Notational Conventions

The following conventions are used in this book:

Convention	Meaning
boldface text	Used for emphasis. Also used for menu items, button names, labels, and other user interface elements.
italicized text	Used to introduce new terms as well as to indicate configuration elements and their related attributes.
`courier font`	Used for path and file names and for code and text that you type.
CAPS	Used for environment variables, command line keywords, and built-ins and package names on an NT platform. The UNIX platform uses lower case.

Part I

Preparing Your Environment

Part I contains overview information about the Oracle9*i*AS Reports Services environment and provides practical information about preparing that environment for running reports. This includes starting and stopping Oracle9*i*AS Reports Services, configuring Reports-related Oracle9*i*AS components, securing your environment through Oracle9*i*AS Portal, clustering multiple Reports Servers, and setting up data source single sign-on.

Part I includes the following chapters:

- Chapter 1, "Oracle9iAS Reports Services Architecture"
- Chapter 2, "Starting and Stopping Oracle9iAS Reports Services"
- Chapter 3, "Configuring Oracle9iAS Reports Services"
- Chapter 4, "Configuring Destinations for Oracle9iAS Reports Services"
- Chapter 5, "Controlling User Access"
- Chapter 6, "Reports Server Clusters"
- Chapter 7, "Data Source Single Sign-On"

1

Oracle9iAS Reports Services Architecture

When you're ready to publish your reports, all the Web server and application server tools you'll need are available in the Oracle9*i* Application Server (Oracle9*i*AS). This chapter describes the architecture of relevant Oracle9*i*AS components in combination with its reports publishing component: Oracle9*i*AS Reports Services. It also provides an overview of reports runtime processing and offers some things to consider when you set up your server environment.

This chapter includes the following sections:

- Overview of Oracle9iAS Reports Services
- Oracle9iAS Reports Services Components
- Oracle9iAS Reports Services Runtime Process
- Things to Consider When You Set Up Your System

1.1 Overview of Oracle9*i*AS Reports Services

Oracle9*i*AS is a comprehensive and integrated application server that runs any Web site, portal, or Internet application. It enables you to make applications available from Web browsers, mobile devices, and command lines. Oracle9*i*AS Reports Services is the reports publishing component of Oracle9*i*AS. It is an enterprise reporting service for producing high quality production reports that dynamically retrieve, format, and distribute any data, in any format, anywhere. You can use Oracle9*i*AS Reports Services to publish in both Web-based and non-Web-based environments.

Oracle9*i*AS Reports Services provides a scalable, flexible architecture for the distribution and automated management of report generation engines on the same server and across multiple servers. Additionally, it caches report outputs for reuse on similar requests. It integrates into standard Web environments with JSPs, Java

Servlets, and CGI. It enables you to run reports on both local and remote application servers and to implement a multi-tiered architecture for running your reports.

When used in conjunction with servlet, JSP, or CGI (maintained only for backward compatibility), Oracle9*i*AS Reports Services enables you to run reports on any platform from a Web browser using a standard URL syntax. For servlet implementations, the in-process server is available for faster response and easier administration. The in-process server cuts down on the communication expense between processes and consequently increases response times.

Oracle9*i*AS Reports Services handles client requests to run reports by entering all requests into a job queue. When one of the server's engines becomes available, the next job in the queue is dispatched to run. As the number of jobs in the queue increases, the server can start more engines until it reaches the maximum limit specified in your server configuration. Similarly, engines are shut down automatically after having been idle for a period of time that you specify (see Chapter 3, "Configuring Oracle9iAS Reports Services").

Oracle9*i*AS Reports Services keeps track of all jobs submitted to the server, including jobs that are running, scheduled to run, finished, or failed. The Reports Queue Manager (Windows), the Reports Queue Viewer (UNIX), the `showjobs` command (Web), and the Reports Services pages in Oracle Enterprise Manager (OEM) enable you to view information on when jobs are scheduled, queued, started, finished, and failed, as well as the job output and the final status of the report.

With Oracle9*i*AS Reports Services, job objects are persistent. This means that if the server is shut down then restarted, all jobs are recovered,[1] not just scheduled jobs.

When used in a Web environment, the Oracle9*i*AS Reports Services architecture consists of four tiers:

> **Note:** The term *tier* refers to the logical location of the components that comprise the Oracle9*i*AS Reports Services architecture. Each of the tiers, though, could reside on the same or different machines.

- The client tier (a Web browser)
- The Web server tier

[1] Only synchronous jobs and jobs that are currently running are lost in this case.

- The Oracle9*i*AS Reports Services tier
- The data tier, including databases and all other data sources

When used in a non-Web environment, there are three tiers (a Web server being unnecessary):

- The client tier (a small, proprietary application on each client machine)
- Oracle9*i*AS Reports Services tier
- The data tier, including databases and pluggable data sources

The way you set up these tiers can range from having all of them on one machine to having each of them on a separate machine. Additionally, you can have multiple Web servers on multiple machines as well as multiple application servers on multiple machines.

If you choose to have your Web server on multiple machines, the Oracle9*i*AS HTTP Server provides a load balancing feature to allow sharing of the Web server load across multiple machines. If you choose to have your application server on multiple machines, Oracle9*i*AS Reports Services provides peer-level clustering to allow sharing of the Reports Server load among multiple machines.

The difference between load balancing and peer clustering is that with load balancing, one machine manages the traffic across all machines; while with peer clustering, all machines are aware of the traffic on each machine, and each machine shares the task of monitoring and responding to requests. The advantage of peer-level clustering is the elimination of a single point of failure, further supporting the possibility of a fail-safe environment.

Note: Reports Server clustering is discussed in detail in Chapter 6, "Reports Server Clusters".

Oracle9*i*AS Reports Services provides event-based reporting. This uses database events to trigger the generation of a report. For example, you can define an event that signals a change in revenue levels above or below a particular watermark. If the change occurs in the database (the event), a report is automatically generated. This feature is discussed in detail in Chapter 11, "Event-Driven Publishing".

Oracle9*i*AS Reports Services includes a distribution module that uses XML to define unique configurations for the distribution of reports. Call the desired XML file from the runtime command line or URL to generate one report, and let the server handle diverse outputs and destinations. Processing time is significantly reduced and

configuration changes can all be handled within the XML file. This feature is discussed in detail in Chapter 9, "Creating Advanced Distributions".

1.2 Oracle9*i*AS Reports Services Components

Figure 1–1 Oracle9iAS Reports Services Components

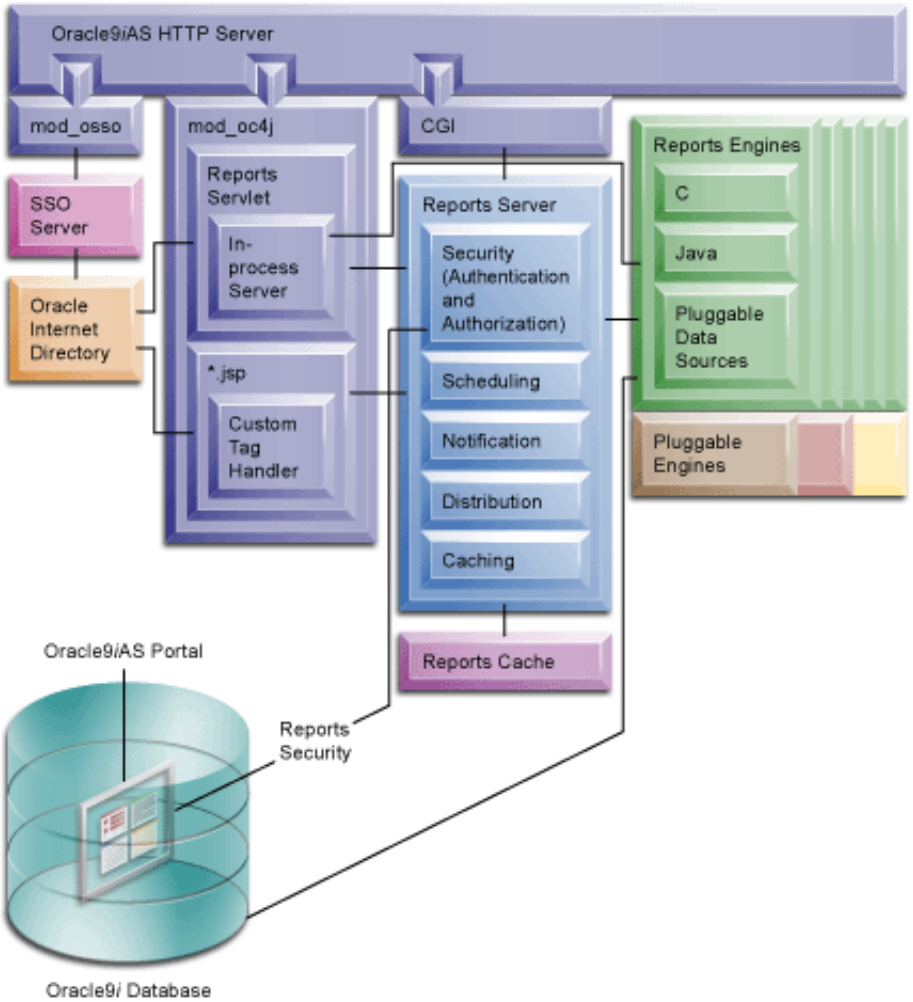


Figure 1–1 illustrates the components of a working Oracle9*i*AS Reports Services environment. This includes:

1. The **Oracle9*i*AS HTTP Server**:

   Oracle9*i*AS provides a Web server called the Oracle9*i*AS HTTP Server. It incorporates an OpenSSL module to provide support for Secure Sockets Layer (SSL) and HTTP Secure Sockets Layer (HTTPS). It also provides a servlet engine to support the running of Java Servlet applications.

2. The module **mod_oc4j**. This is used by the Oracle9iAS HTTP Listener to redirect requests from servlets and JSPs to Oracle9*i*AS Containers for Java 2 Enterprise Edition (OC4J). OC4J provides a complete J2EE environment that includes a JSP translator, a JSP servlet engine (OJSP), and an Enterprise JavaBeans (EJB) container. It provides a fast, lightweight, highly scalable, easy-to-use, complete J2EE environment. It is written entirely in Java and executes on the standard Java Development Kit (JDK) Virtual Machine (JVM).

3. The **Reports Servlet** is a component of Oracle9*i*AS Reports Services that runs inside of the Web server's servlet engine; the Reports Servlet translates and delivers information between HTTP and the Reports Server. It includes:

   - The **In-Process Server**, which reduces the maintenance and administration of the Reports Server by providing a means for starting the server automatically, whenever it receives the first request from the client via the Reports Servlet (rwservlet) or a Reports JSP.
   - The **Custom Tag Handler**, which processes custom Reports tags included in a JSP file. (In a JSP file, Reports-related custom tags are identified by the prefix rw:; other custom tags using other prefixes may also be present).

4. The **Reports CGI** component of the Web server translates and delivers information between HTTP and the Reports Server.

   Note: Reports CGI is maintained only for backward compatibility.

5. The **Reports Server** processes client requests, which includes ushering them through its various services, such as authentication and authorization checking, scheduling, caching, and distribution (including distribution to custom—or pluggable—output destinations). Reports Server also spawns runtime engines for generating requested reports, fetches completed reports from the Reports cache, and notifies the client that the job is ready.

6. The **Reports Cache** securely stores completed job outputs.

7. The **Reports Engine** includes components for running SQL- and Pluggable Data Source-based reports. It fetches requested data from the data source, formats the report, sends the output to cache, and notifies the Reports Server that the job is complete.
8. The **Pluggable Engines** are custom engines that use Java APIs to pass jobs to the Reports Server, as well as leverage the server's other features, such as scheduling, distribution, notification, and caching. Oracle9*i*AS Reports Services provides an out-of-the-box pluggable engine called the URL engine. The URL engine enables you to distribute content from any publicly available URL to such destinations such as e-mail, Oracle9*i*AS Portal, and WebDav.

1.3 Oracle9*i*AS Reports Services Runtime Process

Here is how the various components of Oracle9*i*AS Reports Services contribute to the process of running a report:

1. The client requests a report by contacting a server through either a URL (Web) or a non-Web, Reports-related command, such as `rwclient`.
   - The URL goes to JSP, rwservlet, or CGI, all associated with the Oracle HTTP Listener. The JSP and rwservlet requests go to mod_oc4j. (For jobs run as JSPs, mod_oc4j uses OJSP to translate the JSP into a servlet.) The CGI requests go to a CGI component.

     The URL may contain runtime parameters or a keyword that refers to a runtime parameter configuration section within cgicmd.dat, or it may contain both, though parameters explicitly named in the URL must not also be present in the relevant keyword section of cgicmd.dat.
   - rwclient goes directly to the Reports Server.

     The command line may contain runtime parameters. If you have a lot of runtime parameters, you can create a batch file or shell script that contains the `rwclient` command along with a string of parameters.
2. The rwservlet or the Reports CGI (rwcgi, maintained only for backward compatibility) component translates requests between HTTP and the Reports Server and sends requests to the Reports Server:
   - Server requests from JSP or using rwservlet can by run by the in-process Reports Server or as a stand-alone Reports Server process, whichever is specified in the servlet configuration file, rwservlet.properties (*`ORACLE_HOME`*`\reports\conf\`). An in-process server requires less maintenance than a stand-alone server because, unlike the stand-alone server, it starts

automatically in response to requests from the client. Additionally, an in-process server cuts down on the communication between processes, increasing the potential for faster performance.

- Server requests using rwcgi go to the stand-alone server.

3. The Reports Server processes the request:

   If the request includes a TOLERANCE argument, then the Reports Server checks its cache to determine whether it already has output that satisfies the request. If it finds acceptable output in its cache, then it immediately returns that output rather than rerunning the report.

   Note: For any job request that you send to the Reports Server, you can include a tolerance argument. Tolerance defines the oldest output that the requester would consider acceptable. For example, if the requester specified five minutes as the tolerance, the Reports Server would check its cache for the last duplicate report output that had been generated within the last five minutes. An expiration argument defines the point in time when the report output should be deleted from the cache (for example, expiration might equal a specific date and time for when the output should expire). For more information, see Appendix A, "Command Line Arguments".

   If the request is the same as a currently running job, then the request will reuse the output from the current job rather than rerunning the report.

   If neither of these conditions is met, the Reports Server processes the request:

   a. If configured, the Reports Server initiates an authentication and authorization check through mod_osso, part of the Oracle HTTP Server.

   b. If the report is scheduled, the Reports Server stores the request in the scheduled job queue, and the report is run according to schedule. If the report is not scheduled, it is queued in the current job queue for execution when a Reports Engine becomes available.

> **Note:** When you configure the Reports Server (in <*server_name*>.conf), you can specify the maximum number of Oracle9*i*AS Reports Engines it can use. If the Oracle9*i*AS Reports Server is under this maximum, then it can send the job to an idle engine or start a new engine to handle the request. Otherwise, the request must wait until one of the current Oracle9*i*AS Reports Engines completes its current job.

   c. At runtime, the Reports Server spawns a Reports Engine and sends the request to that engine to be run.

4. The Reports Engine retrieves and formats the data.
5. The Reports Engine populates the Reports cache.
6. The Reports Engine notifies the Reports Server that the report is ready.
7. The Reports Server accesses the cache and sends the report to output according to the runtime parameters specified in either the URL, the command line, or the keyword section in the cgicmd.dat file (URL requests only).

Another way to create a report is through event-driven publishing. With event-driven publishing, the client is the database (rather than the end user). Events are defined through the Event-Driven Publishing API. The event invokes a database trigger, an advanced queuing application, or a PL/SQL package that calls the Event-Driven Publishing API to submit jobs to the Reports Server. Event-Driven Publishing is discussed in detail in Chapter 11, "Event-Driven Publishing".

1.4 Things to Consider When You Set Up Your System

The way you set up Oracle9*i*AS Reports Services can vary widely depending upon the requirements of your system. Before you set up Oracle9*i*AS Reports Services, you must make some decisions based upon your requirements. By making these decisions beforehand, you can greatly simplify the set-up process.

The following subsections discuss some of the decisions involved in:

- Choosing the Types of Requests You Will Service
- Choosing Servlet, JSP, or CGI
- Choosing Single- Or Multiple-Machine Configurations
- Choosing Whether to Cluster Multiple Servers

1.4.1 Choosing the Types of Requests You Will Service

Oracle9*i*AS Reports Services can be configured to accept both Web and non-Web job requests.

In the Web case, you can run reports by clicking or typing a URL in a Web browser. Depending on the URL, the report output is then served back to you in your browser or sent to a specified destination (for example, a printer). To enable users to launch reports from a browser, you will use either the Oracle9*i*AS Reports Servlet, a JSP, or Oracle9*i*AS Reports CGI components with your Web server. One or the other of these components must be present on the Web server to enable communications between it and the Oracle9*i*AS Reports Server and to enable the processing of report requests from Web clients.

Note: For more information, refer to the Section 1.4.2, "Choosing Servlet, JSP, or CGI".

In the non-Web case, you can send job requests using the rwclient executable, installed on each of your user's machines.

From the perspective of configuration, these are the key differences between enabling Web and non-Web requests:

- Enabling Web requests requires that you choose between Reports Servlet, JSP, or the Reports CGI (maintained only for backward compatibility) for the server side, but eliminates the need to install any client software beyond a standard Web browser.
- Enabling non-Web requests requires that you install client software on each machine that will be used to run requests. This introduces the need to administer client software on each client machine.

The Web case is clearly the most cost effective because it reduces client maintenance costs. But there might be cases where launching non-Web requests is a necessity. Oracle9*i*AS Reports Services supports the implementation of both Web and non-Web requests in a single deployment environment.

1.4.2 Choosing Servlet, JSP, or CGI

To use Oracle9*i*AS Reports Services in a Web environment, you must use a servlet, JSP, or CGI implementation. Our recommendation is that you choose servlet or JSP. The CGI implementation in Oracle9*i*AS Reports Services is being maintained only for backward compatibility.

Between servlet and JSP there are additional considerations. A JSP-only implementation means that you can publish a layout that is optimized for Web delivery. The servlet enables you to include paper layouts in your report publishing solution and fully leverage the distribution features of Oracle9*i*AS Reports Services.

Using the servlet does not imply that you cannot also use JSP files because JSP files can contain both Web and paper layouts. When you run a report stored in a JSP, you specify the servlet in the URL and call the JSP with the command line argument: `report=<myreport>.jsp`.

For more information on running reports, see Chapter 8, "Running Report Requests".

1.4.3 Choosing Single- Or Multiple-Machine Configurations

You can place Oracle9*i*AS Reports Services on the same machine as your Web server or on a different machine. Both scenarios have pros and cons.

For example, while it's true that having Oracle9*i*AS Reports Services and the Web server on the same machine requires more of the machine's memory and disk space, it's also true that such an implementation reduces network traffic. This is because requests traveling between the Web server and the application server do not have to travel across a network, only incoming requests must do so.

If you are using the in-process server (available only with servlet implementations) you can further amplify the performance advantages of a single machine. The in-process server speeds up processing time by allowing for faster and more efficient communication between Oracle9*i*AS Reports Services components. We recommend that you use the in-process server unless you will not use the Reports Servlet to deploy reports.

On the other hand, if you have a single machine configuration and that machine fails, everything fails.

While there is a greater amount of network traffic when the Web server and the application server are on different machines, you also benefit from the increase in system resources, in the form of additional CPUs, more disk space, and more available memory. Even in a multiple machine configuration, the in-process server will aid performance by speeding communication between Oracle9*i*AS Reports Services components

Another possibility is placing your Web server and your application server each on multiple machines. This will require additional configuration, but it allows you to implement load balancing on the Web server.

If you will be deploying reports in multiple languages, you'll want to set up multiple Reports Servers: one or more for each language.

1.4.4 Choosing Whether to Cluster Multiple Servers

A cluster is a virtual grouping of servers into a community for the purpose of sharing request processing efficiently across members of the cluster. Unlike in previous versions, clustering in Oracle9*i*AS Reports Services is peer-level, rather than master/slave. Peer-level clustering means that all members of the cluster take equal responsibility for sharing and processing incoming requests. Incoming requests are sent to the cluster as a whole rather than any one Reports Server in the cluster. Thus, if one member is shut down, the other members carry on managing the request load. There is no single-point-of-failure, where one machine's malfunction brings the whole system down.

Each cluster member machine must be configured in more or less the same way to allow a report to run on each server member in the same way. This means that configuration files should have most of the same settings: a distinction can be drawn between job-related settings and machine-related settings. Job-related settings must be the same from cluster member to cluster member. Job-related settings include settings related to security, data sources, and destination types. Machine-related settings include such attributes as *maxEngine*, *minEngine*, *maxIdle*, *initEngine*, and the like—these can be different from member to member.

Additionally, for cluster members:

- Server-related environment variables should be set to the same values.
- TNS settings should point to the same databases in the same way.

For servers to be members of the same cluster, they must share a cluster name (appended to each server's server name) and have the same public and private keys.

If your machines require different job-related configuration settings, you will not benefit from clustering.

If you must set your servers up for different languages, you'll want to set up multiple clusters: one or more for each language.

2

Starting and Stopping Oracle9*i*AS Reports Services

This chapter provides information on starting and stopping Oracle9*i*AS Reports Services. It includes the following main sections:

- Starting the Reports Server
- Verifying the Reports Servlet and Server Are Running
- Verifying that the Oracle HTTP Server Is Running
- Stopping the Reports Server

> **Note:** The examples in this chapter use *ORACLE_HOME* to denote where the Oracle9*i* Application Server is installed. This includes Oracle9*i*AS Reports Services.

If you plan to run reports on the Web, you must first start the Oracle HTTP Server. You'll find information on doing this in your Oracle9*i*AS documentation. When you follow any of the procedures in this chapter, we assume you have already started the Oracle HTTP Server.

2.1 Starting the Reports Server

You have these options for running the Reports Server:

- Installing and Starting the Reports Server as a Service (Windows NT/2000)
- Starting the Reports Server as a Servlet (Windows and UNIX)
- Starting the Reports Server from a Command Line (Windows and UNIX)

The following subsections tell you how to set up each of these options.

2.1.1 Installing and Starting the Reports Server as a Service (Windows NT/2000)

By default, the Reports Server is installed as an in-process server, but, if you wish, you can install the Reports Server as a service on a Windows NT/2000 machine. To do so, at the command prompt enter:

```
rwserver -install <server_name> [batch=yes/no] [autostart=yes/no]
```

For *batch*, the default is *no*. Enter *yes* if you don't want to be prompted for confirmation during installation. For *autostart*, the default is *no*. Enter *yes* if you want the service to start automatically at reboot without requiring a user to logon and manually start the Reports Server.

Add the cluster name to this command if this server will be a member of a cluster. For example:

```
rwserver -install <server_name>.<cluster_name> [batch=yes/no] [autostart=yes/no]
```

To learn more about clustering servers together, see Chapter 6, "Reports Server Clusters".

Note: To remove the Reports Server NT/2000 service, at a command prompt enter: `rwserver -uninstall <server_name>`. Include the cluster name if the server is a member of a cluster, for example: `rwserver -uninstall <server_name>.<cluster_name>`.

To start your Reports Server on Windows NT/2000:

1. On the machine that hosts the Reports Server, choose **Start > Settings > Control Panel** and double-click **Services** in the Control Panel folder.
2. In the Services dialog box, choose **Oracle Reports Server [*repserver*]**, where <*repserver*> is the name of the Reports Server instance, and click **Startup**. The Services dialog window displays.
3. In the Services dialog window, select **This Account** in the **Log On As** section, and select an operating system user name and password. This specifies the user account under which the server process is run.

> **Note:** If you want to output to Postscript or to a printer, then ensure the user running the Reports Server service has access to a default printer. Do this by using a specific, real user who has printer access when you set up the **Log On As** section of your NT/2000 service. Typically, the System Account does not have access to printers.
>
> For that matter, the user running the Reports Server service must have access to anything the server may need. For example, the server may need write access to another drive.

4. Set the **Startup Type** of the service to **Automatic** when the system is started.
5. Click **OK**.
6. Click **Start**.

   A **Service Control** message box indicates when your Reports Server has started.

2.1.2 Starting the Reports Server as a Servlet (Windows and UNIX)

If you are using the Reports Server as an in-process server (the default configuration), just sending a request starts up the servlet; however, if you are sending a request via a command line, the servlet must be invoked via a URL first. When you have successfully started the servlet, this also means you have successfully started the HTTP Server.

To start the Reports Servlet from a URL, enter the following from your Web browser:

```
http://<your_machine_name>:<your_port_num>/reports/rwservlet
```

2.1.3 Starting the Reports Server from a Command Line (Windows and UNIX)

You can also start the Reports Server as a stand-alone server on Windows NT/2000 using the following command:

```
rwserver server=<server_name>
```

Add the BATCH command line keyword to start up the server without displaying dialog boxes or messages.

```
rwserver server=<server_name> batch=yes
```

You can run this command on UNIX using the following syntax:

```
rwserver.sh server=<server_name>
```

Or:

```
rwserver.sh server=<server_name> batch=yes
```

You can run this command from any directory as long as the executable can be reached in your PATH environment variable.

2.2 Verifying the Reports Servlet and Server Are Running

To verify that the Oracle9*i*AS Reports Servlet is running, navigate to the following URL:

```
http://<your_machine_name>.<domain_name>:<your_port_
number>/reports/rwservlet/help
```

Note that the URL is case sensitive. If this URL executes successfully, you should get a help page describing the rwservlet command line arguments.

To verify that the Reports Server is running, navigate to the following URL:

```
http://<your_machine_name>.<domain_name>:<your_port_
number>/reports/rwservlet/showjobs?server=<server_name>
```

The `server=<server_name>` argument is not required if you are using the default Reports Server name (`rep_<machine_name>`) or the Reports Server specified in the servlet configuration file, rwservlet.properties (*ORACLE_HOME*`\reports\conf\`). If this URL executes successfully, you should see a listing of the job queue for the specified Reports Server.

Note: You'll find more information about the servlet configuration file in Chapter 3, "Configuring Oracle9iAS Reports Services".

2.3 Verifying that the Oracle HTTP Server Is Running

To verify that the Oracle HTTP Server is running, in your browser, navigate to the following URL:

```
http://<server_name>.<domain>:<port_number>/
```

2.4 Stopping the Reports Server

This section discusses how to stop the Reports Server on Windows NT/2000 and UNIX.

- If the Reports Server is running on NT/2000 as a service, stop it through the Services control panel.
- If the Reports Server running on NT/2000 through a batch file, or on UNIX through a shell script, click the **Shutdown** button in the Oracle Reports Server dialog box.
- Launch Oracle Enterprise Manager, and navigate to the Reports Server you wish to shut down; click the **Stop** button on the selected Reports Server's home page. For more information about Reports and Oracle Enterprise Manager, see Chapter 13, "Managing and Monitoring Oracle9iAS Reports Services".
- If the Reports Server is running as an in-process server via the Reports Servlet, shut down OC4J through the Oracle Program Manager (OPMN). See the Oracle9*i*AS documentation for more information on OPMN.
- If the Reports Server running from a command line on NT/2000 or UNIX, at the command prompt enter the following command:

  For **Windows NT/2000** and **UNIX** (on UNIX use rwserver.sh in lieu of rwserver):

  This shuts down the server normally (i.e., finishes pending jobs and then stops):

  ```
  rwserver server=<server> shutdown=normal authid=<admin/pword>
  ```

  This shuts down the server immediately (i.e., stops without finishing pending jobs):

  ```
  rwserver server=<server> shutdown=immediate authid=<admin/pword>
  ```

  This shuts down the server without displaying any related messages:

  ```
  rwserver server=<server> shutdown=normal authid=<admin/pword> batch=yes
  ```

The keywords used with the `rwserver` command are described in Appendix A, "Command Line Arguments".

3

Configuring Oracle9*i*AS Reports Services

When you install Oracle9*i* Application Server (Oracle9*i*AS), Reports Services is configured automatically for you. There will likely be adjustments you wish to make to customize your environment, but you will not be required to set up the entire environment, or even most of it.

This chapter is included largely for reference, should you wish to introduce customizations or have a better understanding of the default configuration. It lists services-related configuration files and describes in detail the content of most of them. It includes the following main sections:

- Oracle9iAS Reports Services Configuration Files
- Configuring the Oracle9iAS Reports Server
- Configuring the Reports Servlet
- Configuring the URL Engine

> **Note:** The examples in this chapter use *ORACLE_HOME* to denote where the Oracle9*i* Application Server (Oracle9*i*AS) is installed. Oracle9*i*AS includes Oracle9*i*AS Reports Services.

Another aspect of configuration is the setting of environment variables. These are set for you automatically during installation. For reference, environment variables are discussed in Appendix B, "Reports-Related Environment Variables".

3.1 Oracle9*i*AS Reports Services Configuration Files

This section identifies the various configuration files associated with Oracle9*i*AS Reports Services. In most cases, you can leave these files untouched. Because they

control many aspects of your server environment, you could put that environment at risk if you change a file in some unsupported way. Always keep a back-up of the current version of any configuration file you plan to change.

The configuration files associated with Oracle9*i*AS Reports Services relate to the Reports Server and the Reports Servlet. They are listed and described in Table 3–1:

> **Note:** The paths specified in Table 3–1 are the same for both Windows and UNIX environments, though they are expressed here using the Windows backslash convention (\).

Table 3–1 Oracle9iAS Reports Services Configuration Files

Component	Configuration File
Reports Server	***ORACLE_HOME*\reports\conf\<*server_name*>.conf** Use this XML file to define initial values for the Reports Cache, the Reports Engine, and security; to register valid destination types; to specify the information to be logged; and to set other server-related values. This file is automatically created when you start up the server. If you want to rename your server and wish to keep custom configuration settings you've entered into this file, you must first rename this file to the new server name, then rename the server. Otherwise, the server will create its own new default configuration file. You'll find more information about this file in the section Configuring the Oracle9iAS Reports Server.
Reports Server	***ORACLE_HOME*\reports\dtd\rwserverconf.dtd** This file contains data type definitions for <*server_name*>.conf and rwbuilder.conf elements and attributes. Data type definitions lists all elements allowed in an associated XML file, the attributes associated with those elements, and default values for those attributes. You'll find more information about this file in Reports Server Configuration Elements (rwserverconf.dtd).

Table 3–1 Oracle9iAS Reports Services Configuration Files

Component	Configuration File
Reports Server	***ORACLE_HOME*\reports\conf\rwbuilder.conf** Use this XML file to configure the Reports Server that runs in the same process as the Builder. All run requests must go through the Reports Server, meaning that the Reports Builder requires a server to run reports. The Reports Builder automatically starts a Reports Server to handle its requests. When you run a report from the Builder, this file provides the configuration for the in-process server instance that is invoked. Like the <*server_name*>.conf file, this file relies on the rwserverconf.dtd file for its data type definitions, though several elements do not apply, including the *compatible*, *persistFile*, *and security* elements. Because this file shares most configuration elements found in <server_name>.conf, you'll find the information you need for configuring this file in Configuring the Oracle9iAS Reports Server.
Reports Servlet	***ORACLE_HOME*\reports\conf\rwservlet.properties** Among other things, this file is where you specify the location and filename of the Reports key map file (cgicmd.dat) and specify whether you will use the Reports Servlet's in-process server. You'll find more information about this file in Configuring the Reports Servlet.

3.2 Configuring the Oracle9*i*AS Reports Server

The Reports Server component of Oracle9*i*AS Reports Services is configurable via the XML files <*server_name*>.conf and rwbuilder.conf, located in the following directory (on both Windows and UNIX):

ORACLE_HOME\reports\conf\<*server_name or* rwbuilder>.conf

Both files are supported by the rwserver.template file, which contains default server configuration values (*ORACLE_HOME*\reports\conf\rwserver.template on both Windows and UNIX).

The <*server_name*>.conf file is the default server configuration file. The rwbuilder.conf file configures the server instance used in-process by the Reports Builder. All run requests must go through the Reports Server, meaning that the Reports Builder requires a server to run reports. When you run a report from the Builder, the rwbuilder.conf file provides the configuration for the server instance that is invoked (an in-process server) when it runs in the same process as the Reports Builder. The <*server_name*>.conf and rwbuilder.conf files are nearly

identical. The only difference between them is that rwbuilder.conf does not use the *compatible*, *persistFile*, or *security* configuration elements, described later in this section, and <*server_name*>.conf does.

Both of these files are created automatically, under the following circumstances:

- The <*server_name*>.conf file is created the first time you start the server. It is based on the rwserver.template file.
- The rwbuilder.conf file is created the first time you run a report through the Reports Builder. It also is based on the rwserver.template file.
- After you rename the server, a new <*server_name*>.conf file is created the next time you start the server. The new configuration file is based on the default values present in the rwserver.template file. If you wish to retain the configuration associated with the old server name, you must rename your <*server_name*>.conf file to the new server name (<*new_server_name*>.conf), before starting the renamed server.
- If you delete one of these files, the deleted file is recreated the next time you start the server. The new file is based on the default values present in the rwserver.template file.

To explain the syntax and values allowed in these files we'll look at the rwserverconf.dtd file, located in the following directory (on both Windows and UNIX):

```
ORACLE_HOME\reports\dtd\rwserverconf.dtd
```

3.2.1 Reports Server Configuration Elements (rwserverconf.dtd)

The rwserverconf.dtd file provides the following elements for configuring the Reports Server:

- server
- compatible
- cache
- engine
- security
- destination
- job
- notification

- log
- jobStatusRepository
- trace
- connection
- queue
- persistFile
- identifier
- pluginParam

Note that these are XML elements, and XML is case sensitive.

Additionally, when you add any of these elements to the server configuration file (<*server_name*>.conf), you'll save yourself potential error messages from any XML editor you may use if you follow the order in which they are listed in the rwserverconf.dtd file located in *ORACLE_HOME*\reports\dtd\ on both Windows and UNIX. The configuration file will work even if you do not follow this order, but it will not work if you fail to follow the case specified in rwserverconf.dtd.

These elements along with their related attributes and sub-elements are discussed in the following subsections.

3.2.1.1 server

EXAMPLE

```
<server>
    [One or more configuration specifications]
</server>
```

REQUIRED/OPTIONAL

Required. You can have a maximum of one open and close *server* element in a given configuration file.

DESCRIPTION

The *server* element opens and closes the content area of the server configuration file. In terms of the file's hierarchy, all the other elements are subordinate to the *server* element.

3.2.1.2 compatible

EXAMPLE

```
<compatible version="6i"/>
```

REQUIRED/OPTIONAL

Optional. You can have a maximum of one *compatible* element in your server configuration file.

DESCRIPTION

The *compatible* element is available for backward compatibility with Reports 6*i* clients (RWCLI60, RWCGI60, RWQMU60.EXE, RWQM60.EXE, RWQV60.EXE). When *compatible* is set to 6i, the Reports Server will make use of an executable file, named rwproxy, that listens for requests from a 6*i* client and forwards them to a 9*i* server.

Compatible has one attribute: *version*, described in Table 3–2.

Table 3–2 Attributes of the compatible element

Attribute	Valid values	Description
version	6i	Setting *version* to 6i enables Reports 6*i* clients to run under Oracle9*i*AS. When *version* is set to 6i, versions earlier than 6i may also run under Oracle9*i*AS, but they are not certified to do so and are not supported by Oracle.

If you use the *compatible* element, you must also have an entry for the Reports Server in your tnsnames.ora file, as you would have had for the 6*i* version of the Reports Server. For example:

```
testsvr.world = (ADDRESS=
    (PROTOCOL=tcp)
    (HOST=testhost.us.oracle.com)
    (PORT=1949)
    )
```

You can bypass this requirement by turning compatibility off. To turn compatibility off, remove the *compatibile* element from the server configuration file.

Note: If *compatible* is set to `6i`, and you have a TNS entry for the Reports Server in your tnsnames.ora file, you should include the cluster name if the server is a member of a cluster. If you use a cluster name, you should exclude the default domain that was specified in the `sqlnet.ora` file. For example:

```
myserver.world (standalone server with default domain)
myserver.cluster1 (server part of cluster1)
myserver.cluster1.world (invalid entry)
```

3.2.1.3 cache

EXAMPLE

```
<cache class="oracle.reports.cache.RWCache”>
    <property name="cacheSize" value="50"/>
    <property name="cacheDir" value="D:\orawin\reports\server\cache"/>
</cache>
```

REQUIRED/OPTIONAL

Optional. You can have a maximum of one *cache* element in your server configuration file. If no cache element is specified, the default is used (oracle.reports.cache.RWCache).

DESCRIPTION

The *cache* element is available for specifying the Java class that defines the server's cache implementation. You can use the default cache Java class or develop your own implementation through the Oracle9*i*AS Reports Services Cache API.

Note: Look for upcoming information about Reports APIs on the Oracle Technology Network: http://otn.oracle.com.

The *cache* element has one attribute: *class*, described in Table 3–3.

Table 3–3 Attributes of the cache element

Attribute	Valid values	Description
class	*see the Description column*	**Default:** oracle.reports.cache.RWCache A fully qualified Java class that implements the oracle.reports.cache.Cache interface.

You can also enter from zero to multiple properties under the *cache* element. Properties are name/value pairs recognized and understood by the implementation class you register under *cache*. For example, if you use the default cache Java class that is provided with Reports Services, your configuration entry might look like this:

```
<cache class="oracle.reports.cache.RWCache″>
    <property name="cacheSize" value="50"/>
    <property name="cacheDir" value="D:\orawin\reports\server\cache"/>
</cache>
```

In the preceding example, *cacheSize* is measured in megabytes, and *cacheDir*, which points to the location of the cache, is specified for a Windows platform. On UNIX, use UNIX standards, for example:

```
<property name="cacheDir" value="/ORACLE_HOME/reports/server/cache"/>
```

Should your cache implementation require access to additional information specify the information in the *pluginParam* element.

3.2.1.4 engine

EXAMPLE

```
<engine id=″rwEng″ class="oracle.reports.engine.EngineImpl″ initEngine=″1″
maxEngine=″5″ minEngine=″1″ engLife=″50″ maxIdle=″15″ callbackTimeOut=″60000″>
    <property name="sourceDir" value="D:\orawin\reports\myReport"/>
    <property name=″tempDir″ value=″D:\orawin\reports\myTemp″/>
</engine>
```

REQUIRED/OPTIONAL

Required. You must have at least one *engine* element in your configuration file, and you can have more than one.

DESCRIPTION

The *engine* element identifies the fully qualified Java class that starts an engine and provides a number of attributes that set operational controls on the engine. You can use the default engine provided with Reports Services (oracle.reports.engine.EngineImpl) or develop your own implementation through the Oracle9*i*AS Reports Services Engine API. As an example of a custom engine, you may have developed an engine to execute an operating system command should an event occur in your database.

Note: Look for upcoming information about Reports APIs on the Oracle Technology Network: http://otn.oracle.com.

The *engine* element has several attributes, described in Table 3–4.

Table 3–4 Attributes of the engine element

Attribute	Valid values	Description
id	string	A keyword, unique within a given configuration XML file, that identifies a particular *engine* element. This can be a text string or a number, for example: `id="rwEng"`
class	*see the Description column*	**Default:** oracle.reports.engine.EngineImpl A fully qualified Java class that implements two interfaces: oracle.reports.engine.Engine and oracle.reports.engine.EngineInterface.
classPath	string	The directory path to the Java class specified in the *class* attribute. To specify the directory, use the conventions required by the server platform, for example: **Windows:** classPath="D:*ORACLE_HOME*\myEngine.jar" **UNIX:** classPath="*ORACLE_HOME*/myEngine.jar"
initEngine	number	**Default:** 1 The number of engines you want the Reports Server to start at initialization.

Table 3–4 Attributes of the engine element

Attribute	Valid values	Description
maxEngine	number	**Default:** 1 The maximum number of this type of engine that can run on the server.
minEngine	number	**Default:** 0 The minimum number of this type of engine that is maintained by the server.
engLife	number	**Default:** 50 The number of jobs the engine can run before the engine is terminated, and, if necessary, a new engine is started. This feature is available to thwart memory leaks.
maxIdle	number	**Default:** 30 The number of minutes of allowable idle time before the engine is shut down, provided the current number of engines is higher than *minEngine*. For example, if *minEngine* is 0, *maxIdle* is 30, and one engine has been running but unused for 30 minutes, that engine will shut down. If, under the same conditions, *minEngine* is 1, the active engine will not shut down, even if it has been idle for 30 minutes.
callbackTimeOut	number	**Default:** 60000 The number of milliseconds of allowable waiting time between when the server calls an engine and the engine calls the server back. If the machine that hosts the server is very fast, you can reduce this number for faster performance.

You can also enter from zero to multiple properties under the *engine* element. The only requirement is that they be name/value pairs recognized by the Java class that implements the Reports engine. For example, if you use the default engine Java class that is provided with Reports Services, your *engine* configuration entry might look like this:

```
<engine id="rwEng" class="oracle.reports.engine.EngineImpl" initEngine="1"
maxEngine="5" minEngine="1" engLife="50" maxIdle="15" callbackTimeOut="60000">
```

```
    <property name="sourceDir" value="D:\orawin\reports\myReport"/>
    <property name="tempDir" value="D:\orawin\reports\myTemp"/>
</engine>
```

In this example, *sourceDir* and *tempDir* are set up for a Windows environment (UNIX would be `sourceDir="ORACLE_HOME/reports/myReport"` and `tempDir="ORACLE_HOME/reports/myTemp"`). The *sourceDir* property identifies the default directory you will use for report definition files. It overrides path information specified in the REPORTS_PATH environment variable.

The *tempDir* property identifies the name and location of the temporary directory Reports Services will use for its temporary files. If this value is unspecified for a default engine, Reports Services will use the temporary directory specified in the REPORTS_TMP environment variable. If REPORTS_TMP is also not specified, Reports Services will use your operating system's default temporary directory.

The *classPath* attribute is not specified because this configuration uses the default engine class.

Should your engine require access to additional information, such as an outgoing mail server, specify the additional information in the *pluginParam* element.

3.2.1.5 security

EXAMPLE

```
<security id="rwSec" class="oracle.reports.server.RWSecurity">
    <property name="securityUserid" value="portal_id/portal_password@portal_
    schema" confidential="yes" encrypted="no"/>
</security>
```

REQUIRED/OPTIONAL

Optional. If you do not enter a *security* element in the configuration file, the Reports Server is not secure. You can have from zero to multiple *security* elements in your configuration file.

DESCRIPTION

The *security* element identifies the fully qualified Java class that controls server access. You can use the default security class provided with Reports Services, which relies on security features available through Oracle9*i*AS Portal (included with Oracle9*i*AS), or develop your own implementation through the Reports Server Security API.

Note: Look for upcoming information about Reports APIs on the Oracle Technology Network: http://otn.oracle.com.

S*ecurity* attributes are described in Table 3–5.

Table 3–5 Attributes of the security element

Attribute	Valid values	Description
id	string	A keyword, unique within a given configuration XML file, that identifies a particular *security* element. This can be a text string or a number, for example: `id="rwSec"`
class	*see the Description column*	**Default:** oracle.reports.server.RWSecurity A fully qualified Java class that implements the Reports Server Security Java interface (oracle.reports.server.Security). The default relies on security features available through Oracle9*i*AS Portal (included with Oracle9*i*AS).

You also have the option of entering multiple properties under the *security* element. The only requirement is that they be name/value pairs recognized by the Java class that implements Reports Server security. For example, if you use the default security Java class that is provided with Reports Services, your *security* configuration entry might look like this:

```
<security id=”rwSec” class="oracle.reports.server.RWSecurity”>
    <property name="securityUserid" value="portal_id/portal_password@portal_
    schema" confidential=”yes” encrypted="no"/>
</security>
```

In this example, connect information is provided to enable the Reports Server to access to Oracle9*i*AS Portal security features. The property attributes *confidential* and *encrypted* are available for encrypting the information within the property. Once the `confidential="yes"` and `encrypted="no"` attributes are entered, the property value will be encrypted automatically by the Reports Server after you restart the server. When you next open the configuration file, the password information will be scrambled, and `encrypted` will be set to `yes`. If you forget the password you entered in the configuration file, you can delete the property and reenter it with new values, making sure to set `encrypted` to `no`.

Should your security implementation require access to additional information specify the information in the *pluginParam* element.

When setting up Security in a clustered environment, each cluster member should use the same security policy to prevent users experiencing unexpected behavior.

3.2.1.6 destination

EXAMPLE

```
<destination destype="oraclePortal"
class="oracle.reports.server.DesOraclePortal">
    <property name="portalUserid" value="portal_id/portal_password@portal_
    schema" confidential="yes" encrypted="no"/>
</destination>
```

REQUIRED/OPTIONAL

Optional. If you do not enter a destination element in the server configuration file, the provided destination classes will be used (printer, mail, file, cache, and Oracle9*i*AS Portal—which is an exception in that it requires an entry in the server configuration file so that you may specify the userid and password the server will use to log in to the portal). You can have from zero to multiple *destination* elements in your server configuration file.

DESCRIPTION

Use the *destination* element to register destination types with the server. There is no need, with the exception of Oracle9*i*AS Portal, to register provided (default) destinations, such as printers, e-mail, files, or cache. You must register the destination types you create through the Oracle9*i*AS Reports Services Destinations API.

Note: Look for upcoming information about Reports APIs on the Oracle Technology Network: http://otn.oracle.com.

Configuring destinations is discussed in detail in Chapter 4, "Configuring Destinations for Oracle9iAS Reports Services".

Destination attributes are listed and described in Table 3–6.

Table 3–6 Attributes of the destination element

Attribute	Valid values	Description
destype	string	Identifies the destination type, for example: `destype="printer"`
class	*See the Description column*	A fully qualified Java class that is a subclass of the Reports Server Destination Java class (oracle.reports.server.Destination). Allowable values include: ▪ oracle.reports.server.DesMail ▪ oracle.reports.server.DesFile ▪ oracle.reports.server.DesPrinter ▪ oracle.reports.server.DesOraclePortal

You also have the option of entering multiple properties under the *destination* element. The only requirement is that they be name/value pairs recognized by the Java class that is a subclass of the Reports Server Destination Java class. For example:

```
<destination destype="oraclePortal"
class="oracle.reports.server.DesOraclePortal">
    <property name="portalUserid" value="portal_id/portal_password@portal_
    schema" confidential="yes" encrypted="no"/>
</destination>
```

In this example, the property provides connect information to enable the Reports Server to access Oracle9*i*AS Portal. The *confidential* and *encrypted* attributes are included to automatically invoke encryption on the *portalUserid* value the next time the Reports Server is started.

Should your destination implementation require additional information, specify the information in the *pluginParam* element.

3.2.1.7 job

EXAMPLE

```
<job jobType="report" engineId="rwEng" securityId="rwSec"/>
```

REQUIRED/OPTIONAL

Required. You must have at least one *job* element and can have more than one.

DESCRIPTION

The *job* element works in collaboration with the *engine* and *security* elements. Use *job* to identify a job type and specify which engine and which security implementation should be used with that type of job. For example, you may have developed an engine to execute an operating system command should an event occur in your database. Using Oracle9*i*AS Report Service's event-driven publishing API, you identify the event as a specific job type. When the event occurs, the job type information is sent to the Reports Server, which looks up the job type under the *job* element in its configuration file, and follows the direction provided in the element's attributes to the engine (and, if applicable, security implementation) specified for that type of job.

Attributes of the *job* element are listed and described in Table 3–7.

Table 3–7 Attributes of the job element

Attribute	Valid values	Description
jobType	string	**Default:** report Describes the type of job to be processed by the server. You can enter any type of job, as long as the Reports Server has an engine to process it.
engineId	ID reference	References the ID entered for the engine that will process this job type. Available IDs are specified under the *engine* element in the server configuration file using the *id* attribute. The *id* is a unique keyword (that you devise) within a given configuration XML file that identifies a particular engine.
securityId	ID reference	References the ID entered for the security mechanism that will be applied to this job type. Available IDs are specified under the *security* element in the server configuration file.

3.2.1.8 notification

EXAMPLE

```
<notification id="tellMe02" class="oracle.reports.server.MailNotify"/>
```

REQUIRED/OPTIONAL

Optional. If you do not enter a *notification* element in the configuration file, the notification function is disabled. You can have from zero to multiple *notification* elements in your configuration file.

DESCRIPTION

Use the *notification* element to specify a Java class that defines the type of notification that should be sent when a job succeeds or fails. You can use the default notification class, which provides for notification via e-mail, or design your own with the Reports Notification API.

> **Note:** Look for upcoming information about Reports APIs on the Oracle Technology Network: http://otn.oracle.com.

Attributes of the *notification* element are listed and described in Table 3–8.

Table 3–8 Attributes of the job element

Attribute	Valid values	Description
id	string	**Default:** mailNotify A keyword, unique within a given configuration XML file, that identifies a particular *notification* element. This can be a text string or a number, for example: `id="tellMe01"`
class	ID reference	**Default:** oracle.reports.server.MailNotify A fully qualified Java class that implements the Reports Server Notification Java class oracle.reports.server.Notification.

If you use the default e-mail notification implementation, use the *pluginParam* element to specify the outgoing SMTP mail server to be used to send the mail. Use the runtime commands `notifysuccess` and `notifyfailure` to specify the e-mail address where *notification* should be sent (for more information, see Appendix A, "Command Line Arguments"). For example, you can include these commands in your runtime URL:

`notifysuccess=<`*`recipient's e-mail address`*`>&notifyfailure=<`*`recipient's e-mail address`*`>`

With the default e-mail implementation, you can specify only one address for each type of *notification*. You can specify one or both types of *notification*. You can send *notification* each to the same address or each to a different addresses.

A *notification* element in the server configuration file might look like this:

```
<notification id="mailNotify" class="oracle.reports.server.MailNotify"/>
    <property name="succNoteFile" value="succnote.txt"/>
    <property name=failNoteFile value="failnote.txt"/>
```

With the default notification implementation, it's not necessary to specify a path to the success or failure text files, provided they're in the default location: *ORACLE_HOME*\reports\templates. Otherwise, enter the directory path along with the filenames according to the requirements of the platform that hosts the server.

3.2.1.9 log

EXAMPLE

```
<log option="allJobs"/>
```

REQUIRED/OPTIONAL

Optional. You can have a maximum of one *log* element in your server configuration file.

DESCRIPTION

The *log* element is available for backward compatibility. It invokes the generation and population of a Reports log file. The log file is automatically generated and stored in the following path (the path is the same for Windows and UNIX):

ORACLE_HOME\reports\logs*.log

The *log* element has one attribute: *option*, described in Table 3–9.

Table 3–9 Attributes of the log element

Attribute	Valid values	Description
option	allJobs succeededJobs failedJobs noJob	**Default:** noJob Describes the type of jobs to be included in the log, in addition to default server activities that are logged. Choose from: ▪ **allJobs**: all jobs will be logged ▪ **succeededJobs**: only jobs that ran successfully will be logged ▪ **failedJobs**: only jobs that failed will be logged ▪ **noJob**: no jobs will be logged

3.2.1.10 jobStatusRepository

EXAMPLE

```
<jobStatusRepository class="oracle.reports.server.JobRepositoryDB">
    <property name="repositoryConn" value="scott/tiger@ORCL" confidential="yes"
    encrypted="no"/>
</jobStatusRepository>
```

REQUIRED/OPTIONAL

Optional. You can have a maximum of one *jobStatusRepository* elements in your server configuration file.

DESCRIPTION

The *jobStatusRepository* element specifies the Java class that implements a job status repository. It provides an additional means (over the *persistFile* element) of storing job status information.

The *persistFile* is a binary file and, therefore, cannot be used to publish job status information within your application. The *jobStatusRepository* provides a means of including status information in your application by providing additional ways of storing it.

The default class, oracle.reports.server.JobRepositoryDB, stores information in a database. Use the Reports APIs to create your own implementation of the Reports Server Job Repository interface (oracle.reports.server.JobRepository) that stores information wherever you wish.

Note: Look for upcoming information about Reports APIs on the Oracle Technology Network: http://otn.oracle.com.

The *jobStatusRepository* element has one attribute: *class*, described in Table 3–10.

Table 3–10 Attributes of the jobStatusRepository element

Attribute	Valid values	Description
class	*see the Description column*	**Default:** oracle.reports.server.JobRepositoryDB A fully qualified Java class that implements the Reports Server Job Repository Java class (oracle.reports.server.JobRepository).

The *jobStatusRepository* element allows for zero or multiple properties for passing arguments into the repository. The only requirement is that the class you specify in the server configuration file must recognize the name/value pairs you introduce.

The *jobStatusRepository* element might look like this in your server configuration file:

```
<jobStatusRepository class="oracle.reports.server.JobRepositoryDB">
    <property name="repositoryConn" value="scott/tiger@ORCL" confidential="yes"
    encrypted="no"/>
</jobStatusRepository>
```

In this example, the value for the *repositoryConn* property is the login for access to the database that stores the repository. The *confidential* and *encrypted* attributes are used to invoke encryption on the login information once the Reports Server is restarted.

Should your job status repository require additional information, specify the information in the *pluginParam* element.

3.2.1.11 trace

EXAMPLE

```
<trace traceFile="neptune.trc" traceOpts="trace_prf|trace_dbg|trace_wrn"
traceMode="trace_append"/>
```

REQUIRED/OPTIONAL

Optional. You can have a maximum of one *trace* element in your server configuration file.

DESCRIPTION

Use the *trace* element to create a file for tracing your report's execution and to specify the objects and activities you want to trace. The *trace* element controls tracing only for the server and the engine.

Note: Tracing for the servlet and JSP are configured in the servlet configuration file, rwservlet.properties, discussed in Section 3.3. Tracing for an individual report can be built into the reports runtime command line, discussed in Appendix A, "Command Line Arguments".

Trace attributes are listed and described in Table 3–11.

Table 3–11 Attributes of the trace element

Attribute	Valid values	Description
traceFile	*.trc	**Default:** <*server_name*>.trc The filename of the trace file. If no path is specified, the trace file will be in the following directory on both Windows and UNIX: *ORACLE_HOME*/reports/logs/
traceOpts	see Table 3–12	**Default:** trace_all This attribute defines the activities that will be traced. You can have one or more traceOpt values. For example: <traceOpts="trace_prf|trace_brk"> Separate values with a vertical bar (|).Valid values are listed and described in Table 3–12.
traceMode	trace_replace trace_append	**Default:** trace_replace Defines whether new trace information will either overwrite the existing trace file (trace_replace), or be added to the end of the trace, leaving existing trace information intact (trace_append).

Table 3–12 Valid values for the traceOpts attribute

Value	Description
trace_prf	Logs server and engine profile
trace_brk	Lists debug breakpoints
trace_app	Logs information on all report objects
trace_pls	Logs information on all PL / SQL objects
trace_sql	Logs information on all SQL
trace_tms	Enters a timestamp for each entry in the trace file
trace_dst	Lists distribution lists Use this value to determine which report section was sent to which destination.
trace_log	Duplicates log information in your trace file If you have specified a *log* element in your server configuration file, in addition to using the *trace* element, this value will cause information that is sent to the log file to also be sent to the trace file.
trace_err	Lists server error messages
trace_inf	This is a catch-all option that dumps any information not covered by the other options into the trace file
trace_dbg	Logs debug information
trace_wrn	Lists server warning messages
trace_sta	Provides server and engine state information, such as initialize, ready, run, and shut-down
trace_all	Logs all possible server and engine information in the trace file

When you specify multiple trace elements, separate them with vertical bars. For example:

```
traceOpts="trace_prf|trace_dbg|trace_wrn"
```

3.2.1.12 connection

EXAMPLE

```
<connection maxConnect="50" idleTimeOut="20">
    <orbClient id="RWClient" publicKeyFile="clientpub.key"/>
```

```
    <cluster publicKeyFile="ORACLE_HOME\reports\server\serverpub.key"
    privateKeyFile="ORACLE_HOME\reports\server\serverpri.key"/>
</connection>
```

REQUIRED/OPTIONAL

Optional. If you do not specify a *connection* element in your server configuration file, default values will be used (see Table 3–13). You can have a maximum of one *connection* element in your server configuration file.

DESCRIPTION

The *connection* element defines the rules of engagement between the server and the clients connected to it.

Connection attributes are listed and described in Table 3–13.

Table 3–13 Attributes of the connection element

Attribute	Valid values	Description
maxConnect	Number	**Default:** 20 The maximum number of requests that the server can service simultaneously. Requests in excess of the *maxConnect* value return a Java exception.
idleTimeOut	Number	**Default:** 15 Allowable amount of time in minutes the connection can be idle.

In addition to its attributes, *connection* has two sub-elements: *orbClient* and *cluster*.

Use *orbClient* to provide the name of the public key file that the client will use to connect to the Reports Server. You can have from zero to multiple *orbClient* sub-elements in your server configuration file.

The *orbClient* element attributes are listed and described in Table 3–14.

Table 3–14 Attributes of the orbClient element

Attribute	Valid values	Description
id	string	**Default:** RWClient Identifies the reports client to be served by the public and private key. You can also specify custom-built clients through the pluggable clients API. You'll find more information about Oracle9*i*AS Reports Services APIs on the Oracle Technology Network, http://otn.oracle.com.
publicKeyFile	<*filename*>.key	**Default:** clientpub.key Identifies the public key file that the client will use to connect to the Reports Server. The default file is stored in the rwrun.jar file.

We provide default client public and private key files, clientpub.key and clientpri.key. These key files are in place for all components of Oracle9*i*AS Reports Services. You can regenerate public and private key files to replace the default key pair. To do this, at the command prompt use the following command:

```
java oracle.report.utility.KeyManager <path_and_client_public_key_file_name>
<path_and_client_private_key_file_name>
```

If you regenerate these keys, you can specify the public key file locations with the *publicKeyFile* attribute, and replace the private key file in the `rwrun.jar` file (*`ORACLE_HOME`*`\reports\jlib\rwrun.jar`). To do this, you must unjar the file, place the regenerated private key into it, and rejar the file.

Use the *cluster* sub-element to specify the public and private key files to be used for all cluster members. You can have zero or one *cluster* element in your server configuration file.

Note: For more information on server clusters, see Chapter 6, "Reports Server Clusters".

For servers to be members of the same cluster, they must share the same extended cluster name and public and private keys. This means that their extended cluster names (e.g., serverA.*clus*, serverB.*clus*—in this case, *.clus* is the extended cluster name) should be the same, and the same public and private key files should be specified in each cluster member's server configuration file (<*server_name*>.conf).

The default server public and private keys are stored in the `rwrun.jar` file, in the path *`ORACLE_HOME`*`\reports\jlib\rwrun.jar` on both Windows and UNIX. However, there is no need to jar the newly-generated public and private keys that will be used for the cluster. Put them anywhere, and specify the absolute path and filename for them in the server configuration file.

3.2.1.13 queue

EXAMPLE

```
<queue maxQueueSize="1000"/>
```

REQUIRED/OPTIONAL

Optional. You can have a maximum of one *queue* element in your server configuration file. If you have no *queue* element, the default, 1000, will remain in effect.

DESCRIPTION

Use the *queue* element to specify the maximum number of jobs that can be held in each of the Reports queues. Reports Services has three queue components:

- a queue of scheduled jobs
- a queue of jobs in progress
- a queue of completed jobs

The *queue* element provides the allowable value for each of these components. If the number of jobs exceeds the specified maximum of a given queue, that queue will automatically purge its expired, then its oldest jobs. You can also use the Reports Queue Manager to manually reduce the number of jobs held in the queue.

> **Note:** For more information, see the Reports Queue Manager online help.

The *queue* element has one attribute: *maxQueueSize*, described in Table 3–15.

Table 3–15 Attributes of the queue element

Attribute	Valid values	Description
maxQueueSize	Number	**Default:** 1000 The maximum number of jobs that can be held in a given Reports job queue.

3.2.1.14 persistFile

EXAMPLE

```
<persistFile filename="neptune.dat"/>
```

REQUIRED/OPTIONAL

Optional. If you do not specify a file, the server will create one of its own with the default name <*server_name*>.dat. You can have a maximum of one *persistFile* element.

DESCRIPTION

The *persistFile* element identifies the file that records all job status. It is used by the Reports Server to restore the server to the status it held before shutdown.

It is named *persistFile* because the file remains intact, or persists, even when the server is brought down and restarted.

The server persistent file is created automatically the first time you start the server or the first time you start the server after the current server persistent file has been deleted or renamed. If you want to rename this file but continue using it, enter the new name in the server configuration file before you actually rename the file, then restart the server.

The *persistFile* element has one attribute, *fileName,* described in Table 3–16.

Table 3–16 Attributes of the persistFile element

Attribute	Valid values	Description
fileName	string	**Default:** <*server_name*>.dat The name and, optionally, the path of the server persistent file. You can leave the path off if the file is kept in its default directory: *ORACLE_HOME*\reports\server\ The path is the same for Windows or UNIX.

3.2.1.15 identifier

EXAMPLE

```
<identifier confidential="yes"
encrypted="yes">fpoiVNFvnlkjRPortn+sneU88=NnN</identifier>
```

REQUIRED/OPTIONAL

Optional. You can have a maximum of one *identifier* element in your server configuration file.

DESCRIPTION

The *identifier* element is automatically written to the configuration file when you first log in to the Reports Queue Manager as an administrator. The first login sets the Queue Manager's administrator user ID and password. That information is encrypted and written to the server configuration file, then used for authentication for all future Queue Manager logins.

If you forget the Queue Manager login, delete it from the server configuration file, and reenter the information in the Reports Server configuration file in the following format:

```
<identifier confidential="yes" encrypted="no">username/password</identifier>
```

The next time you run the Queue Manager, the Reports Server will automatically encrypt the identifier and reset `encrypted` to `yes`. It will look something like this:

```
<identifier confidential="yes"
encrypted="yes">fpoiVNFvnlkjRPortn+sneU88=NnN</identifier>
```

For more information on the Reports Queue Manager, see the Reports Queue Manager online help.

3.2.1.16 pluginParam

EXAMPLE

```
<pluginParam name="mailServer">smtp01.mycorp.com</pluginParam>
```

REQUIRED/OPTIONAL

Optional. You can have as many *pluginParam* elements as you require.

DESCRIPTION

The *pluginParam* element works in cooperation with all pluggable components of Oracle9*i*AS Reports Services. This includes the engine, security, cache, destination, and jobstatusRepository components. Any one of these may need access to a mail server, an FTP URL, or some other type of plugin. The *pluginParam* element provides a means of specifying plugins that can be used by all pluggable components. This spares you the task of including this information in the class definition of the pluggable component and allows you to rapidly and easily change the source of the plugin.

For example, your custom pluggable engine and destination Java classes may both need access to a proxy server. Instead of hard-coding access to the server in both of these classes, you can merely call the type of plugin you need, for example *proxy*, and point to its location under *pluginParam* in the server configuration file.

You can put in any plugin and name it in any way as long as it is a plugin supported or required by the pluggable component, and the pluggable component knows its name.

The *pluginParam* attributes are listed and described in Table 3–17.

Table 3–17 Attributes of the pluginParam element

Attribute	Valid values	Description
name	string	The name of the plug-in parameter.

Table 3–17 Attributes of the pluginParam element

Attribute	Valid values	Description
type	text	**Default:** text
	file	Describes the type of plugin being specified.
	url	▪ For *text*, put in the string that is required to identify the named plugin, for example, the name of a mail server. *Text* means the content of the *pluginParam* element is text, so the getPluginParam() method will return the exact content specified in the element. ▪ For *file*, put in the directory path and filename of the plugin file. Use the standards for specifying directory paths appropriate to the Reports Server's host machine (either Windows or UNIX). *File* means that the content of the *pluginParam* element is a filename, and the getPluginParam() method will return the content read from the file specified. ▪ For *url*, put in the full, absolute URL required by the plugin, for example, the full URL to an FTP site. URL means the content of the pluginParam element is a URL, and the getPluginParam(() method will return the content read from that URL. The URL you use must reside on the same side of the firewall as Oracle9*i*AS Reports Services.
		Note that when you have a default type (text), it is not necessary to specify it in the *pluginParam* string. The example that heads this section doesn't specify a type because the plugin, a mail server name, is the default type, text.

3.3 Configuring the Reports Servlet

Configure the Reports Servlet with a file named rwservlet.properties, located in the following path (Windows and UNIX use the same path):

```
ORACLE_HOME\reports\conf\rwservlet.properties
```

You may notice that the servlet uses components you may have become familiar with if you used to employ a CGI implementation.

Use the Reports Servlet configuration file for:

- Specifying the location of the key map file
- Reloading the Key Map File
- Hiding Verbose Error Messages
- Selecting Login Dialog Boxes
- Setting up Trace Options for the Reports Servlet and JSPs
- Customizing the Appearance of Server Error Messages
- Specifying an In-Process Server
- Identifying the Default Reports Server
- Pointing to Dynamically Generated Images
- Setting Expiration for DB Authentication and SYSAUTH Cookies
- Setting an Encryption Key for the DB Authentication Cookie
- Adding Formatting to Diagnostic/Debugging Output
- Specifying an SSL Port Number
- Defining the rwservlet Help File
- Specifying the Use of Single Sign-On

The entries in this configuration file are not case sensitive.

For Windows, note that the servlet configuration file uses double backslashes (\ \) in lieu of single backslashes to specify a directory path. The first slash "escapes" the second, which would otherwise have another meaning in this file. For example, in a Windows-based Reports Servlet file, the path:

```
d:\orawin\reports\conf\filename.ext
```

Becomes:

```
d:\\orawin\\reports\\conf\\filename.ext
```

For UNIX, use that platform's standard for specifying directory paths, for example:

```
orawin/reports/conf/filename.ext
```

3.3.1 Specifying the location of the key map file

Your report runtime command line may include information you do not want to expose to your users. Additionally, it may be comprised of a long string of options that is difficult to remember or makes for an ungainly URL.

You have the option of entering a report's command line arguments in a key map file (cgicmd.dat), then limiting the exposed runtime command to the name of the particular key section in this file that holds the required arguments.

The key map file is discussed in Chapter 8, "Running Report Requests". Use the Reports Servlet configuration file to list the location of this file.

For example:

```
KeyMapFile=d:\\orawin\\reports\\conf\\cgicmd.dat
```

This example uses the default filename and location. An entry for the location and filename of the key map file doesn't appear by default in the servlet configuration file because the servlet already knows what to look for and where to look for it. If you use a file with a different name and/or different location, you must include a `KeyMapFile` parameter in your servlet configuration file that includes the directory path and filename.

3.3.2 Reloading the Key Map File

Use the RELOAD_KEYMAP parameter to specify whether the key map file (cgicmd.dat) should be reloaded each time the servlet receives a request.

For example:

```
RELOAD_KEYMAP=yes
```

This is useful if you frequently make changes to the map file and want the process of loading your changes to be automatic. Runtime performance will be affected according to how long it takes to reload the file.

Typically, this parameter is set to *no* in a production environment and *yes* in a testing environment.

3.3.3 Hiding Verbose Error Messages

Should a runtime error occur, the body of the resulting error message can include the details of the runtime command line. If your runtime command line contains sensitive information, such as a userid and password, you may not wish to have it display along with the text of the error message.

Use the DIAGNOSTIC parameter in the servlet configuration file to specify whether command line information should be included along with the rest of the error message.

Enter NO for the DIAGNOSTIC parameter if you don't want command line details to display. NO is the default value. For example:

```
DIAGNOSTIC=NO
```

Enter YES if you prefer that all error messages include the detailed runtime command line.

3.3.4 Selecting Login Dialog Boxes

The servlet configuration file offers a number of parameters dealing with templates for userid/password dialog boxes that should open when a user logs in to a database or runs a secure report. Generally, these parameters point to various templates to be used for setting up your login screens. You can customize these templates with your company logo, linked buttons, and/or any other HTML you care to use.

The DBAUTH and SYSAUTH parameters are for specifying the location and filename of the HTML templates to be used for individual login screens.

For example, the following entry points to the template for the database login screen:

```
DBAUTH=dbauth.htm
```

SYSAUTH points to a login screen for a secure report. For example:

```
SYSAUTH=SYSAUTH.HTM
```

It isn't necessary to enter the path to a template when it is stored in the default template directory:

```
ORACLE_HOME\reports\templates
```

The DB_SYS_DIFFAUTH and DB_SYS_SAMEAUTH templates are for combining these login screens into one screen. Use DB_SYS_DIFFAUTH when your users must enter different login information for database and report access.

For example:

```
DB_SYS_DIFFAUTH=dbsysdif.htm
```

Use DB_SYS_SAMEAUTH template when your users enter the same login information for both types of access.

For example:

```
DB_SYS_SAMEAUTH=dbsyssam.htm
```

In this case, the user is prompted to enter the login information only once, and the servlet will take care of passing the values to both the database and the Reports Server.

3.3.5 Setting up Trace Options for the Reports Servlet and JSPs

Trace has been added to the Reports Services environment to allow the logging of many different types of runtime information on various Reports Services components.

Note: Tracing for the Reports Server is configured in the server configuration file, <*server_name*>.conf, discussed in Section 3.2.1.11. Tracing for an individual report can be built into the reports runtime command line, discussed in Appendix A, "Command Line Arguments".

If you wish to track and log runtime information on the Reports Servlet and JSPs, use the TRACEOPTS parameter in the servlet configuration file. You can enter from zero to multiple trace options. Separate each option with a vertical bar.

For example:

```
TRACEOPTS=trace_prf|trace_pls|trace_dbg
```

All available trace options are listed and described in Table 3–12.

Additionally, you can use the TRACEFILE and TRACEMODE parameters.

Use TRACEFILE to specify the filename of the trace file. For example:

```
TRACEFILE=<server_name>.trc
```

If no path is specified, the trace file will be in the following directory on both Windows and UNIX:

```
ORACLE_HOME\reports\logs
```

Use TRACEMODE to define whether new trace information will either overwrite the existing trace file (trace_replace), or be added to the end of the trace, leaving existing trace information intact (trace_append). For example:

```
TRACEMODE=trace_append
```

The default for TRACEMODE is trace_replace.

3.3.6 Customizing the Appearance of Server Error Messages

Reports Services provides a template for server error messages. These messages are generated automatically, according to cause. The template provides the visual setting within which the error message is displayed.

You may wish to customize the appearance of error messages, for example with your company logo, or with an icon you plan to associate with errors. You may wish to add buttons that link your users to a help system, your company home page, or back to the last browser window. You can do this by providing your own HTML framework for automatically generated error messages.

The entry in the servlet configuration file is for pointing to the name and location of your error message template.

By default, the entry is:

```
ERRORTEMPLATE=rwerror.htm
```

It isn't necessary to enter the path to the error message template when it is stored in the default template directory:

ORACLE_HOME\reports\templates

3.3.7 Specifying an In-Process Server

If you choose to run the Reports Server within the same process as the Reports Servlet, you indicate that with the SERVER_IN_PROCESS parameter. To run the Reports Server as an in-process server, specify the following in the servlet configuration file:

```
SERVER_IN_PROCESS=yes
```

Enter *no* if you do not want the Reports Server to run within the same process as the Reports Servlet.

Note: The pros and cons of running an in-process server are explored in Chapter 1, "Oracle9iAS Reports Services Architecture".

3.3.8 Identifying the Default Reports Server

The Reports Servlet uses the SERVER parameter to identify the default Reports Server. If a server name is not specified, for example, in the runtime URL, the default server specified here is used. Enter the name of your default Reports Server in the servlet configuration file.

For example:

```
SERVER=<server_name>
```

If the default Reports Server is a member of a server cluster, use the cluster name:

```
SERVER=<cluster_name>
```

If you use a combination of the server name and cluster name, requests sent to the default server will go to this specific machine. If this machine is down, an error message will be returned and the report will not be run. By specifying just the cluster name, requests will be sent to a random cluster member, and fowarded to another if the target machine doesn't have an idle engine.

If you don't specify a SERVER parameter in `rwservlet.properties`, the default server name is `rep_<machine_name>`.

3.3.9 Pointing to Dynamically Generated Images

Use the IMAGE_URL parameter to specify where the Reports dynamically generated images can be accessed.

For example:

```
IMAGE_URL=http://<server_or_web_server_name>.<domain_
name>:<port>/reports/rwservlet
```

This parameter is in place for JSPs that do not run via the Reports Servlet. It ensures that dynamically generated images, such as charts, will be viewable only by the person who runs the report. JSPs, and other report types, that run through the Reports Servlet have this protection automatically.

3.3.10 Setting Expiration for DB Authentication and SYSAUTH Cookies

Use the COOKIEEXPIRE parameter to set the lifetime (in minutes) of the database and system authentication cookie. For example:

```
COOKIEEXPIRE=20
```

The default is 30.

Cookies save encrypted user names and passwords on the client-side when users first authenticate themselves. When the server receives a cookie from the client, the server compares the time saved in the cookie with the current system time. If the time is longer than the number of minutes defined in COOKIEEXPIRE, the server rejects the cookie and returns to the client the database authentication form along with an error message. Users must re-authenticate to run the report.

3.3.11 Setting an Encryption Key for the DB Authentication Cookie

Use ENCRYPTIONKEY to specify the encryption key to be used to encrypt the user name and password of the DB authentication cookie. The encryption key can be any character string. For example:

```
ENCRYPTIONKEY=egbdf
```

3.3.12 Adding Formatting to Diagnostic/Debugging Output

The DIAGBODYTAGS and DIAGHEADTAGS parameters are available for including additional HTML encoding in the <body> and <head> tags in the output files associated with diagnostic and debugging output.

DIAGBODYTAGS defines the entire <body> tag; while DIAGHEADTAGS defines tags to appear between the open and close <head>/</head> tags.

You can use these to include formatting arguments to make diagnostic and debugging output easier to read. For example:

```
DIAGBODYTAGS=<BODY [additional HTML encoding]>

DIAGHEADTAGS=<HEAD>[additional HTML encoding]</HEAD>
```

3.3.13 Specifying an SSL Port Number

Use SSLPORT if you're using Secure Sockets Layer (SSL) and you want to use a port other than the default (443). For example:

```
SSLPORT=<port number>
```

3.3.14 Defining the rwservlet Help File

A HELP keyword is available with the rwservlet command for bringing up a servlet-related help topic. The help file is invoked when you specify the following URL:

`http://yourwebserver/yourservletpath/rwservlet/help`

> **Note:** For more about the HELP keyword, see Appendix A, "Command Line Arguments".

We provide a default help file for the servlet, which will be displayed if you leave this parameter undefined. (The default information is pulled from code and is not a file as such.) You may want to supply a help file of your own. To do this, specify the name and location URL of your servlet help file with the HELPURL parameter in the servlet configuration file. For example:

`HELPURL=http://your_web_server/your_help_file_path/helpfile.htm`

3.3.15 Specifying the Use of Single Sign-On

If you plan to take advantage of Oracle9*i*AS Reports Services single sign-on capability, you must specify this in the servlet configuration file with the SINGLESIGNON parameter. Enter YES to indicate that you will use single sign-on to authenticate users; enter NO if you will not use single sign-on. If you choose NO, the Reports Server will use its own authentication mechanism to authenticate users (i.e., as was used in Reports6*i*).

To specify that you will use single sign-on to authenticate users, enter:

```
SINGLESIGNON=yes
```

3.4 Configuring the URL Engine

The Reports Server includes a URL engine that can take the contents of any URL and distribute them. The URL engine allows you to leverage the powerful scheduling and distribution capabilities of the Reports Server to distribute content from any publicly available URL to various destinations such as e-mail, Oracle9*i*AS Portal, and WebDav. Since the Reports Server's destinations are pluggable, you can also add your own custom destinations for the URL content.

Furthermore, if you use the URL engine in conjunction with the Reports Server's event-based APIs, database events can trigger the content distribution. For example,

suppose you have created a JSP report for high fidelity Web publishing of data stored in a table containing employee expense data. You could then use the URL engine and the event-based API to e-mail that JSP whenever the expense application stores new or updated employee expense data in the table.

By default, the URL engine is not activated when you install Oracle9*i*AS Reports Services. You can activate the URL engine by doing the following:

1. Add an engine element for the URL engine to the server configuration file. For example, your engine element might be as follows:

```
<engine id="rwURLEng"
             class="oracle.reports.engine.URLEngineImpl"
             initEngine="1"
             maxEngine="1"
             minEngine="0"
             engLife="50"
             maxIdle="30"
             callbackTimeOut="60000"
/>
```

2. Add a job element that associates the appropriate job types with the URL engine to the server configuration file. For example, your job element might be as follows:

```
<job  jobType="urlEngine"
         engineId="rwURLEng"
/>
```

3. Stop and restart the Reports Server.

Note: When you restart your Reports Server with these new elements, you should see the number of engines increase accordingly in the Reports Server status message box. In the above example, the number of engines would increase by one (the value of initEngine) when you restart the Reports Server.

To learn about sending requests to the URL engine, refer to Chapter 8, "Running Report Requests".

3.5 Entering Proxy Information

Some features of Oracle9*i*AS Reports Services support retrieving or sending information through a firewall. For example, the URL engine, the XML data source, the Text data source, and the mail destination features all retrieve or send information through the firewall. For these features to function properly, the Reports Server requires certain proxy information. In the interests of simplicity, you store the necessary proxy information in a single location and point to it from the Reports Server configuration file. To configure your Reports Server with proxy information, you do the following:

1. Add the pluginParam element to the server configuration file and have it point to the proxy information file (e.g., proxyinfo.xml). For example, your pluginParam element might be as follows:

   ```
   <pluginParam name="proxy" type="file">proxyinfo.xml</pluginParam>
   ```

   Note: You can optionally specify a path for the proxy information file. By default, this file is located in this file is located in <ORACLE_HOME>/reports/conf.

2. Update the proxy information file with the necessary proxy values for your configuration. For example, proxyinfo.xml might contain the following:

   ```
   <proxyInfo>
      <proxyServers>
        <proxyServer name="xyz.abc.com" port="80" protocol="http"/>
        <proxyServer name="www-proxy1.xyz.abc.com" port="80" protocol="ftp"/>
        <proxyServer name="www-prox21.xyz.abc.com" port="80" protocol="https"/>
      </proxyServers>
      <bypassProxy>
        <domain>*.abc.com</domain>
      </bypassProxy>
   </proxyInfo>
   ```

   Note: Refer to the default proxy information file, <ORACLE_HOME>/reports/conf/proxyinfo.xml, for additional information.

3.6 Configuring the Reports Server for Oracle Enterprise Manager

Oracle Enterprise Manager (OEM), included with Oracle9*i*AS, provides managing and monitoring services to Oracle9*i*AS Reports Services. To take advantage of these services, OEM must be configured to work with the Reports Server. The default Reports Server instance that is configured and started as part of the Oracle9*i*AS installation is automatically configured with OEM. If you add another Reports Server instance, you must configure it with OEM by running the following command line:

On Unix:

```
<ORACLE_HOME>/bin/addNewServerTarget.sh <reports_server_name>
```

On Windows:

```
<ORACLE_HOME>\bin\addNewServerTarget.bat <reports_server_name>
```

Note: By default, the above script takes the user name and password for the Oracle9iAS Portal installation as the user name and password for the OEM-Reports Server integration. If Oracle9iAS Portal is not installed, the default user name is repadmin and the password is reports.

For more information about using OEM for the Reports Server, refer to Chapter 13, "Managing and Monitoring Oracle9iAS Reports Services".

4

Configuring Destinations for Oracle9*i*AS Reports Services

Two things to consider when you run a report are how the report should be output (destination) and who should receive it (distribution). Distribution is discussed in Chapter 9, "Creating Advanced Distributions". This chapter explores how Oracle9*i*AS Reports Services handles output processing to default and custom destinations. It provides an overview of output processing and information on registering destination types with the Oracle9*i*AS Reports Server.

It includes the following sections:

- Overview of Output Processing
- Registering Destination Types with the Server

4.1 Overview of Output Processing

How the report should be output is controlled by the *destypes* you specify at runtime, which, in turn, are determined by the destination output types you have registered in your server configuration file (<*server_name*>.conf). You can register no output types and simply use the default types provided by Oracle9*i*AS Reports Services:

- Cache (i.e., browser)
- SMTP-compliant e-mail
- File
- Printer
- Oracle9*i*AS Portal (this is an exception in that, for access to the portal, it requires the specification of a userid and password in the server configuration file)

You can also define a custom output types, such as fax, FTP, Oracle's Internet File System (*i*FS), or any type you care to define through the Oracle9*i*AS Reports Services Destinations API. This API enables you to define new destination types and build handlers to usher your reports to custom destinations.

Note: Build a custom destination type via the Oracle9*i*AS Reports Services Destinations API. Look for upcoming information about Reports APIs and destination types for download on the Oracle Technology Network: http://otn.oracle.com.

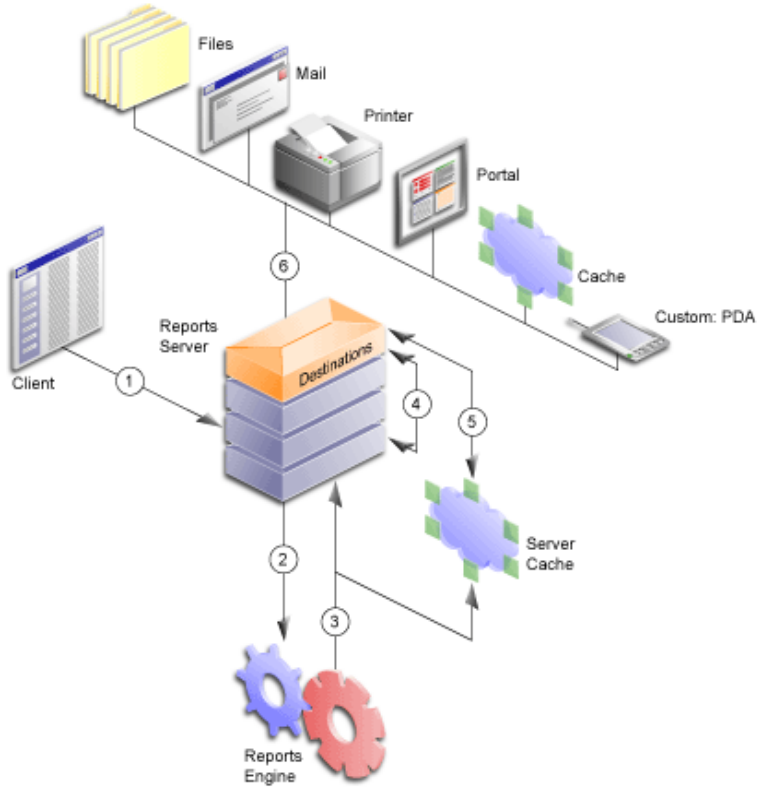
Oracle9*i*AS Reports Services architecture standardizes the way output is generated and delivered. It takes responsibility for delivering report output to the appropriate destination (via the Reports Server), yet generates output independent of its destination (via the Reports Engine). This provides a significant improvement in efficiency by allowing one run of a report to be used in a number of different ways. It also opens up the output processing architecture to allow for any number of destination types.

In the past, the Reports runtime engine was totally responsible for delivering the output. Consequently, it had to know how to communicate with output destinations. This resulted in a tight coupling between the engine and the supported destinations.

Oracle9*i*AS Reports Services eliminates this tight coupling and its attendant restrictions. The runtime engine now treats all destinations alike. It doesn't need to know the destination type for which the output is being produced. The server hands output off to destination handlers that prepare the material for delivery to their associated destination types. You can use predefined destination types (with predefined handlers) or create a handler for a custom destination type you intend to support. Almost any type of destination can be plugged into Reports.

Figure 4–1 illustrates the main components of the Oracle9*i*AS Reports Services output processing architecture.

Figure 4–1 Main components of destination/distribution architecture



Requests flow through the output processing architecture in the following sequence:

1. The user submits a request from a client or browser to the Reports Server.
2. The server passes it along to the runtime engine.
3. The runtime engine creates / processes the destination objects (which include file lists for specific destinations as well as any properties related to those destinations) and the report output; the runtime engine sends the destination objects to the Reports Server and the report output to cache.
4. The Reports Server sends the destination objects to the Reports Server's destination component.
5. The destination component of the Reports Server fetches the report output from cache.

6. The Reports Server destination component sends the report and the destination objects (which specify how the destination device should handle the output) to the appropriate destination handler.

4.2 Registering Destination Types with the Server

Before the Oracle9*i*AS Reports Server can send a report to a particular destination type, the type must be a default type (printer, e-mail, cache, or file) or a type registered in the server's configuration file, <*server_name*>.conf. The configuration file contains a *destination* element for registering destination types that are valid for your reports. You can register anywhere from zero to any number of destination types.

Registering a destination type with the server involves:

- Setting Up a Destination Section in the Server Configuration File
- Entering Valid Values for a Destination

These tasks are described in the following sections.

4.2.1 Setting Up a Destination Section in the Server Configuration File

To set up a destination section in the <*server_name*>.conf file:

1. Open the server configuration file with your preferred text editor.

   You'll find the server configuration file in the following directory (Windows and UNIX use the same path):

   ORACLE_HOME\reports\conf\<*server_name*>.conf

2. If the configuration file does not have a *destination* section, create one underneath the element that precedes it in the configuration file's data type definition file (rwserverconf.dtd) section.

   Note: The server configuration file follows the order of elements defined in the file's related document type definition file (*ORACLE_HOME*\reports\dtd\rwserverconf.dtd). Place *destination* after the elements that precede it, whichever are present in your server configuration file.

3. Use the following syntax to register all the destination types you will use for outputting reports:

```
<destination destype="output_type_1" class="java_class_1">
    <property name="valid_destype_property" value="valid_value"/>
    <property name="valid_destype_property" value="valid_value"/>
</destination>
<destination destype="output_type_2" class="java_class_2">
    <property name="valid_destype_property" value="valid_value"/>
</destination>
```

The valid values for these tags are discussed in the following sections.

4.2.2 Entering Valid Values for a Destination

4.2.2.1 Destination destypes and classes

The *destype* and *class* attributes are required for valid registration of a non-default output type. They specify the destination types and their associated Java classes. The predefined (default) destination types and classes that come with Oracle9*i*AS Reports Services are listed in Table 4–1:

Table 4–1 Standard destination types and classes

Destination	destype	class
Oracle9*i*AS Portal content area	oraclePortal	oracle.reports.server.DesOraclePortal
SMTP-compliant e-mail	mail	oracle.reports.server.DesMail
file	file	oracle.reports.server.DesFile
cache	cache	oracle.reports.serverDesCache
printer	printer	oracle.reports.server.DesPrint

Contrary to the other default types, you must register an oraclePortal destype. This is because the oraclePortal destype requires the specification of a userid and password for accessing the portal.

You are not limited to the predefined destypes and classes provided with the server. You can register custom destination types, such as a fax or FTP, once you have defined a custom handler (through the Destinations API).

Note: Look for upcoming information about Reports APIs on the Oracle Technology Network: http://otn.oracle.com.

4.2.2.2 Destination Property name/value Pairs

The server configuration file allows the association of an unlimited number of properties with a registered destination. Destination properties consist of name/value pairs that define some aspect of an output type's configuration. They are expressed in terminology recognized by the destination type. For example, a destination with a destype of oraclePortal would recognize the name/value pair:

```
<property name="portalUserid" value="portal_id/portal_password@portal_schema"
confidential="yes" encrypted="no"/>
```

This example defines the values to be associated with a portal user ID. It includes the attributes *confidential* and *encrypted*: `confidential="yes"` indicates that the values within this element should be encrypted; `encrypted="no"` indicates that the values are not yet encrypted. The next time the Reports Server starts, it will automatically encrypt the values and reset `encrypted` to `yes`.

Note: Elements and attributes allowable in server configuration file are determined by the syntax defined in the rwserverconf.dtd file (*ORACLE_HOME*`\reports\dtd\rwserverconf.dtd`). This is discussed in detail in Chapter 3, "Configuring Oracle9iAS Reports Services".

What is valid for a destination type's properties depends entirely on the destination type. These values do not come from Reports and are not put to use by the Reports Server. They come from the destination type itself and use terms the destination recognizes. It is up to the developer to understand the requirements of a custom destination and to know what properties to associate with a given custom output type.

When we begin to discuss distribution, you may note that within the distribution XML file, the *destype* element also allows for the use of property name/value pairs. It's important to make a distinction between properties entered for a *destination* element in the server configuration file and those entered for a *destype* element in the distribution XML file:

- Properties entered for a *destination* element in the server configuration file should deal only with configuring an output type, for example setting an allowable number of retries for a destination fax.
- Properties entered for a *destype* element in the distribution XML file should deal only with specifying a runtime parameter, for example the identity of the fax's intended recipient.

5

Controlling User Access

The celebrated openness of the Internet brings with it concerns about controlling who has access to what confidential company information. Oracle9*i*AS Portal provides a number of security features available to Oracle9*i*AS Reports Services that enable you to ensure that the appropriate users are getting important data in a secure fashion. With Oracle9*i*AS Portal security features in place, your users see only the data they're supposed to see.

Use Oracle9*i*AS Reports Services to control:

- Who has access to each report
- When a report can be run
- Which servers and printers can be used to run a report
- Which report parameters a user can edit with what range of values

This chapter describes how to use Oracle9*i*AS Reports Services security and the out-of-the-box security implementation provided with Oracle9*i*AS Portal to control user access to your Reports environment. It includes the following sections:

- Introduction to Oracle9iAS Portal
- Defining Portal-Based Security in the Server Configuration File
- Creating Reports Users and Named Groups in Oracle9iAS Portal
- Setting Up Access Controls

Before you can set up security controls, both Oracle9*i*AS Portal and Oracle9*i*AS Reports Services must be installed and configured. See Chapter 3, "Configuring Oracle9iAS Reports Services" for information on configuring Reports Services. See the Oracle9*i*AS Portal documentation for information on configuring Oracle Portal. See also the Oracle9*i*AS Install Guide, for information on installing both components. You'll find information about Oracle9*i*AS and Oracle9*i*AS Portal on the

Oracle9*i*AS documentation CD and on the Oracle Technology Network, http://otn.oracle.com.

5.1 Introduction to Oracle9*i*AS Portal

Oracle9*i*AS Portal is a browser-based, data publishing and developing solution that offers Web-based tools for publishing information on the Web and building Web-based, data-driven applications.

Oracle9*i*AS Portal is tightly integrated with Oracle9*i*AS Reports Services to create a robust and secure data publishing environment. Oracle9*i*AS Portal provides easy-to-use wizards for setting up Oracle9*i*AS Reports Services security. These include wizards for defining user access to reports, Oracle9*i*AS Reports Servers, printers, output formats, and report parameters.

Once you define access control information, it's stored in the Oracle9*i*AS Portal repository. As an Oracle9*i*AS Portal user, you can then, optionally, publish registered RDFs and JSPs to an Oracle9*i*AS Portal page. As with all Oracle9*i*AS Portal functionality, using Portal to deliver your reports is not required. You can deliver reports through command lines, as you may always have, and still benefit from the access control features available to you through Oracle9*i*AS Portal.

Access to Oracle9*i*AS Reports Services' security features is not dependent on whether you also use Portal to publish report links or report content. Even if you don't publish via Portal, you can still take advantage of the Oracle9*i*AS Reports Services' security features available in Oracle9*i*AS Portal to control user access to all of your reports.

When you expose a report as a portlet through Oracle9*i*AS Portal, Oracle9*i*AS Reports Services leverages the Oracle Single Sign-on feature. Oracle Single Sign-on eliminates the need for users to enter multiple logins, first to the portal then to each of the applications exposed through portlets within the portal. With Single Sign-On, when you log in, Oracle9*i*AS Portal automatically logs you into all registered portlet providers and subsystems.

Refer to the Oracle9*i*AS Security Guide for more information about Single Sign-on. You'll find this and other related documentation on the Oracle Technology Network, http://otn.oracle.com.

5.2 Defining Portal-Based Security in the Server Configuration File

If you plan to use the security features, you must set up the *security* element in your Reports Server configuration file: <*server_name*>.conf. You'll find this file in the following directory path on both UNIX and Windows platforms:

ORACLE_HOME\reports\conf\<*server_name*>.conf

For the out-of-the-box Portal security implementation, the Reports Server configuration file's *security* element requires a property that includes a valid Portal username, password, and database connect string SID. The Reports Server uses this information to connect the server to Portal and retrieve the access control parameters you set there. The server's connection to Portal is performed in the background and does not present any displayed components. A user can run a report that has access controls without being aware that those controls were specified in and served up via Oracle9*i*AS Portal.

In the Reports Server configuration file, your security configuration entry might look like this:

```
<security id=″rwSec″ class="oracle.reports.server.RWSecurity″>
    <property name="securityUserid" value="portal_id/portal_password@portal_
    schema" confidential="yes" encrypted="no"/>
</security>
```

Note: If you implement your own security interface, you need to implement in Java. Refer to the Oracle Technology Network (http://otn.oracle.com/products/reports/) for more information on the Oracle9*i*AS Reports Services APIs .

Valid attributes of the *security* element are described in Table 5–1.

Table 5–1 Attributes of the security element

Attribute	Valid values	Precondition or default	Description
id	string	required	A keyword, unique within a given configuration XML file, that identifies a particular *security* element. This can be a text string or a number, for example `id="rwSec"`.

Table 5–1 Attributes of the security element

Attribute	Valid values	Precondition or default	Description
class	*see Description*	required	A fully qualified Java class that implements the Reports Server Security Java interface (oracle.reports.server.Security). The default value is oracle.reports.server.RWSecurity, which relies on security features available through Oracle Portal (included with Oracle9*i*AS).

Additionally, the security element in this example uses the name/value pair: `securityUserid`/*`portal_id/portal_password@portal_schema.`* `securityUserid` is the name of the property, and *`portal_id/portal_password@portal_schema`* describes a valid, administrator-level user id, password, and SID for entry into Portal. This example also includes the attributes *confidential* and *encrypted*: `confidential="yes"` indicates that the values within this element should be encrypted; `encrypted="no"` indicates that the values are not yet encrypted. The next time the Reports Server starts, it will automatically encrypt the values and reset `encrypted` to `yes`.

The *security* configuration element is explained in detail in Section 3.2.1.5, "security" in Chapter 3.

5.3 Creating Reports Users and Named Groups in Oracle9*i*AS Portal

If you use the security features in Oracle Portal to control access to your reports, you must register all of your Reports users in the Oracle Internet Directory (OID) and assign security privileges to all of them through Oracle9*i*AS Portal.

Note: If you have a large user population already entered into an LDAP-compatible directory, you can use Oracle Internet Directory (OID) features to synchronize the directories and save yourself the effort of entering your users individually. You'll find information about OID's Directory Integration Server in the *OID Administrator's Guide*.

In Portal, security privileges can be granted to individual users and to named groups of users. Named groups are useful for streamlining the process of granting access privileges. You can assign a set of access privileges to a named group, and grant the entire set of privileges to an individual simply by adding that person to the group.

The next sections provide overview information on how to create users and groups in Oracle9*i*AS Portal. They include:

- Default Reports-Related Groups
- Creating Users and Groups

5.3.1 Default Reports-Related Groups

When you install Oracle9*i*AS Portal, Reports-related groups are created for you automatically. These include the following groups:

- RW_BASIC_USER
- RW_POWER_USER
- RW_DEVELOPER
- RW_ADMINISTRATOR

Every person who will access your reports should belong to one of these groups. Each of these groups comes with a set of access privileges, which you may customize if you wish. If users try to run reports without being a member of one of these groups, by default, they are assigned the privileges of a basic user. The groups and their privileges are described in the following subsections.

5.3.1.1 RW_BASIC_USER

Basic users have EXECUTE privileges. They can run a report and see the result. Should the security check fail, they see less detailed error messages than the other Reports user groups see, such as:

```
Security Check Error
```

5.3.1.2 RW_POWER_USER

In addition to the privileges of the RW_BASIC_USER group, the RW_POWER_ USER group sees error messages that are more detailed than those displayed to basic users. For example, if they are not permitted to run to HTML, but they try anyway, they might get the message:

```
Cannot run report to HTML
```

This is more detailed than the message an RW_BASIC_USER would receive for the same error.

5.3.1.3 RW_DEVELOPER

In addition to the privileges of the RW_POWER_USER and RW_BASIC_USER groups, the RW_DEVELOPER group can run commands, such as SHOWENV and SHOWMAP, which show the system environment. You would assign a developer who needs to do testing and to retrieve detailed error messages to this group.

5.3.1.4 RW_ADMINISTRATOR

Users assigned to this group have MANAGE privileges. They can CREATE, UPDATE, and DELETE the registered report definition files, servers, and printer objects in Oracle9*i*AS Portal. In addition to all the links activated for the developer user, administrators can navigate to the Access tab on the Component Management Page, accessible in Oracle9*i*AS Portal. This is where the administrator can specify who will have access to this report. People with administrator privileges can assign security privileges for other people and receive full error messages from Oracle9*i*AS Reports Services.

These users also have access to the administrator's functionality in Oracle9*i*AS Reports Queue Manager, which means they can manage the server queue, including rescheduling, deleting, reordering jobs in the server, and shutting down a server.

Refer to the *Oracle9iAS Security Guide* for information on creating and managing a user.

5.3.2 Creating Users and Groups

Oracle9*i*AS Portal uses the Delegated Administration Service (DAS) interface to the Oracle Internet Directory (OID) to register users for access to Portal. You can enter the DAS interface through Portal to create new users. The creation of new users and groups is discussed in the Oracle9*i*AS Security Guide, available on the Oracle9*i*AS documentation CD. Look for the chapter entitled, "Configuring Oracle9*i*AS Portal Security."

5.4 Setting Up Access Controls

Before you begin, you must have a sufficient level of privileges in Oracle9*i*AS Portal in order to access the portlets and complete the tasks required for setting access

controls. You will find information about joining privileged groups in the Oracle9*i*AS Security Guide, available on the Oracle9*i*AS documentation CD. Look for the chapter entitled, "Configuring Oracle9*i*AS Portal Security."

Once you have a sufficient level of privileges, you can use the information in the following sections to learn about:

- Creating an Availability Calendar
- Registering a Printer
- Registering a Reports Server
- Registering a Report

5.4.1 Creating an Availability Calendar

Defining availability calendars is an optional step that allows you to further restrict access to reports, servers, and printers by specifying when they can and cannot be accessed. Availability calendars are not necessary if the reports, the Reports Servers, and printers are always available for processing.

This section provides information on:

- Creating a Simple Availability Calendar
- Creating a Combined Availability Calendar

You can associate only one availability calendar with a report, a Reports Server, or a printer. If your production environment requires more than one availability rule, then you can combine availability calendars.

5.4.1.1 Creating a Simple Availability Calendar

A simple availability calendar defines a single availability rule (for example, Sunday through Saturday from 12:00 a.m. to 10:00 p.m.).

To create a simple availability calendar:

1. Log in as an administrator to Oracle9*i*AS Portal.
2. Click the **Builder** button at top of the Portal main page.
3. On the resulting page, click the **Administer** tab.
4. Under the **Oracle Reports Security** portlet, click **Create Reports Simple Calendar Access**.

5. Specify an internal name, display name, and Portal DB Provider for the calendar:

   - In the **Name** field, enter a unique name that will identify the availability calendar internally in Portal, for example, MY_CALENDAR. This name must follow the Portal rules for a valid component name, specified in the Portal online help.
   - In the **Display Name** field, enter the name you want to display for this availability calendar when it is exposed through Portal. Unlike the internal name, the display name can have spaces in it.
   - Select a **Portal DB Provider** from the provider list of values. All components added to or created in Portal must belong to a Portal DB Provider. This list contains the names of only those providers with which you have privileges to build components.

   Note: For information on creating a Portal DB Provider, see the Oracle9*i*AS Portal online help.

6. Click the **Next** button.
7. Optionally, enter a description of the calendar under **Description**.
8. Click the **Next** button.
9. On the **Date/Time Availability** page, define the parameters for the calendar:

   Under **Duration**, specify the length of time that comprises a unit of duration (or duration period). For example, if you plan to set this calendar up to allow report access between 9:00 AM to 5:00 PM on a given day, then both **Start** and **End** would be the same month, day, and year, but the hour and minute setting for **Start** would be 9:00 AM and for **End** would be 5:00 PM. In this example, the duration of availability of a report on a given day is from 9:00 AM to 5:00 PM.

   Under **Repeat**, specify how frequently the duration period is repeated:

   - **Occurs only once** means the duration period does not repeat, and associated components are no longer available when the period expires. For example, if you select **Occurs only once** and set a duration period of one year, then the associated components cease to be available after one year.
   - **Yearly** means the duration period restarts each year. If you select **Yearly** and have the same start and end date in your **Duration** setting, but your **Start** hour is set to 9:00 AM and your **End** hour is set to 5:00 PM, then the

Reports components associated with this availability calendar will be available one day a year between 9:00 and 5:00.

- **Monthly** means the duration period restarts each month between the **Start** and **End** dates specified under **Duration**. If you select **Monthly** and have the same date and year in both **Start** and **End**—July 25, 2001—but set the **Start** hour for 9:00 AM and the **End** hour for 5 PM, then the associated components will be available between 9:00 AM and 5:00 PM on the 25th of each month.
- The **by Date/Day** setting applies only to **Monthly**. With **by Date/Day**, you specify whether the duration period is set by the particular date (e.g., always on the 25th through the 29th of the month) or by the particular day(s) (e.g., always on Monday through Friday—which happen this month to fall on the 25th through the 29th).
- **Weekly** means the duration period restarts on a weekly basis between the days specified under **Duration**.
- **Daily** means the duration period restarts each day between the hours specified under **Duration**.
- **Frequency** fills in the missing value for the phrase: Repeat every *n* (years, months, weeks, days—depending on what you selected under **Repeat**). For example, if you set the duration period to repeat weekly, then set **Frequency** to 2, the duration period restarts every two weeks, or every other week.
- Optionally, check **Repeat Until** and assign a termination date/time for the calendar. Availability for all associated Reports components ends on the **Repeat Until** date/time.

  Note that no validation is run on your calendar. If the duration period exceeds the repetition setting, no error message will be generated. For example, if you set the duration period for 10 days and the repetition for weekly, the periods will overlap, but you will not be notified of the overlap.

10. Click the **Next** button.
11. On the **Summary** page, click the **Show Calendar** button to preview your availability calendar. If you wish to change some settings, click the **Previous** button and make your changes.
12. On the **Summary** page, click the **Finish** button to complete the availability calendar.
13. Click the **Close** button to return to the **Oracle Reports Security** page.

5.4.1.2 Creating a Combined Availability Calendar

A combined availability calendar combines two or more availability calendars into a single availability calendar. This is useful when you want to set up an availability period, then exclude specific days, such as holidays, from that period.

When you combine calendars, you can indicate that all the days on one of them be excluded from all the days on the other. For example, one calendar could describe availability Monday through Friday; another could describe availability only on Wednesday. You could combine these, excluding the Wednesday calendar, so that the combined calendar describes availability Monday, Tuesday, Thursday, Friday.

Conceivably, you could create a simple calendar that covers the weekdays of an entire year, then multiple additional simple calendars, where one excludes New Years, another excludes a second holiday, another excludes a third, and so on. You could combine all these calendars, excluding all the holiday calendars, so that components were available only on the days your company is open for business, between certain times of day, throughout the year.

To combine availability calendars:

1. Log in as an administrator to Oracle9*i*AS Portal.
2. Click the **Builder** button at the top of the Portal main page.
3. On the resulting page, click the **Administer** tab.
4. Under the Oracle Reports Security portlet, click **Create Reports Combined Calendar Access**.
5. Specify an internal name, display name, and Portal DB Provider for the calendar:
   - In the **Name** field, enter a unique name that will identify the combined availability calendar internally in Oracle9*i*AS Portal, for example, MY_COMBINED_CALENDAR. This name must follow the Portal rules for a valid component name set out in the Portal online help.
   - In the **Display Name** field, enter the name you want to display for this combined availability calendar when it is exposed through Portal. Unlike the internal name, the display name can have spaces in it.
   - Select a **Portal DB Provider** from the provider list of values. All components that you add to or create in Portal must belong to a Portal DB Provider. This list contains the names of only those providers with which you have privileges to build components.

Note: For information on creating a Portal DB Provider, see the Oracle9*i*AS Portal online help.

6. Click the **Next** button.
7. Optionally, enter a description of the calendar under **Description**.
8. Click the **Next** button.
9. On the **Selection** page, highlight the calendars on the **Availability Calendars** list that you want to combine.

   On **Windows**, control-click to select multiple calendars.

   On **UNIX**, click each calendar you want to select.

   This page lists the availability calendars that have been defined for the same Portal DB Provider under which you are creating this combined availability calendar.
10. Click the right arrow to move the selected calendars to the **Selected Availability Calendars** list.
11. Click the **Next** button.
12. On the **Exclude** page, highlight the calendar(s) on the **Availability Calendars** list whose dates you want to exclude.

    On **Windows**, control-click to select multiple calendars.

    On **UNIX**, click each calendar you want to select.

    These are the calendars with dates on which you wish to withdraw availability.
13. Click the right arrow to move the selected calendars to the **Excluded Availability Calendars** list.
14. Click the **Next** button.
15. On the Summary page, click the **Show Calendar** button to preview your calendar.

    If your exclusion isn't showing up, select a different view. For example, instead of the monthly view, select the weekly.

    If you want to change the combination, close the calendar and click the **Previous** button one or more times to return to the desired page.

16. Click **Finish** to complete creation of the combined calendar.

You can combine this calendar with other calendars or apply it "as is" to registered Reports components.

5.4.2 Registering a Printer

It is not required that you register a printer within the security framework of Oracle9*i*AS Portal. You can run a report on any printer as long as it is available to the Reports Server. However, you might want to confine Oracle9*i*AS Portal users to a subset of those printers, constrain the use of a printer for certain periods of time, or identify a particular printer to be used for printing output of certain reports.

Printer registration with Portal is meaningful for reports that you run through Portal as well as those you run through a stand-alone URL.

Once printers are registered within Oracle9*i*AS Portal, you can associate them with an Oracle9*i*AS Reports Server. Many printers can be registered. However, only printers associated with particular Oracle9*i*AS Reports Servers are available to print when you register a report with Portal and choose those Reports Servers.

You can choose to restrict even further the registered subset of printers that a registered report can be sent to. For example, an Oracle9*i*AS Reports Server might be connected to the printer in the office of the CEO, but its selection should not be available to employees running the general ledger report, unless it is the CEO who is running the report. A subset of printers can be listed to the Oracle9*i*AS Portal user running a report request to select where output should be sent.

To register a printer:

1. Log in as an administrator to Oracle9*i*AS Portal.
2. Click the **Builder** button at the top of the Portal main page.
3. On the resulting page, click the **Administer** tab.
4. Under the **Oracle Reports Security** portlet, click **Oracle Reports Security Settings**.
5. Under the **Reports Printer Access** portlet, click **Create Reports Printer Access**.
6. On the resulting page, the **Name**, **Display Name**, and **Portal DB Provider** fields are filled in with default values. Change these to your desired values:
    - In the **Name** field, enter a unique name that will identify the printer internally in Oracle9*i*AS Portal, for example, MY_PRINTER. This name

must follow the Portal rules for a valid component name set forth in the Portal online help.

- In the **Display Name** field, enter the name you want to display for this printer when it is exposed to your users through Portal. Unlike the internal name, the display name can have spaces in it.
- From the **Portal DB Provider** list of values, choose the Portal DB Provider that will own the printer. All components you add to or create in Portal must belong to a Portal DB Provider. This list contains the names of only those providers with which you have privileges to build components.

7. Click the **Next** button.
8. On the resulting page, fill in desired values:
   - In the **OS Printer Name** field, enter the operating system printer name, for example:

     UNIX: <*printer_name*>

     Windows: \\<*printer_server*>\<*printer_name*> (for a remote printer)
     <*printer_name*> (for a local printer)

     This printer must be available to the Reports Server.

     Note: Printer availability is set via the operating system on the Report Server's host machine.

   - Optionally, in the **Description** field, type a description of the printer.
9. Click the **Next** button.
10. Optionally, choose an availability calendar to restrict the days and times the printer can be used. You'll find more information about availability calendars in Creating an Availability Calendar.
11. Click the **Finish** button.

    The resulting page summarizes your Portal settings for this printer. On this page, you can edit your settings, get detailed registration information about the printer, or delete it from Portal altogether.
12. Click the **Close** button to close this page and return to Portal's **Oracle Reports Security** page.

You have completed registering a printer with Portal. This registration is meaningful for reports that are run through Portal as well as those run outside of Portal.

5.4.3 Registering a Reports Server

Before you can define access controls for the Reports Server, you must register your server within Portal (i.e., this is required). Registration provides Portal with the information it needs to identify and locate all available Reports Servers. This becomes particularly important when you register individual reports; during this process you are required to choose from a list of Reports Servers, and servers must be registered to appear on this list.

This section describes how to register Reports Servers in Oracle9*i*AS Portal.

To register a Reports Server:

1. Log in as an administrator to Oracle9*i*AS Portal.
2. Click the **Builder** button at the top of the Portal main page.
3. On the resulting page, click the **Administer** tab.
4. Under the **Oracle Reports Security** portlet, click **Oracle Reports Security Settings**.
5. Click **Create Reports Server Access**.
6. On the resulting page, the **Name**, **Display Name**, and **Portal DB Provider** fields are filled in with default values. Change these to your desired values:
   - In the **Name** field, enter a unique name that will identify the Reports Server internally in Oracle9*i*AS Portal, for example, MY_REPORTS_SERVER. This name must follow the Portal rules for a valid component name set forth in the Portal online help.
   - In the **Display Name** field, enter the name you want to display for this server when it is exposed to your users through Portal. Unlike the internal name, the display name can have spaces in it.
   - From the **Portal DB Provider** list of values, choose the Portal DB Provider that will own the server. All components you add to or create in Portal must belong to a Portal DB Provider. This list contains the names of only those providers with which you have privileges to build components.
7. Click the **Next** button.
8. On the **Server Definition** page:

- In the **Reports Server Name** field, enter the name of the Reports Server. This is the unique name you gave the server when you installed it, provided you haven't changed that name. (If so, enter the new name.)
- In the **Description** field, optionally, enter a description of this Reports Server.
- In the **Oracle Reports Web Gateway URL for JSP reports** field, enter the URL that points to the location of your JavaServer Page (JSP) files. For example:

  `http://<your_web_server>.<domain>:<port>/<virtual_path_to_JSPs>/`

  Note: For information on specifying the virtual path, see Chapter 3, "Configuring Oracle9iAS Reports Services".

- In the Oracle Reports Web Gateway URL for RDF reports field, Enter the URL that points to the location of the Reports Servlet. For example:

  `http://<your_web_server>.<domain>:<port>/<virtual_path_to_rwservlet>/rwservlet`

  Note: For information on specifying the virtual path, see Chapter 3, "Configuring Oracle9iAS Reports Services".

- If you want only the report definition files that are registered in Portal to run when requested, check the **Run Only Registered Report Definition Files** check box.

  Leave this box unchecked if you want this Reports Server to accept any report definition file, including those not registered in Portal, as long as the user who submits the report request has access privileges to this Reports Server.

- From the **Printers** list, select the printer(s) that you want to make available to this Reports Server. Multiple selections are possible:

  On **Windows**, control-click the printers you want to select.

  On **UNIX**, click the printers you want to select.

9. Click the **Next** button.

10. On the **Destination Types** page, enter custom destination types for this Reports Server. Note that default destinations that come with your Oracle9*i*AS installation need no further configuration in Portal.

    Note: For information on custom destination types, see Chapter 4, "Configuring Destinations for Oracle9iAS Reports Services".

11. Click the **Next** button.
12. On the **Availability Calendar** page, optionally choose an availability calendar to control the days and times this Reports Server is available to accept report requests.
13. Click the **Finish** button.
14. Click the **Close** button.

    This returns you to the **Oracle Reports Security Setting** page.

You have registered an Oracle9*i*AS Reports Server. Now you can register a report.

5.4.4 Registering a Report

Registering a report is a required step that allows you to define who can run a report, when a report is available to run, which server(s) can be used to process report requests, how a report is delivered, and the printer(s) to which a report can be sent.

In addition to using registration to designate which users have access to a report, you can also specify, via a Portal parameter form, how users are to interact with the report.

In the Reports Builder, users create user parameters. Then, in Portal, users specify the names of these parameters, enabling end users to select or enter values for these parameters when they run the report. At runtime, the Reports Server disregards parameters you set in Portal that were not also defined in the Reports Builder at design time.

Use the parameter settings available through Portal to duplicate or create a subset of those parameters defined in the Reports Builder at design time. This way you get parameter coverage when you run the report via Oracle9*i*AS Portal.

Registering a report within Oracle9*i*AS Portal creates an Oracle9*i*AS Portal component that can be deployed as a portlet through Portal. We recommend that

you register only one instance of a report file in Portal. If you define multiple Portal Reports objects for one report, all are given security checks at runtime. If any of them fail the security check, then all fail it, and the job will not run.

To register a report:

1. Log in as an administrator to Oracle9*i*AS Portal.
2. Click the **Builder** button at the top of the Portal main page.
3. On the resulting page, click the **Administer** tab.
4. Under the **Oracle Reports Security** portlet, click **Oracle Reports Security Settings**.
5. Under the **Reports Definition File Access** portlet, click **Create Reports Definition File Access**.
6. The **Name**, **Display Name**, and **Portal DB Provider** fields are filled in with default values. Change these to your desired values:
   - In the **Name** field, enter a unique name that will identify the report internally in Portal, for example, MY_REPORT. This name must follow the Portal rules for a valid component name set forth in the Portal online help.
   - In the **Display Name** field, enter the name you want to display for this report when it is exposed through Portal. Unlike the internal name, the display name can have spaces in it.
   - From the **Portal DB Provider** list of values, choose the Portal DB Provider that will own the report. All components added to or created in Portal must belong to a Portal DB Provider. This list contains the names of only those providers with which you have privileges to build components.
7. Click the **Next** button.
8. Enter or select information as follows:
   - From the **Reports Servers** list of values, select the Reports Server(s) to be available to run this report.

     To select multiple servers:

     On **Windows**, control-click each server.

     On **UNIX**, click each server.
   - In the **Oracle Reports File Name** field, enter the name you gave the report in the Builder, including its extension.

The report definition file can be an .rdf, .jsp, or .xml file. If the path to this file is included in your REPORTS_PATH environment variable, do not enter it here. If the path is not included in REPORTS_PATH, include it here along with the filename. Do this for all report definition files except those you will run as stand-alone JSPs.

- Optionally, type a description of this report in the **Description** field.
- In the **Execute** field, choose between **via servlet** and **as JSP**.

  Choose **via servlet** if you plan to run the report via the Reports Servlet. Choose **as JSP** if you will run a JSP report stand-alone, without going through the Reports Servlet.

  The selection you make here will affect the choices that are available on the next wizard page.

9. Click the **Next** button.
10. On the **Required Parameters** page set required runtime information. These settings are only applicable if running through the Reports Servlet. At runtime, anywhere you have indicated multiple selections, using control-click, a list of values will be offered to your users, from which they can set their own runtime information:

    - **Types** specifies the destination types acceptable for this report. Choose among Cache, File, Mail, OraclePortal, Printer, or custom destination types. If the server you associate with this report supports custom destination types, which you indicated when you registered the Reports Server in Portal, the types you indicated will display on this list.
    - **Formats** defines the acceptable output format(s) for this report. Choose among HTML, HTMLCSS, PDF, XML, RTF, Delimited, PostScript, and Character
    - **Printers** specifies the registered printer(s) to which this report can be sent. The printers that appear on this list are determined by those you chose when you set up access to the Reports Server(s) you are associating with this report. All registered printers are listed. When users choose a Reports Server on the runtime parameter form, only those printers that are associated with the selected Reports Server and that are accessible to those users are listed.
    - **Parameter Form Template** specifies the template that will define the look and feel of the Portal parameter form from which you will run the report. This value is used only when the report is exposed through the Portal.

Choose a template from the list of values. Click **Preview Template** to see what the selected template looks like.

Note: For information about adding your own templates to this list, see the Oracle9*i*AS Portal online help.

---

11. Click the **Next** button.
12. On the **Optional Parameters** page define limits for the report's existing parameters.
    - **Name** is the name of the system or user parameter on which you wish to restrict the values available to users, for example, SALES_REGION or COPIES.
    - **Display Name** is the name used to identify the parameter on the runtime parameter form.
    - **LOV** is the name of a predefined list of values to be included in the parameter form. The list must already exist. For information on creating a list of values, see the Portal online help.
    - **Low Value** is the lowest value you wish to set for a range of values.
    - **High Value** is the highest value you wish to set for a range of values.
    - Click **More Parameters** if you wish to add more rows for additional parameters and values.
13. Click the **Next** button.
14. Optionally, enter the name of the availability calendar (or choose from a list of calendars).

    Use the availability calendar to limit the days and times this report can be run. For more information, see Section 5.4.1, "Creating an Availability Calendar".
15. Click the **Next** button.
16. Optionally, enter a validation trigger to create a programmatic restriction.

    Use validation triggers to create conditional restrictions that cannot be defined on either the **Required Parameters** page or the **Optional Parameters** page. Validation triggers are PL/SQL functions.

The function that you specify as a validation trigger must return a boolean value (TRUE or FALSE). If the function returns TRUE, the job is run. If the function returns FALSE, an error message is displayed and the job is not run.

17. Click **Finish** to close the wizard and complete report registration.

    The resulting page summarizes your registration information and provides the opportunity to perform additional actions on your report.

    - Click **Customize** to view the report's runtime parameter form.

      Table 5–2 summarizes the options available on this page.

Table 5–2 Options on the runtime parameter form

Option	Description
Run Report	Click to run this report with the specified parameter values.
Save Parameters	Click to save the parameter value selections.
Server	Choose the Oracle Reports Server that you want to receive this report request. Only the servers you chose from the Report Name and Servers page are displayed in this list box.
Printer	Choose the printer that you want to print your report output. Only the printers you chose from the Required Parameters page are displayed in this list box.
Destype	Choose the destination type. Only the destination types you chose from the Required Parameters page are displayed in this list box.
Desformat	Choose the destination format. Only the destination formats you chose from the Required Parameters page are displayed in this list box.
Desname	Enter the name of the output file when Destype is FILE, or enter the e-mail addresses when the Destype is MAIL. Separate multiple addresses with commas. The destination name is required when you choose FILE or MAIL as the Destype.
SSOCONN	Enter one or more SSO connection strings. Separate multiple strings with a comma (but no spaces).
Visible to user	Check each parameter that you want to make available in the runtime parameter form when users run this report request. If the box in not checked, then the parameter is not displayed to users.
CGI/Servlet Command Key	Optionally, enter the key from the cgicmd.dat file that identifies the command line to run for this report.

Table 5–2 Options on the runtime parameter form

Option	Description
Additional User Parameters	Use this field to enter additional user parameters. For example, you can use this field to enter the path and name of the distribution XML file that defines how this report should be distributed. Use the same syntax you would use to specify these values in a command line request or within the cgicmd.dat file. If you wish to enter multiple additional parameters, simply separate each entry with a space. For more information about the distribution XML file, see Chapter 9, "Creating Advanced Distributions".

6

Reports Server Clusters

A cluster is a virtual grouping of servers into a community for the purpose of sharing request processing efficiently across members of the cluster. Clustering in Oracle9*i*AS Reports Services is peer-level, which means that all members of the cluster take equal responsibility for sharing and processing incoming requests. If one member is shut down, the other members carry on managing the request load. If the output is present in one member's cache, another member can use it. There is no single-point-of-failure, where one machine's malfunction brings the whole system down.

This chapter contains information about enrolling a server in a cluster and benefits of clustering servers together. It contains the following sections:

- Cluster Overview
- Setting Up a Cluster

6.1 Cluster Overview

Assume you have the following servers:

```
serverA.cluster1
serverB
serverC.cluster1
```

`ServerA.cluster1` and `serverC.cluster1` are members of the same cluster called `cluster1`. They cooperate to process requests from a client. If a client sends a synchronous request to `serverA.cluster1` and it does not have an idle engine of the specific job type, then it checks to see if `serverC.cluster1` does. If `serverC.cluster1` does have an idle engine, then `serverA.cluster1` passes the request to `serverC.cluster1` for processing.

In this example, `ServerB` is a stand-alone server and cannot receive processing requests from other servers, nor can it send processing requests to other servers.

You can have an unlimited number of servers in a cluster. If a cluster member is shut down, then it redistributes its pending synchronous jobs to another server in the cluster. As long as one server in the cluster is running, the cluster is working.

When the cluster is making its decision as to where an upcoming scheduled or immediate request should be processed, it prioritizes according to the following criteria:

1. Does any server in the cluster have information in cache that matches the request?
2. Is there a current, similar job in the queue?
3. Is an idle engine of the particular job type available?
4. Is the number of currently active engines less then the MAXENGINE number specified for the server for that job type?

Both stand-alone and clustered servers share the same, basic configuration. The cluster has no special configuration requirements, beyond needing to share a common cluster name and common public and private keys. There are no limitations on the platform used, the number of servers in the cluster, or the location of the server. There is no requirement to share resources within the cluster servers.

Engine output is locally cached in a particular Reports Server within the cluster, but it is also known and available to the entire cluster. If a server is down, that server's cached files are no longer available for reuse. This means that another server within the cluster must rerun the request to obtain the output. When the server is running again, all of the cached files become available due to the persistent state of the cache.

6.2 Setting Up a Cluster

Clustering in Oracle9*i*AS Reports Services is as easy as naming all member servers with the same "dot extension," for example <*server_name*>.cluster or <*server_name*>.xyz, and ensuring that all member clusters share the same public and private key.

This section covers renaming your Reports Server, creating and specifying public and private keys, and submitting requests to a cluster. It contains the following sections:

- Renaming a Reports Server

- Generating New Public and Private Keys
- Entering Public and Private Keys in the Server Configuration File
- Restarting the Reports Server
- Submitting a Request to a Cluster

6.2.1 Renaming a Reports Server

It is likely that you are reading this material after you've already set up at least one Reports Server. If this is the case, you'll need to change the name of your server to add the cluster name to the server name.

Note: If you haven't yet installed your servers, when you do install them you must give them all different server names but the same cluster name, for example servernameA.cluster1, servernameB.cluster1.

To rename a Reports Server:

1. If the server is running, shut it down:
   - If it's running on NT as a service, stop it through the Services control panel.
   - If it's running on NT through a server executable, or on UNIX through a shell script, click the **Shutdown** button in the Oracle Reports Server dialog box.
   - If it's running from a command line on NT or UNIX, at the command prompt enter the following command for Windows or UNIX:

     This shuts down the server normally:

     ```
     rwserver server=<server> shutdown=normal authid=<admin/pword>
     ```

     This shuts down the server immediately:

     ```
     rwserver server=<server> shutdown=immediate authid=<admin/pword>
     ```

     This shuts down the server without displaying any related messages:

     ```
     rwserver server=<server> shutdown=normal authid=<admin/pword> batch=yes
     ```

The keywords used with the `rwserver` command are described in Appendix A, "Command Line Arguments".

2. If you have custom configuration settings in your Reports Server configuration file (<*server_name*>.conf), rename this file to the new cluster name (<*server_name*>.<*cluster_name*>.conf).

   You'll find the configuration file in the following path on UNIX and Windows:

   ORACLE_HOME\reports\conf\<*server_name*>.conf

   If you don't have custom configuration settings in your Reports Server configuration file, a new configuration file with the new name will be generated automatically when you restart the renamed server(s).

3. Rename the Reports Server in all affected files, giving each cluster member the same cluster name.

   - Open the servlet configuration file (rwservlet.properties) and respecify the server name to include the name of your cluster. For example:

     SERVER=<*server_name*>.<*cluster_name*>

     You'll find the servlet configuration file on both Windows NT and UNIX in the same path:

     ORACLE_HOME\reports\conf\rwservlet.properties

   - If you run the server as an NT service, to rename the server you must uninstall and reinstall the service:

     To uninstall the NT service, at the command prompt enter:

     rwserver -uninstall <*server_name*>

     To reinstall the NT service, at the command prompt enter:

     rwserver -install <*server_name*>.<*cluster_name*>

     Note: Reinstalling the server also starts it up. You may want to shut it down until you have renamed all server cluster members, then start them all up together once you've set up your cluster. You'll find information on shutting the server down in Chapter 2, "Starting and Stopping Oracle9iAS Reports Services".

Before you restart your Reports Server(s), you may generate server public and private keys and enter the resulting information in each member server's configuration file. How to do this is discussed in the next sections.

6.2.2 Generating New Public and Private Keys

The server public and private key files aid with message encryption and authentication between cluster members. The default files are stored in the rwrun.jar file in the following path (on both UNIX and Windows):

ORACLE_HOME\reports\jlib\rwrun.jar

Each member of a cluster must have the same public and private key files specified in their configuration files (<*server_name*>.<*cluster_name*>.conf). To ensure that your cluster members share exclusive public and private key files, generate new versions of them when you set up your cluster. Servers that will not be members of the cluster can go on using the default keys provided with Oracle9*i*AS Reports Services.

To generate new public and private key files, at the command prompt, enter the following command:

```
java oracle.report.utility.KeyManager <public_key_file_name> <private_key_file_
name>
```

You can generate these files to specific directories by specifying the desired path in the command line along with the new public and private key file names. If you just specify the file name in the command line, the key files will be generated in the current directory.

6.2.3 Entering Public and Private Keys in the Server Configuration File

Once you generate new public and private key files, you must enter that information into all cluster members' Reports Server configuration files. You'll find each cluster member's version of this file in the following path for both UNIX and Windows on each server's host machine:

ORACLE_HOME\reports\conf\<server_name>.<cluster_name>.conf

To change public and private key files, go to the *connection* element in the server configuration file, and change (or add) entries for the *cluster* sub-element as follows:

```
<cluster publicKeyFile="path and filename of new public key"
privateKeyFile="path and filename of new private key">
```

You'll find more information about the *connection* element in Chapter 3, "Configuring Oracle9iAS Reports Services".

6.2.4 Restarting the Reports Server

Once you have renamed your cluster members and respecified a common public and private key for each, you may start up your Reports Servers to activate the cluster.

To start up a Reports Server:

- If you're starting the Reports Server as an NT service, open the Service control panel, and start the service.
- If you're starting the Reports Server from a command line, at the command prompt, enter the following command:

  On **Windows NT**:

  ```
  rwserver server=<server_name>.<cluster_name>
  ```

  On **UNIX**:

  ```
  rwserver.sh server=<server_name>.<cluster_name>
  ```

Once you've renamed your cluster members, respecified your public and private keys, and restarted your Reports Servers, you've completed the process of setting up your cluster.

6.2.5 Submitting a Request to a Cluster

To submit a request to a cluster:

In the Reports Servlet or JSP, specify:

```
server=<cluster_name>
```

For example, if you have two cluster members—one named mercury.cluster1, the other named venus.cluster1—then your `server` entry would be:

```
server=cluster1
```

The Reports Servlet or JSP will find a running Reports Server in the cluster and send the request to that Reports Server. Depending on the cache match or the server load, that Reports Server will either handle the request or redirect it to another server in the cluster.

7

Data Source Single Sign-On

Now that pluggable data sources are part of the benefits offered to you through Oracle9*i*AS Reports Services, you may want to spare your users having to log in to multiple data sources in order to run one job. You can do this through single sign-on (SSO).

SSO enables you to establish unique identities for each user that are tied to resources unique to that user. The user's resources contain key-identified connection strings for accessing different data sources. The user is uniquely identified through his or her once-per-session login, and the login references the user's resources to ensure that he or she has access to the appropriate data sources, without users having to enter this information themselves.

SSO is made possible through the partnership of Oracle9*i*AS Reports Services, Oracle Internet Directory, and the Oracle Login Server, all delivered through the Oracle9*i* Application Server.

SSO can be implemented only in a secure server environment. This means that you must have a security policy in place in your Reports Server configuration file before you can consider implementing SSO with Reports.

Note: Security settings are discussed in the following places: Chapter 3, "Configuring Oracle9iAS Reports Services" tells you how to specify the Java class that defines the security policy for the server; Chapter 5, "Controlling User Access" tells you how to apply security settings to servers, printers, and reports through Oracle9*i*AS Portal; Appendix A, "Command Line Arguments" provides information about the `SSOCONN` command line argument.

With SSO, your administrator establishes a user identity for each user. The administrator does this in the Oracle Internet Directory (OID), through its user interface, the Delegated Administration Service (DAS), or through Oracle9*i*AS Portal (once you register a user in Portal, that information is saved to the OID).

The user identity is comprised of the user name and password. Once users are established, the administrator can assign resources to them, comprised of connection strings to different data sources. At login, users will enter their user names and passwords (their user identities), which will in turn have access to all resources associated with those identities. The Oracle Login Server issues a session cookie that effectively acts as a key that opens all authorized doorways for that session.

This chapter discusses data source SSO. It includes information about the architecture of the SSO environment and helpful tips for setting up user resources (connection strings) for SSO.

Note: For detailed information about the requirements and procedures required for setting up SSO-related components, such as the Oracle Internet Directory, see the *Oracle Internet Directory Administrator's Guide* and the *Oracle HTTP Server Administrator's Guide* on the Oracle9*i*AS documentation CD and on the Oracle Technology Network (http://otn.oracle.com).

This chapter contains the following main sections:

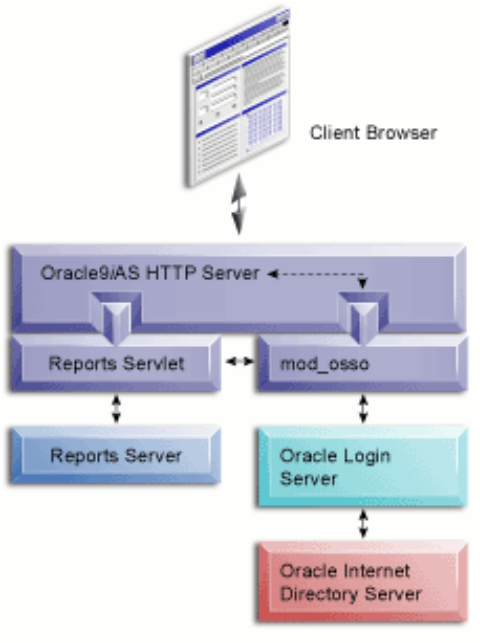
- SSO Architecture
- Methods for Setting Up User Connection Strings

7.1 SSO Architecture

7.1.1 SSO Components

Figure 7–1 provides an overview of SSO component architecture.

Figure 7–1 SSO architecture



The components of the SSO environment include:

- A client **Web browser**
- **Oracle HTTP Server**

  Within the context of SSO, the Oracle HTTP server acts as an intermediary between the client and mod_osso.

- The **mod_osso** module

  The mod_osso module is an Oracle Login Server partner application that provides authentication support. In the Reports environment, it acts as an intermediary between the Reports Servlet and the Oracle Login Server.

- The **Reports Servlet**

  The Reports Servlet is a component of Oracle9*i*AS Reports Services that runs inside of the Oracle HTTP Server's Oracle9*i*AS Containers for Java 2 Enterprise Edition (OC4J). Within the SSO context, the Reports Servlet acts as an intermediary between mod_osso and the Reports Server.

- The **Reports Server**

  The Reports Server processes client requests, which includes ushering them through authentication and authorization checking, scheduling, caching, and distribution. Within the context of SSO, the Reports Server informs the Reports Servlet whether or not the Reports Server is secure.

- The **Oracle Single Sign-on (SSO Server)**

  The Oracle SSO Server is responsible for managing users' single sign-on sessions. The mod_osso module is the intermediary between the Reports Servlet and the SSO Server. The mod_osso module passes authentication requests to the server. The server verifies users' login credentials by looking them up in the Oracle Internet Directory (OID).

- The **Oracle Internet Directory** (OID)

  OID is a repository for user information. This includes personal information, such as home address, e-mail, and telephone number, as well as access information, such as user IDs and connection strings. OID is Oracle's native Lightweight Directory Access Protocol version 3 (LDAPv3) service. It's built as an application on top of the Oracle9*i* relational database. All LDAP-enabled products that Oracle produces, including Oracle9*i*, Oracle9*i*AS, and Oracle9*i*AS Portal, ship as supported Oracle Internet Directory clients.

- **Oracle Delegated Administration Service (DAS)**

  DAS provides the user interface service for management of users and groups in OID. You use the DAS interface to enter user and connection string information into OID. DAS is a servlet written as an OID client. You access it through a browser, through a URL you set up during installation.

7.1.2 SSO Transactions

A transaction in an SSO environment follows these steps:

1. A request is sent through the client browser to the Oracle HTTP Server.
2. The HTTP Server passes the request to the Reports Servlet (with the job's runtime URL), which, in turn, calls the Reports Server to verify that the Reports Server is secure.
3. The Reports Server responds to the servlet: if yes, the Reports Servlet sends message 401 back to mod_osso, requesting authentication; if no, the Reports Servlet processes the request.

4. The mod_osso module gets message 401, connects with the Oracle Login Server, and checks whether the user has already been authenticated.
5. The Login Server responds: if no, the Login Server sends a login screen to the client; if yes, the original URL request goes through along with the user's identity, and the request is handled by the Reports Servlet.
6. The client sends login information to the Login Server, which checks it against information in the Oracle Internet Directory.
7. If the login information is accurate, the original URL request goes through and the process is complete. If the login information is not accurate, an error is returned, and the client is either prompted to retry or the process stops. The allowable number of retries is specified through SSO Server configuration.

7.2 Methods for Setting Up User Connection Strings

Although the Oracle Internet Directory (OID) provides tools that enable you to batch load users from an LDAP source to the OID, there currently are no tools for doing the same for those users' connection strings (the passwords and schema IDs that allow users to access data sources). Consequently, this information must be entered manually, or a procedure must be developed to handle it. (A knowledgeable LDAP programmer can create a procedure that will populate the resources in OID.)

In the presence of a large user base, this task can be daunting.

Fortunately, there are a couple of methods wherein each user enters his or her own connection string information. This section provides an overview of those methods.

7.2.1 Initial Requirements

To begin with, both methods require that your users are already entered into the OID. If you are new to the OID, and you have your user base entered in some other LDAP data source, you can use the tools OID provides to batch load your users.

Note: See the *Oracle Internet Directory Administrator's Guide* for information on batch loading. You'll find it on the Oracle9*i*AS documentation CD and on the Oracle Technology Network (http://otn.oracle.com).

If you do not have users in an LDAP data source, you must enter them manually.

7.2.2 Method 1: Giving Users Access to the OID

The first method for getting users to enter their own connection string information is to give them access to OID. The user interface into OID is called Oracle Delegated Administration Service, or DAS.

Note: Before users can access DAS, an administrator must have already entered a user identity in the OID for each user. This can be done by batch loading information that is already entered into an LDAP directory in some other source.

See the *Oracle Internet Directory Administrator's Guide* for information on batch loading. You'll find it on the Oracle9*i*AS documentation CD and on the Oracle Technology Network (http://otn.oracle.com).

During Oracle9*i*AS installation, you specify the location of DAS. When you provide users access to DAS, you do so by giving them a URL that points to this location.

Once in DAS, users can enter their own information via the Resource Assignment section of the Extended Preferences tab. One possible hitch with this scenario is that, for the Extended Preferences tab to appear to users, there must already be a resource in place.

You could enter default resources for your user base, but this might also prove too time consuming.

7.2.3 Method 2: Assigning Connection Strings and Letting Users Input at Login

The second method is probably more secure and more efficient. It's more secure in that it does not require that users make a direct entry into DAS. It's more efficient in that the information entry is just an integral part of sending a report request.

This method involves letting users enter their own information the first time they log in to a data source.

The OID administrator sends an e-mail to each user with a URL to a report. Each e-mail includes a unique password and schema connection string for the recipient and contains instructions to the user to use that connection string.

Note: For reports that make use of multiple data sources, multiple connection strings should be provided. Using the technique described in this section, you can send one e-mail with multiple connection strings or one e-mail per connection string.

The URL includes the SSOCONN command line option, which calls a connection key or keys that do not yet exist. For example:

```
ssoconn=mykey1/DB,mykey2/PDSApp
```

Each URL can call the same connection key (e.g., mykey1). Because, rather than the key name, the unique information is the data that each user enters.

The user enters the connection information when prompted, and that information is automatically saved in the OID.

In the future, when users run reports, they'll be prompted for their user identity, that is, their user name and password. The resulting cookie will contain the user identity, which will be sent to the Reports Servlet to get connection string information (resources) in OID.

Part II

Sending Requests to the Server

Part II provides detailed, practical information about publishing reports, including how to run requests; how to set up sophisticated, automatic report distributions; how to customize reports at runtime via XML customization files, and how to use database triggers to automatically invoke reports.

Part II includes the following chapters:

- Chapter 8, "Running Report Requests"
- Chapter 9, "Creating Advanced Distributions"
- Chapter 10, "Customizing Reports with XML"
- Chapter 11, "Event-Driven Publishing"

8

Running Report Requests

This chapter discusses various ways to send report requests to the Reports Server. It includes the following sections:

- The Reports URL Syntax
- Report Request Methods
- Publishing a Report Portlet in Oracle9iAS Portal
- Specifying a Report Request from a Web Browser
- Sending a Request to the URL Engine
- Scheduling Reports to Run Automatically
- Reusing Report Output from Cache
- Using a Key Map File

8.1 The Reports URL Syntax

This section provides quick reference information on formulating a URL for publishing a report. It covers three deployment types:

- Servlet
- JSP
- CGI (for backward compatibility only)

The information is largely the same for both Windows and UNIX environments. Differences are noted.

8.1.1 Servlet

The syntax for the URL of a report run via the Reports Servlet is:

```
http://<web_server>.<domain_name>:<port>/<alias>/rwservlet?<parameters>
```

Table 8–1 lists and describes the components of the servlet URL.

Table 8–1 Components of a URL that calls the Reports Servlet

Component	Description
<*web_server*>	The name you gave the Oracle HTTP Server when you installed it.
<*domain_name*>	Your organization's domain name.
<*port*>	The port number on which the Oracle HTTP Server listens for requests. When no port is specified, the default is used (80).
<*alias*>	The virtual path that stands in for the absolute path to the files a URL will access.
rwservlet	Invokes the Oracle9*i*AS Reports Services servlet.
?	Identifies the beginning of the command line arguments.
<*parameters*>	All the command line arguments, or the key to the key map file where command line arguments are specified.

The URL that calls the Reports Servlet could look like this:

```
http://neptune.world.com:80/reports/rwservlet?keyname
```

Keyname refers to a command line listed under a unique header (the key name) in the cgicmd.dat file. Note that this works differently for JSP files, which use the keyword / value pair `cmdkey=value` to specify key names for command lines that are stored in the cgicmd.dat file. You'll find more information about using key mapping in Section 8.9, "Using a Key Map File".

Using the servlet does not mean that you cannot also use Reports JSP files, if the JSP files contain both Web and paper layouts. When you run the report, specify the servlet in the URL and call the JSP with the command line argument: `report=<myreport>.jsp`.

For example:

```
http://neptune.world.com:80/reports/rwservlet?report=myreport.jsp&destype=cache&
desformat=html
```

You'll find more information about command lines in Appendix A, "Command Line Arguments".

> **Note:** Yo can also supply these parameters within the JSP file itself.

8.1.2 JSP

The syntax for a JSP-based report URL is:

```
http://<web_server>.<domain_name>:<port>/<alias>/myreport.jsp?<parameters>
```

Table 8–2 lists and describes the components of the JSP-based report URL.

Table 8–2 Components of a JSP-based report URL

Component	Description
<*web_server*>	The name you gave the Oracle HTTP Server when you installed it.
<*domain_name*>	Your organization's domain name.
<*port*>	The port number on which the Oracle HTTP Server listens for requests. When no port is specified, the default is used (80).
<*alias*>	The virtual path that stands in for the absolute path to the files a URL will access.
myreport.jsp	The report *.jsp file you want this URL to execute.
?	Identifies the beginning of the command line arguments.
<*parameters*>	All the command line arguments, and / or the key to the key map file where command line arguments are specified.

The URL used to invoke a JSP-based report could look like this:

```
http://neptune.world.com:80/jsp/myreport.jsp?
```

You can specify a key in the URL that refers to a command line in the cgicmd.dat file that contains additional command line parameters. In this case, you must use the name value pair: `cmdkey=`*`keyname`*. This can appear anywhere in your URL, provided it follows the start of the query string (marked by a question mark). For example:

```
http://neptune.world.com:80/jsp/myreport.jsp?userid=scott/tiger@hrdb&cmdkey=key1
```

In your URL, use an ampersand (&) with no spaces to string parameters together.

Using a JSP does not mean that you cannot also use the Reports Servlet. When you run the report, specify the servlet in the URL and call the JSP with the command line argument: `report=<myreport>.jsp`.

For example:

```
http://neptune.world.com:80/reports/rwservlet?report=myreport.jsp&destype=cache&
desformat=html
```

You'll find more information about command line keywords in Appendix A, "Command Line Arguments". You'll find more information about the cgicmd.dat file in Section 8.9, "Using a Key Map File". For information on choosing whether to use the Reports Servlet to run JSP reports, refer to Chapter 1, "Oracle9iAS Reports Services Architecture".

8.1.3 CGI

> **Note:** The Reports CGI is included in Oracle9*i*AS Reports Services for backward compatibility. We strongly recommend that you deploy your reports with either a servlet or JSP implementation.

The syntax for the URL of a report run via the Reports CGI on **Windows** is:

```
http://<web_server>.<domain_name>:<port>/<alias>/rwcgi.exe?<parameters>
```

And on **UNIX** is:

```
http://<web_server>.<domain_name>:<port>/<alias>/rwcgi.sh?<parameters>
```

Table 8–3 lists and describes the components of a CGI-based report URL.

Table 8–3 Components of a URL that calls the Reports CGI

Component	Description
<*web_server*>	The name you gave the Oracle HTTP Server when you installed it.
<*domain_name*>	Your organization's domain name.
<*port*>	The port number on which the Oracle HTTP Server listens for requests. When no port is specified, the default is used (80).

Table 8–3 Components of a URL that calls the Reports CGI

Component	Description
<*alias*>	The virtual path that stands in for the absolute path to the files a URL will access.
rwcgi.exe	The executable file that invokes the CGI component of Oracle9*i*AS Reports Services. If Reports Services is installed on a UNIX machine, use ".sh" in lieu of ".exe".
?	Identifies the beginning of the command line arguments.
<*parameters*>	All the command line arguments, or the key to the key map file where command line arguments are specified.

The URL used to invoke a CGI implementation could look like this on **Windows**:

```
http://neptune.world.com:80/CGI-BIN/rwcgi.exe?key2
```

And like this on **UNIX**:

```
http://neptune.world.com:80/CGI-BIN/rwcgi.sh?key2
```

If Reports Services is installed on a UNIX machine, use ".sh" in lieu of ".exe". For example:

```
http://neptune.world.com:80/CGI-BIN/rwcgi/sh?key2
```

8.2 Report Request Methods

There are a number of request methods available to you for running your report requests. These include:

- **The `rwclient` command line**

  The `rwclient` command line (`rwclient.sh` on UNIX) is available for running report requests from a command line in a non-Web architecture. It references an executable file that parses and transfers the command line to the specified Oracle9*i*AS Reports Server. It can use command line arguments similar to those used with the Oracle9*i*AS Reports Runtime executable file, rwrun (rwrun.sh on UNIX).

  On Windows, a typical `rwclient` command line request looks like this:

  ```
  rwclient report=<my_report>.rdf userid=<username>/<password>@<my_db>
  server=<server_name> destype=html desformat=cache
  ```

On UNIX, the same command would look like this:

```
rwclient.sh report=<my_report>.rdf userid=<username>/<password>@<my_db>
server=<server_name> destype=html desformat=cache
```

See Appendix A, "Command Line Arguments" for more information about command line arguments.

- **A URL**

  To run a report from a browser, use the URL syntax.The Oracle9*i*AS Reports Servlet (and CGI, for backward compatibility) converts the URL syntax into an `rwclient` command line request that is processed by Oracle9*i*AS Reports Services. You can give your users the URL syntax needed to make the report request from their browser, or you can add the URL syntax to a Web site as a hyperlink. The remainder of this chapter discusses this method in more detail.

- **Via Oracle9*i*AS Portal**

  The Oracle9*i*AS Portal component enables you to add a link to a report in an Oracle9*i*AS Portal page or portlet, or to output report results directly into a portlet. Each report link points to a packaged procedure that contains information about the report request. Oracle9*i*AS Reports Services system administrators use Oracle9*i*AS Portal wizards to create the packaged procedure making it more convenient and secure to publish the report via the Web. Authorized users accessing the Oracle9*i*AS Portal page group simply click the link to run the report. System administrators can run the report directly from the wizard. See the Oracle9*i*AS Portal online help for more information.

  Refer to Publishing a Report Portlet in Oracle9iAS Portal for more information about how to publish your report as a portlet.

- **A packaged procedure**

  SRW.RUN_REPORT is a built-in that runs an Oracle9*i*AS Reports Runtime command. When you specify SRW.RUN_REPORT, set the SERVER argument to the Oracle9*i*AS Reports Services server name to cause the SRW.RUN_REPORT command to behave as though you executed an `rwclient` command.

  Refer to the Oracle9*i* Developer Suite (Oracle9*i*DS) Reports Builder online help for more information.

8.3 Publishing a Report Portlet in Oracle9*i*AS Portal

One of the best ways to publish your reports is through the declarative, secure interface of Oracle9*i*AS Portal. To a expose a report in a portal, you must do the following:

1. **Create a provider for your reports.** This step defines a provider to contain the reports you wish to make available to users in the portal.
2. **Create the report definition file access.** This step makes the report available as a portlet to page designers within the portal by defining the reports properties, in particular the provider that contains it.
3. **Add the report portlet to a page and optionally customize it.** This step makes the report available to users in a portlet on a page and enables the page designer to set the report parameters and schedule it to run automatically.

8.3.1 Creating a Provider for Your Reports

If you do not already have a provider defined to contain your reports, you need to create one. For more information on creating a provider, see the Oracle9*i*AS Portal online help.

Note: The provider that contains your reports must be a database provider and must have the Expose as Provider setting selected on its Access page.

8.3.2 Creating the Report Definition File Access

To make your report available as a portlet, you must do the following:

Note: If you need to create report definition file access for a number of reports, it may be more efficient to batch register them. For more information, see Appendix C, "Batch Registering Reports in Oracle9iAS Portal".

---

1. From the Oracle9*i*AS Portal home page, click the **Corporate Documents** tab.
2. Click **Builder**.
3. Click the **Administer** tab.
4. In the Oracle Reports Security portlet, click **Oracle Reports Security Settings**.

5. In the Reports Definition File portlet, click **Create Reports Definition File Access**.
6. Follow the steps in the wizard and click the question mark in the upper right corner for additional information about the available settings. At the end of the wizard, click **Finish**.
7. Click the **Access** tab.
8. Click **Publish to Portal**.
9. Click **Apply**. Your report has now been added to the Portlet Repository and you can add it to a page.

8.3.3 Adding the Report Portlet to a Page

Once the portlet for your report is in the Portlet Repository, you may add it to any page just as you would any other portlet.

1. From the Oracle9*i*AS Portal home page, click the **Corporate Documents** tab.
2. Click **Builder**.
3. Click the **Build** tab.
4. In the Page Groups portlet, enter the Page Group Name of the page group in which you want to place your report portlet.
5. Create a new page by clicking **Create a Page** or edit an existing page by entering the name of an existing page and clicking **Edit**.
6. If you are creating a new page, follow the steps in the wizard and click the question mark in the upper right corner for additional information about the available settings. Click **Finish** when you are done. If you are editing an existing page, skip to the next step.
7. In the page region where you wish to add your report portlet, click the **Add Portlet** tool. (**Tip:** Hints for each tool will display when you roll your mouse over them.)
8. Drill down through the Portlet Repository to the provider that contains the report portlet. The report portlet is listed in the Portlet Repository under the Portal DB provider to which it belongs. The location of the provider depends on how the Portlet Repository has been organized. If the Portal DB provider is a fairly new provider, it may be under the New page of the Portlet Repository.
9. Click the name of your report portlet to add it to the **Selected Portlets** list.

10. Click **OK**.
11. Click **Customize** in the upper right corner of your report portlet.
12. Enter parameter values in the **Parameter** tab and, if desired, schedule the job to run automatically in the **Schedule** tab.

8.4 Specifying a Report Request from a Web Browser

You can provide the user with the URL syntax needed to make a report request, or you can add the URL syntax to a Web page as a hyperlink.

Note: Another way to publish reports on a Web site is to create an Oracle9*i*AS Portal component. For more information, refer to Oracle9*i*AS Portal online help.

URL syntax can be presented in the following forms:

- Full URL request, for example:

  ```
  http://<your_webserver>.<domain_name>:<port>/<alias>/rwservlet?
  report=myreport.rdf&USERID=<username>/<password>@<my_db>
  &SERVER=<server_name>&DESFORMAT=html&DESTYPE=cache
  ```

  If you require additional command line arguments, then refer to Appendix A, "Command Line Arguments" for a list of valid `rwclient` command line arguments.

- Simplified URL request using key mapping, for example:

  ```
  http://<your_webserver>.<domain_name>:<port>/<alias>/rwservlet?key1
  ```

To add the URL syntax to a Web page as a hyperlink:

1. Set up an `HREF` tag on the host Web page, for example:

   ```
   <A HREF="http://<web_server>.<domain_name>:<port>/<alias>/rwservlet?key1>
   Employee Directory</A>
   ```

2. Point users to the Web page that hosts the link.
3. Users click the link to run the report.

8.5 Sending a Request to the URL Engine

If you have activated the Reports Server's URL engine, you can send job requests to the URL engine by using the following command line arguments:

- urlParameter identifies the URL to be placed in the cache. For example, http://www.oracle.com or a JSP report.
- jobType is the name of a job type (e.g., urlEngine) in the server configuration file that is associated with a URL engine.

> **Note:** For information on activating the URL engine, refer to Chapter 3, "Configuring Oracle9iAS Reports Services".

For example, a request that specifies an external URL for urlParameter might look like the following:

```
http://localhost.com/servlet/RWServlet?server=ReportsServer
+jobType=urlEngine+urlParameter="http://www.oracle.com"
+destype=mail+desname=foo@bar.com+desformat=htmlcss
```

Alternatively, a request that specifies a JSP report for urlParameter would look like the following:

```
http://<localhost>/servlet/RWServlet?server=ReportsServer+jobType=rwurl
+destype=cache+urlParameter="http%3A%2F%2F<localhost>%2Ffoo.jsp%3Fuserid
%3Dscott%2Ftiger@v815%3Fserver%3DreportsServer
```

> **Note:** If the URL has special characters, they must be encoded as per the x-www-form-urlencoded format.

8.6 Scheduling Reports to Run Automatically

You can use the server to run reports automatically from the Oracle9*i*AS Reports Queue Manager, Oracle9*i*AS Portal, or with the SCHEDULE command line argument. The scheduling feature enables you to specify a time and frequency for the report to run.

Refer to the Oracle9*i*AS Reports Queue Manager online help for more information about scheduling your reports.

If you publish a report as a Portal component on an Oracle9*i*AS Portal page, then you can schedule the report request to run automatically and push the resulting

reports to specified pages. Refer to Oracle9*i*AS Portal online help for more information.

The SCHEDULE keyword is available for use with the rwclient, rwservlet, and rwcgi commands. See Appendix A, Section A.4.83, "SCHEDULE" for more information.

8.7 Additional Parameters

When you send a request to the Reports Server, the following additional parameters, the values of which you cannot change, are implicitly passed along with your request:

Table 8–4 Additional parameters passed with a report request

Name	Description
ACCEPT_LANGUAGE	The comma separated list of languages accepted by the browser / user.
REMOTE_ADDR	The remote IP address from which the user is making the request.
REMOTE_HOST	The remote host name from which the user is making the request.
SCRIPT_NAME	The virtual path of the script being executed.
SERVER_NAME	The host name or IP address of the server on which the Reports Server is running.
SERVER_PORT	The port number of the server on which the Reports Server is running.
SERVER_PROTOCOL	The name and revision of the information protocol with which the request was sent.
USER_AGENT	The description of the remote client's browser.

8.8 Reusing Report Output from Cache

When you run a report, a copy of the report output is saved in the Oracle9*i*AS Reports Services cache. Subsequently, if an identical report is run (that is, with the same cache key), then the current request is recognized as a duplicate job.

There are several scenarios where Reports caching takes effect:

- When a new job request "A" comes to the Reports Server, and there is another job "B" that has the same cache key in the Current Jobs Queue (where it is

waiting for an available engine or is in the middle of execution), then job "A" will use the output from job "B".

The job cache key excludes the destype, desname, server, and tolerance parameters, and includes almost all other parameters.

This level of cache happens automatically. You don't need to specify any other parameters in the command line for it to work.

- If the user specifies `tolerance=`*n* (where *n* is a number in units of minutes) in the new job request "A", and it doesn't happen, then the Reports Server will try to find a job in the Finished Jobs Queue which was successfully completed within *n* minutes. If the Reports Server can find such a job, then the new job request "A" will return the output of job "B".

  Note: Refer to Appendix A, "Command Line Arguments" for more information about the TOLERANCE command line argument.

- In a clustered environment, duplicate job checking (i.e., jobs with the same cache key) is executed across cluster members. If a duplicate job is found in another server in the same cluster, the job request will be transferred to that server to retrieve the cached result.

  Oracle9*i*AS Reports Services cache results are persistent. If the Reports Server is shut down, once it is up again all the previous cache results are recovered and ready to use again.

8.8.1 Usage Notes

- You can set the cache size through the Oracle9*i*AS Reports Queue Manager or via the *cache* element in the server configuration file (<*server_name*>.conf).The Oracle9*i*AS Reports Server attempts to keep the total size of cache files below the set limit, deleting the least recently used files from the cache first. In addition, you can empty the cache through the Oracle9*i*AS Reports Queue Manager.

  For more information on setting the cache, refer to the Oracle9*i*AS Reports Queue Manager online help, and see Chapter 3, "Configuring Oracle9iAS Reports Services".

- If a report is being processed when an identical job is submitted, then Oracle9*i*AS Reports Services reuses the output of the currently running job even

if TOLERANCE is not specified or is equal to zero. For example, suppose that job_1 is currently being run and someone submits `job_2`, which has the same cache key as `job _1`. Oracle9*i*AS Reports Services uses the output from `job_1` for `job_2`. In this case, processing `job_2` is significantly faster since `job_2` is fetched from cache rather than run in its own right.

8.9 Using a Key Map File

If you choose to provide users with a URL or add a hyperlink to a Web site, then you can use a key map file to simplify or hide parameters in your URL requests.

The key map file contains command strings for running reports, each headed by a unique key identifier. Except when you run a report as a JSP, you reference only this key in the runtime URL. The server or servlet sends the key value to the map file (cgicmd.dat), which in turn returns the command associated with the specified key to the server or servlet for processing. By using key mapping, the command line arguments are all hidden from the user.

Key mapping is useful for:

- Shortening the URL, making it more convenient to use
- Remapping the runtime commands without having to change the original URL
- Standardizing several typical run configurations for your company
- Hiding certain parameters from users (for example, the database connect string)
- Restricting the parameters users can use to run a report

When you specify a key name from the key map file (cgicmd.dat), it must always be at the beginning of the query string (after the question mark) in a report request URL. An exception to this is if you use the cmdkey command line keyword, and express the key name as its value: `cmdkey=`*`keyname`*. In this case, you can place the key name anywhere in the query string within the report request URL. The `cmdkey` keyword can be used with jobs run as JSPs and with the `rwservlet` command.

Note: You'll find more information about the `cmdkey` keyword in Appendix A, "Command Line Arguments".

8.9.1 Enabling Key Mapping

Key mapping is enabled when any of these conditions are met:

- A valid file with the standard file name, cgicmd.dat, is present in the default location: the *ORACLE_HOME*\reports\conf\ directory on the Web server machine (on either Windows or UNIX).
- A valid key map file is entered in the Reports Servlet configuration file (rwservlet.properties) under the KEYMAPFILE parameter.
- When rwcgi is used, when the REPORTS_CGIMAP environment variable on the Web server machine specifies the name of a valid key map file. See Appendix B, "Reports-Related Environment Variables" for more information.

Usage Notes

- In rwcgi URLs, the first argument (that is the first information after the question mark) is treated as a key if it is not otherwise a part of a name/value pair. If the first argument is a name/value pair (i.e., keyword=value), then the whole command line is used in lieu of a cgicmd.dat key entry.

8.9.2 Adding Key Mapping Entries to a Key Map File

To add key mapping entries to a key map file:

1. Navigate to the cgicmd.dat file on the machine that hosts your Reports Server, and open it with a text editor.

   You'll find this file in the following directory on both Windows and UNIX:

   ORACLE_HOME\reports\server\conf\cgicmd.dat

2. Add a key mapping entry. For example:

   ```
   key1: report=<your_report.rdf> USERID=<username>/<password>@<my_db>
   DESFORMAT=html SERVER=<server_name>.<cluster_name (if present)>
   DESTYPE=cache
   ```

   In this example, *key1* is the name of the key.

   Except for the special parameters that are described in the file itself, the command line arguments follow the syntax rules of rwclient. See Appendix A, "Command Line Arguments" for more information.

3. Add or update the hyperlinks on your Web page.

   For more information, see Section 8.4, "Specifying a Report Request from a Web Browser".

8.9.3 Using a Key with Everything but JSPs

When you place a key name in a report request URL, it must always be the first value within the query string (immediately after the question mark). For example:

```
http://…/rwservlet?keyname
```

Below is an example of a key mapping for a restricted run with a parameter form.

The URL might be:

```
http://<web_server>.<domain_name>:<port>/CGI-BIN/rwcgi.exe?key&par1&par2&parN
```

The key mapping file might contain:

```
KEY: REPORT=<myreport> DEPTNO=%1 MYPARAM=%2 %*
```

This would generate the equivalent of the following command line request:

```
rwclient REPORT=<myreport> DEPTNO=par1 MYPARAM=par2 parN
```

8.9.4 Using a Key with a Report Run as a JSP

When you run a report as a JSP and want to call a command key in the cgicmd.dat file, you must use the `cmdkey` keyword in your URL. For example, your JSP URL might look like this:

```
http://…/myreport.jsp?cmdkey=key
```

Note: You can also use *cmdkey* with the `rwservlet` command.

When you use cmdkey with a JSP or `rwservlet`, you can place it anywhere within the query string. For example:

```
http://…/example.jsp?parameter1=value1&cmdkey=keyname
http://…/rwservlet?parameter1=value1&cmdkey=keyname
```

9

Creating Advanced Distributions

When you wish to define an advanced distribution for your report, you can design the distribution by developing a distribution XML file. This file can specify which section or sections of a report should go to what destination via what format of output. In one distribution XML file, you can specify many different destinations, including custom (pluggable) destinations you design.

This chapter provides information on creating a distribution XML file and some example use cases. It includes the following main sections:

- Distribution Overview
- Introduction to Distribution XML Files
- Elements of a Distribution XML File
- Distribution XML File Examples
- XSL Transformation for Custom / Pluggable Destinations

9.1 Distribution Overview

Although distribution XML files are not required for specifying the distribution of report output, they are useful for complex distributions. For example, there may be times when you want to publish the output of one report in a variety of ways. You might want to send an executive summary of a report to senior management while mailing detailed breakdowns to individual managers. In this case, you might produce a single report with two report sections: a portrait-sized summary section and a landscape-sized detail section. You would associate the detail section with a data model group that lists the managers, then alter the destination on each instance of the group to send each department's output to its related manager.

The distribution XML file tames distribution complexity by enabling you to define multiple outputs for a given report in one XML file, then call that file from a command line or URL.

9.2 Introduction to Distribution XML Files

9.2.1 The distribution.dtd File

When you create a distribution XML file, you follow the syntax defined in the `distribution.dtd` file located in the following directory (Windows and UNIX use the same path):

ORACLE_HOME`\reports\dtd`

As you look through the following sections, it may be useful to you to print the distribution.dtd file and refer to it as various elements and attributes are described.

Note: information provided in distribution XML file is case sensitive. The user must preserve case of various elements and attributes as specified in the distribution.dtd file.

The distribution.dtd file lists all elements that are valid within a distribution XML file. Each of these elements have attributes. Attributes that come with default values need not be specified, unless you wish to override the default.

You can create a dynamic distribution by introducing variable values into many different attributes. Variable values reference columns that are present in the report that is using the distribution XML file.

9.2.2 A Brief Word About Using Variables within Attributes

Use variables within attributes by entering &*column_name* or &<*column_name*> in the place of a static value.

Note: The ampersand (&) and less-than symbol (<) have specific meanings in XML, but they are also required symbols for certain Oracle9*i* Reports Developer command line arguments (for example, lexical parameters require the ampersand symbol). To avoid conflict with the XML meanings of these symbols when you set up variables, specify the encoded version of the ampersand (&) and less-than and greater-than symbols (< and >). For example:

Here is what the variable looks like *improperly* coded in an XML file:

```
<mail id="a1" to="&<manager>@mycompany.com" …
```

Here is what the variable looks like *properly* coded in an XML file:

```
<mail id="a1" to="&amp;&lt;manager&gt;@mycompany.com" …>
```

There is no special requirement for the greater-than symbol (>) used with variables, but for consistency, we recommend that you use the encoded version (>).

The variable syntax you use depends on whether the value is expressed by itself or in combination with other values or strings. For example, a value for a "to" attribute in a mail element might be expressed as either:

```
<mail id="a2" to="&amp;email" …>
```

OR

```
<mail id="a3" to="&amp;&lt;first_name&gt;.&amp;&lt;last_name&gt;@myco.com …>
```

In the first example (id="a2"), the variable's referenced column (`email`) contains a full e-mail address and does not require additional information. The second example (id="a3") uses a combination of variable values (`first_name` and `last_name`) and static text to construct an e-mail address (static text is the period after first_name and *@myco.com*). In both cases, you will get dynamic e-mail addressing. The example you use will depend on whether the variable contains all the information you need or requires additional information in order to be complete.

For even more complex layouts, you can also reference report columns you created with PL/SQL formulas. For example, in your report you may define the PL/SQL column:

```
PL/SQL formula CF_MAILID: return(:first_name||'.'||:last_name)
```

You'd reference this column in the distribution XML file as:

```
to="&amp;&lt;CF_MAILID&gt;@mycompany.com"
```

9.3 Elements of a Distribution XML File

The elements of a distribution XML file include:

- destinations
- foreach
- mail
- body
- include
- file
- printer
- destype
- attach
- property

Most of these elements have attributes that define the behavior of the element. The following sections describe the distribution XML file elements and their associated attributes. Section 9.4 provides use cases that demonstrate the distribution XML file elements and attributes in action.

9.3.1 destinations

Example

```
<destinations>
    [One or more distribution specifications]
</destinations>
```

Required/Optional

Required. You must have no more or less than one *destinations* element in your distribution XML file.

Description

The *destinations* element opens and closes the content area of the distribution XML file. In terms of the distribution XML file's tagging hierarchy, all the other elements are subordinate to the *destinations* element.

The *destinations* element has the following sub-elements:

- foreach
- mail
- file
- printer
- destype

Each of these is discussed in the following subsections.

9.3.2 foreach

Example

```
<foreach>
    <mail id="a1" to="my_addressee@mycompany.com" subject="Fourth Quarter
    Results">
        <attach format="pdf" name="dept_&amp;department_ID&gt;.pdf"
        srcType="report" instance="this">
            <include src="mainSection"/>
        </attach>
    </mail>
</foreach>
```

OR

```
<mail id="a4" to="recipient@mycompany.com" subject="Regional Results">
    <foreach>
        <attach format="pdf" name="report.pdf" srcType="report" instance="all">
            <include src="mainSection"/>
        </attach>
    </foreach>
</mail>
```

Required/Optional

Optional. You can have as many *foreach* elements as you require.

Description

Use the *foreach* element to burst your distribution against a repeating group. You can use *foreach* only when the associated report definition file (either RDF, JSP, or XML) has its "Repeat On" property for the section that will be burst set to an appropriate group. The *foreach* element specifies that the distribution defined between its open and close parameters should be performed for each repeating group.

The *foreach* element has the following sub-elements:

- mail
- file
- printer
- destype
- attach

Each of these is discussed in the following subsections.

You can also use the *foreach* element as a sub-element of the *mail* element, as depicted in the second example provided at the start of this section. (In this example, assuming that mainSection repeats on G_DEPARTMENT_ID, the example will produce a single attachment with all the instances of the report's mainSection in a single file.)

The *foreach* element works closely with the *instance* attribute of the *attach* and *file* elements. While *foreach* specifies that the distribution should be performed according to record groups, *instance* specifies whether the burst groups should be distributed in one file (`instance="all"`) or distributed as separate files: one file for each group instance (`instance="this"`).

When used with the *mail* element, *foreach* can mean different things according to its position relative to the *mail* element:

- When *foreach* precedes the *mail* element and `instance="this"`, each group instance is dispatched as a separate mail. For example:

```
<foreach>
    <mail id="a1" to="managers@mycompany.com" subject="results">
        <attach name="department_&amp;&lt;department_id&gt;.pdf"
        instance="this">
        <include src="mainSection" />
        </attach>
    </mail>
</foreach>
```

If the report is grouped according to department_id, and there are four departments, then there are four group instances. This means four e-mails per recipient, each e-mail with its own group instance attached: one e-mail has department 10's report attached; another e-mail has department 20's report attached; and so on. Each recipient receives all four e-mails.

- When *foreach* follows the mail element and `instance="this"`, each group instance is attached to one e-mail going to each recipient. For example:

```
<mail id="a1" to="managers@mycompany.com" subject="results">
    <foreach>
        <attach name="department_&amp;&lt;department_id&gt;.pdf"
        instance="this">
        <include src="mainSection" />
        </attach>
    </foreach>
</mail>
```

9.3.3 mail

Example

```
<mail id="a1" to="jsmith@foo.com" subject="Results">
    <body srcType="text">
        Attached are quarterly results.
    </body>
    <attach srcType="report">
        <include src="headerSection"/>
        <include src="mainSection"/>
    </attach>
</mail>
```

OR

```
<mail id="a4" to="recipient@mycompany.com" subject="Regional Results">
    <foreach>
        <attach format="pdf" name="report.pdf" srcType="report" instance="this">
            <include src="mainSection"/>
        </attach>
    </foreach>
</mail>
```

Required/Optional

Optional. You can have as many *mail* elements as you require.

Description

Use the *mail* element to specify distributions via an outgoing SMTP-based mail server. Use it to specify the recipients, the subject, and the priority of the e-mail.

The *mail* element has three sub-elements:

- body
- attach
- foreach

Between an open and close *mail* element, there can be only one *body* sub-element and anywhere from zero to multiple *attach* and *foreach* sub-elements.

The *mail* element also has a set of related attributes. These are expressed within the *mail* tag. For example, the *id*, *to*, and *subject* attributes are expressed:

```
<mail id="a1" to="jsmith@foo.com" subject="Recent Hires">
```

Table 9–1 lists and describes the attributes associated with the mail element.

Table 9–1 Attributes of the mail element

Attribute	Valid values	Description
id	string	Required. A keyword, unique within a given distribution XML file, that identifies a particular mail element. This can be a combination of a text string and one or more numbers, for example `id="a1"`. The `id` value must always start with an alpha character.
to	string	Required. Variable values allowed. The recipient(s) of the e-mail. Contains the full, formal e-mail address, for example: `to="jsmith@foo.com"`. Multiple recipients must be separated with commas. Can also contain variable values that reference columns used in the associated report. See Section 9.2.2 for more information.
cc	string	Optional. Variable values allowed. The recipient(s) to receive a copy of the e-mail.

Table 9–1 Attributes of the mail element

Attribute	Valid values	Description
bcc	string	Optional. Variable values allowed. The recipient(s) to receive a blind copy of the e-mail.
from	string	Optional. Variable values allowed. The sender of the e-mail.
replyTo	string	Optional. Variable values allowed. The e-mail account where replies should be sent.
subject	string	**Default:** Mail Sent from &Report Optional. Variable values allowed. The subject of the e-mail. In the absence of a specified subject, the subject line will read: Mail Sent from [*Name of Report*]
priority	highest | high | normal | low | lowest	**Default:** normal The e-mail's delivery priority.
returnReceipt	true | false	**Default:** false Indication of whether the replyto individual or account should be notified when the e-mail is received.

organization	string	Optional. Variable values allowed. The name of the organization distributing the e-mail, for example:
		`organization="Region 10 Sales"`
		Or
		`organization="&amp;department_name"`

Note: For the mail element to work properly, the Reports Server must know which outgoing SMTP mail server to send mail to. You specify this information in the Reports Server configuration file (<*server_name*>.conf). This file has a *pluginParam* element where you can enter the name of a mail server. For example:

```
<pluginParam name=mailServer>smtp01.mycorp.com</pluginParam>
```

For more information, see Chapter 3, "Configuring Oracle9iAS Reports Services".

9.3.4 body

Example

On Windows
```
<mail id="a1" to="jsmith@foo.com" subject="Results">
    <body srcType="file">
        <include src="c:\mail\body.html"/>
    </body>
</mail>
```

On UNIX
```
<mail id="a1" to="jsmith@foo.com" subject="Results">
    <body srcType="file">
        <include src="/mail/body.html"/>
    </body>
</mail>
```

Required/Optional

Optional. You can have a maximum of one body element associated with a given mail element.

Description

The *body* element acts as a sub-element to the *mail* element. It specifies the content (or *body*) of the e-mail. With *body*, you can type a text string between the open and close *body* tag or use an *include* sub-element to specify either an external file, a report, or a section of a report. For example:

```
<mail id="a1" to="jsmith@foo.com" subject="Results">
    <body srcType="text">
        Attached are quarterly results.
    </body>
    …
```

Or

```
<mail id="a1" to="jsmith@foo.com" subject="Results">
    <body srcType="file">
        <include src="d:\reports\admin\results.html"/>
    </body>
    …
```

Or

```
<mail id="a1" to="&amp;&lt;first_name&gt;.&amp;&lt;last_name&gt;@myco.com"
```

```
subject="Quarterly Results">
    <body srcType="report" format="html">
        <include src="headerSection"/>
    </body>
    …
```

Body has three attributes: *srcType*, *format*, and *instance*. They are described in Table 9–2.

Table 9–2 Attributes of the body sub-element of mail

Attribute	Valid values	Description
srcType	file | report | text	**Default:** report The source for content of an e-mail. The content is displayed in the body of the e-mail. In the absence of a specified *srcType*, the default is used.
format	html | htmlcss | ascii	**Default:** html Required when *srcType* is *report* with a *format* other than *html*, the default; otherwise *format* is optional. The format of the content.
instance	this | all	**Default:** all Used when the *foreach* element is also present. With a grouped report that is burst into separate reports, *instance* specifies whether the groups will be broken into separate content according to each group instance (this) or all contained within the same content (all).

9.3.5 attach

Example

```
<mail id="a1" to="jsmith@foo.com" subject="Results">
    <body srcType="text">
        Attached are quarterly results.
    </body>
    <foreach>
        <attach format="html" name="contacts.htm" srcType="report"
        instance="all">
            <include src="headerSection"/>
            <include src="mainSection"/>
        </attach>
    </foreach>
```

```
</mail>
```

Required/Optional

Optional. You can have as many *attach* elements as you require with a *mail* element. Note that *attach* is also a sub-element of *foreach*, and *foreach* requires that at least one of its sub-elements be used (out of *mail*, *file*, *printer*, *destype*, and *attach*).

Description

Attach specifies attachments to the e-mail. Additionally, *attach* must have at least one *include* sub-element, and can have more than one if srcType="report". *Attach* attributes are listed and described in Table 9–3.

Table 9–3 Attributes of the attach sub-element of mail

Attribute	Valid values	Description
format	pdf | html | htmlcss | rtf | ascii | xml | dflt	**Default:** pdf Required when *srcType* is report and the report format is other than *pdf*, the default; otherwise *format* is optional. The format of the attached material, for example `format="htmlcss"`.
name	string	Optional. Variable values allowed. The filename of the attached material. Can also contain variable values that reference columns used in the associated report. See Section 9.2.2 for more information.
srcType	file | report | text	**Default:** report The source of the attachment, either a file, a report, or text.
instance	this | all	**Default:** all Used when the *foreach* element is also present. With a grouped report that is burst into separate reports, *instance* specifies whether the groups will be broken into separate content according to each group instance (this) or all contained within the same content (all).

Using these attributes in conjunction with the *foreach* element, you can design a destination that will repeat on a group instance and generate an e-mail for each group attachment. For example:

```
<foreach>
```

```
    <mail id="a2" to="first.name@myco.com,second.name@myco.com,
    third.name@myco.com, fourth.name@myco.com" subject="Department Summaries">
        <body srcType="text">
            Attached is the breakdown of department summaries for the last
            quarter.
        </body>
        <attach format="htmlcss" name="deptsum.html" srcType="report"
        instance="this">
            <include src="report"/>
        </attach>
    </mail>
</foreach>
```

By moving the location of the *foreach* element, you can generate one e-mail with multiple attachments: a separate one for each group instance.

```
<mail id="a2" to="first.name@myco.com,second.name@myco.com, third.name@myco.com,
fourth.name@myco.com" subject="Department Summaries">
    <body srcType="text">
            Attached is the breakdown of department summaries for the last
            quarter.
    </body>
    <foreach>
        <attach format="htmlcss" name="deptsum.html" srcType="report"
        instance="this">
            <include src="report"/>
        </attach>
    </foreach>
</mail>
```

9.3.6 include

Example

```
<mail id="a1" to="jsmith@foo.com" subject="Q4">
    <body srcType="text">
        Attached are quarterly results.
    </body>
    <attach srcType="report" format="pdf">
        <include src="report"/>
    </attach>
</mail>
```

Or

```
<mail id="a1" to="jsmith@foo.com" subject="Q4">
    <body srcType="text">
        Attached are quarterly results.
    </body>
    <attach srcType="report" format="htmlcss">
        <include src="headerSection"/>
    </attach>
</mail>
```

Or

```
<mail id="a1" to="jsmith@foo.com" subject="Q4">
    <body srcType="text">
        Attached are quarterly results.
    </body>
    <attach srcType="file">
        <include src="c:\management\reports\current\Q4.htm"/>
    </attach>
</mail>
```

Required/Optional

Required when used with *body* and *attach* when *srcType* is *report* or *file*, but not when *srcType* is *text*. Also required for *file*, *printer*, and *destype*. In the instances where it is required, you must have one and can have more than one *includes*.

Description

The *include* element is available for use with the *body*, *attach*, *file*, *printer*, and *destype* elements. It specifies the file, report, or report section to be included in the body of an e-mail, as an attachment to an e-mail, in the content of a file, in the printer output, or in the content of a custom destination type.

If you want to specify more than one section, but not the entire report, enter an *include* for each required section. For example:

```
<mail id="a1" to="jsmith@foo.com" subject="Results">
    <body srcType="text">
        Attached are quarterly results.
    </body>
    <attach srcType="report" format="htmlcss">
        <include src="headerSection"/>
        <include src="mainSection"/>
    </attach>
</mail>
```

If the preceding *body* or *attach* element has *srcType* of *file*, the subsequent *include* can specify the file either with a directory path and filename or with just the filename, provided the file is located in a directory listed in the REPORTS_PATH environment variable. For example:

```
<mail id="a1" to="jsmith@foo.com">
    <body srcType="file">
        <include src="q4sales.pdf"/>
    </body>
</mail>
```

If you do specify a path, use the appropriate standard for your platform. For example:

On Windows: `<include src="c:\management\reports\current\Q4.htm"/>`

On UNIX: `<include src="/management/reports/current/Q4.htm"/>`

No other XML elements are placed between an *include* element's open and close tags; however, *include* does have one attribute: *src*, described in Table 9–4.

Table 9–4 Attributes of the include sub-element when used with mail's body or attach

Attribute	Valid values	Description
src	*(path and) filename* report headerSection mainSection trailerSection	Required. The source of material specified in the preceding *attach*, *body*, *file*, *printer*, or *destype* element. Because the distribution XML file is called when you run a specific report, there is no need to specify the report's name or location in the *src* attribute when `src="report"`. When the preceding *body* or *attach* element specifies a *file* srcType, provide the directory path and filename or just a filename, provided the file is located in a directory listed in the REPORTS_PATH environment variable. When the preceding *body* or *attach* element specifies a *report* srcType, specify the entire report (report) or provide the section(s) of the report to be included in the body or to be attached (e.g., headerSection, mainSection, and / or trailerSection).

9.3.7 file

Example

On Windows
```
<file name="c:\management\reports\report.pdf" format="pdf">
    <include src="report"/>
</file>
```

On UNIX
```
<file name="/management/reports/report.pdf" format="pdf">
    <include src="report"/>
</file>
```

Or

```
<foreach>
    <file name="section&amp;&lt;department_id&gt;.pdf" format="pdf"
    instance="this">
        <include src="mainSection"/>
    </file>
</foreach>
```

Required/Optional

Optional. You can have as many *file* elements as you require.

Description

Use the *file* element to specify distributions to a file. *File* elements have one sub-element: *include*. There must be at least one *include* sub-element and there may be more between an open and close *file* element.

When used with the *foreach* element and the `instance="this"` attribute, the *file* element can distribute each group instance of a grouped report to separate files. For example, if you group a report on department_id, and there are four departments, you can use the *foreach* / *file* / instance="this" combination to generate four files, each with a separate department's report. In this case, the *file* entry in the distribution XML file might look like this:

```
<foreach>
    <file id="a3" name="dept_&amp;&lt;department_id&gt;.pdf" format="pdf"
    instance="this">
        <include="report"/>
    </file>
</foreach>
```

In this example, all report sections (header, main, and trailer) must repeat on the same group instance (e.g., department_id).

File elements also have a set of related attributes. These are expressed within the file tag. For example, the "id" and "name" *file* attributes are expressed:

Windows: `<file id="a7" name="d:\reports\2001\q4sales.pdf">`

UNIX: `<file id="a7" name="/reports/2001/q4sales.pdf">`

Table 9–5 lists and describes the attributes associated with a *file* element.

Table 9–5 Attributes of the file element

Attribute	Valid values	Description
id	string	Required. A keyword, unique within a given distribution XML file, that identifies a particular file element. This can be a combination of a text string and one or more numbers, for example `id="a1"`. The `id` value must always start with an alpha character.
name	string	Required. Variable values allowed. The location and filename of the destination file. Enter a directory path. Include the filename. For example: **Windows:** name="d:\reports\file.pdf" **UNIX:** name="reports/file.pdf Can also contain variable values that reference columns used in the associated report. See Section 9.2.2 for more information.
format	pdf | html | htmlcss | rtf | ascii | xml | bitmap	**Default:** pdf The destination file format, for example: `format="htmlcss"`
instance	this | all	**Default:** all Used when the *foreach* element is also present. With a grouped report that is burst into separate reports, *instance* specifies whether the groups will be broken into separate files according to each group instance (this) or all contained within the same file (all).

9.3.8 printer

Example

On Windows

```
<printer id="a1" name="\\server_name\printer_name" copies="5">
    <include src="report"/>
</printer>
```

On UNIX

```
<printer id="a1" name="alias_to_registered_printer" copies="5" instance="all">
    <include src="report"/>
</printer>
```

Required/Optional

Optional. You can have as many *printer* elements as you require.

Description

Use the *printer* element to specify distributions to a printer. *Printer* elements have one sub-element: *include*. There must be at least one *include* sub-element and there may be more between an open and close *printer* element.

When used with the *foreach* element and the instance="this" attribute, the *printer* element can distribute each group instance of a grouped report to a separate print job. For example, if you group a report on department_id, and there are four departments, you can use the *foreach* / *printer* / instance="this" combination to generate four printed reports, each containing a separate department's report. In this case, the *printer* entry in the distribution XML file might look like this:

```
<foreach>
    <printer id="a7" name="\\server_name\printer_name" instance="this">
        <include="report"/>
    </printer>
</foreach>
```

In this example, all report sections (header, main, and trailer) must repeat on the same group instance (e.g., department_id).

Table 9–6 lists and describes the attributes associated with a *printer* element.

Table 9–6 Attributes of the printer element

Attribute	Valid values	Description
id	string	Required. A keyword, unique within a given distribution XML file, that identifies a particular file element. This can be a combination of a text string and one or more numbers, for example `id="a1"`. The `id` value must always start with an alpha character.

name	string	Required. Variable values allowed. The destination printer. How you enter this information differs between Windows and UNIX.
		For Windows, specify the printer server name and the printer name. For example:
		`name="\\server_name\printer_name"`
		For UNIX, specify the alias assigned to a registered printer. For example:
		`name="sales_printer"`
		Can also contain variable values that reference columns used in the associated report. See Section 9.2.2 for more information.

copies	string	**Default**: 1 Number of copies of each report or each report group instance to print.
instance	this | all	**Default**: all Used when the *foreach* element is also present. With a grouped report that is burst into separate reports, *instance* specifies whether the groups will be broken into separate printed reports according to each group instance (this) or all contained within the same printed report (all).

9.3.9 destype

Example

```
<destype id="acustom1" name="fax">
    <include src="headerSection"/>
    <property name="number" value="914925551212"/>
</destype>
```

Required/Optional

Optional. You can have as many *destype* elements as you require.

Description

Use the *destype* element to specify distribution to a custom destination, such as to a fax machine or an FTP site. You also use *destype* to specify distribution to a portal created with Oracle9*i*AS Portal. The *destype* element allows for the use of two sub-elements: *property* and *include*. At least one include is required.

The inclusion of a custom destination type requires that you have a defined distribution handler in place to usher report content to the custom output destination.

Note: Build a custom destination type via the Oracle9*i*AS Reports Services Destinations API. Look for upcoming information about Reports APIs on the Oracle Technology Network: http://otn.oracle.com.

When used with the *foreach* element and the instance="this" attribute, the *destype* element can distribute each group instance of a grouped report to a separate destype instance (e.g., a separate fax). For example, if you group a report on department_id, and there are four departments, you can use the *foreach*/*printer*/instance="this" combination to generate four destype instances, each containing a separate department's report. In this case, the *destype* entry in the distribution XML file might look like this:

```
<foreach>
    <destype id="a9" name="fax" instance="this">
        <include="report"/>
        <property name="number" value="&amp;&lt;fax_number&gt;"/>
    </destype>
</foreach>
```

In this example, all report sections (header, main, and trailer) must repeat on the same group instance (e.g., department_id).

Custom destination types also have a set of related attributes. These are expressed within the *destype* tag. For example, the "id", "name", and "instance" *destype* attributes are expressed:

```
<foreach>
    <destype id="a1" name="name_of_destination_type" instance="all">
```

```
</foreach>
```

Table 9–7 lists and describes the attributes associated with a *destype* element.

Table 9–7 Attributes of the destype element

Attribute	Valid values	Description
id	string	Required. A keyword, unique within a given distribution XML file, that identifies a particular file element. This can be a combination of a text string and one or more numbers, for example `id="a1"`. The `id` value must always start with an alpha character.

name	string	Required. The name of the custom destination. For example, for a fax, this might be:
		`name="fax"`
		For a portal built with Oracle9*i*AS Portal:
		`name="oraclePortal"`

instance	this | all	**Default:** all Used when the *foreach* element is also present. With a grouped report that is burst into separate reports, *instance* specifies whether the groups will be broken into separate destype instances according to each group instance (this) or all contained within the same destype instance (all). For example, if you custom destination type is a fax, `instance="this"` would mean a separate fax for each group instance, and `instance="all"` would mean one fax for all groups.

Oracle9*i*AS Reports Services supports the creation and use of custom destination types (pluggable destinations) in the Reports Services environment. One way it does this is by allowing you to include calls to custom destinations in your distribution XML file. The distribution XML file provides a way to define custom destinations through property name / value pairs used in conjunction with the *destype* element.

9.3.10 property

Example

```
<foreach>
    <destype id="custom1" name="fax" instance="all">
        <include src="headerSection"/>
        <property name="number" value="914925551212"/>
    </destype>
</foreach>
```

Required/Optional

Optional. You can have as many properties as you require under a *destype* element.

Description

The *property* element allows for the inclusion of name/value pairs expressed in terms recognized by a custom destination type (*destype*). Properties are merely passed along to the destination handler. They serve no function within Reports Services. How you specify properties is entirely dependent on the requirements of your custom destination.

9.4 Distribution XML File Examples

This section provides examples, from simple to complex, of distribution XML elements. They are organized according to the main distribution.dtd elements:

- foreach examples
- mail examples
- file examples
- printer examples

9.4.1 foreach examples

The examples in this section include:

- Single E-Mail with Report Groups as Separate Attachments
- Separate E-Mail for Each Group Instance
- Separate E-Mails with Separate Sections as Attachments
- Separate File for Each Section

- Separate Print Run for Each Report

9.4.1.1 Single E-Mail with Report Groups as Separate Attachments

In this example, each attachment contains the corresponding instance from the header, main, and trailer sections. That is, if the report is grouped on department_id, and the first department is department 10, the first attachment will be a report with header, main, and trailer sections all containing department 10 information. This example is valid only if the header, main, and trailer sections repeat on the same group instance, in this case department_id.

```
<mail id="a1" to="managers@mycompany.com" subject="New Hires">
    <foreach>
        <attach format="html" srcType="report" instance="this">
            <include src="report"/>
        </attach>
    </foreach>
</mail>
```

First of all, assume in this example that "managers@mycompany.com" goes to a mailing list that distributes to each department manager. If there are four departments: 10, 20, 30, and 40, the first attachment will contain header, main, and trailer sections corresponding to department 10; the second to 20; and so on. This example will yield one e-mail per recipient, each with four attachments.

9.4.1.2 Separate E-Mail for Each Group Instance

In this example, each recipient will receive a separate e-mail for each grouped report. For example, if the report is grouped on department_id, and there are four departments, one recipient will receive four e-mails, each with a separate department's report attached.

```
<foreach>
    <mail id="weeklies" to="managers@mycompany.com">
        <attach format="htmlcss" srcType="report" instance="this">
            <include src="mainSection"/>
        </attach>
    </mail>
</foreach>
```

9.4.1.3 Separate E-Mails with Separate Sections as Attachments

In this example, different sections repeat on different groups. The distribution is set up so that each recipient will receive a separate e-mail with attachment for each grouped main section and for each grouped trailer section.

```
<foreach>
    <mail id="a6" to="managers@mycompany.com" subject="Personnel Reports">
        <attach format="pdf" name="attach.pdf" srcType="report" instance="this">
            <include src="mainSection"/>
        </attach>
        <attach format="rtf" name="attach.rtf" srcType="report" instance="this">
            <include src="trailerSection"/>
        </attach>
    </mail>
</foreach>
```

9.4.1.4 Separate File for Each Section

In this example, a separate file is generated for each group instance. Groups repeat on department_id. Each file is named with the relevant department ID.

```
<foreach>
    <file id="a10" name="department_&amp;&lt;department_id&gt;.pdf"
    instance="this">
        <include src="mainSection"/>
    </file>
</foreach>
```

Assuming that there are four departments, 10 through 40, this example will result in the creation of four files, named in turn department_10.pdf, department_20.pdf, and so on.

The *format* attribute is not included in the *file* element because it is not required when the *srcType* is *file* or *text*. It is required when the *srcType* is *report*.

Note: If you do not specify unique filenames through the use of variable values (see Section 9.2.2), in this example, each successively created file will overwrite the previously created file. That is, the department.pdf file for department 20 will overwrite the department.pdf file for department 10, and so on, until there is only one file left, department.pdf, with information from the last department report created (e.g., department 40).

9.4.1.5 Separate Print Run for Each Report

The way you specify a printer name differs between Windows and UNIX. The first example is for Windows. The second is for UNIX.

9.4.1.5.1 Windows In this example, assuming that the report is grouped on department_id, a report will be printed for each department.

```
<foreach>
    <printer id="a7" name="\\server_name\printer_name" instance="this">
        <include src="report"/>
    </printer>
</foreach>
```

9.4.1.5.2 UNIX In this example, assuming that the report is grouped on department_id, a report will be printed for each department.

```
<foreach>
    <printer id="a7" name="printer_alias" instance="this">
        <include src="report"/>
    </printer>
</foreach>
```

9.4.2 mail examples

The examples in this section include:

- E-Mail with a Whole Report as the Body
- E-Mail with a Section of a Report as the Body
- E-Mail with Two Report Sections as the Body
- E-Mail with External File as Body and Report as Attachment
- E-Mail with Whole Report and Grouped Sections Attached
- E-Mail to Relevant Manager and Department

9.4.2.1 E-Mail with a Whole Report as the Body

The report will comprise the content of this e-mail. That is, when recipients open this e-mail, they will see the report.

```
<mail id="a5" to="managers@mycompany.com" subject="Quarterly Report">
    <body srcType="report" format="html">
        <include src="report"/>
    </body>
</mail>
```

9.4.2.2 E-Mail with a Section of a Report as the Body

A section of a report will comprise the content of this e-mail. That is, when recipients open this e-mail, they will see a section of the report.

```
<mail id="a6" to="employees@mycompany.com">
    <body srcType="report" format="html">
        <include src="mainSection"/>
    </body>
</mail>
```

The *subject* attribute is not included in this *mail* element, so the default subject will be used: `Mail Sent From &Report`. At runtime, the variable `&Report` will be replaced with the name of the report.

9.4.2.3 E-Mail with Two Report Sections as the Body

Two sections of a report will comprise the body of this e-mail. That is, when recipients open this e-mail, they'll see two sections, headerSection and mainSection, joined together in one report.

```
<mail id="emp_addresses" to="employees@mycompany.com" subject="Employee Address
List">
    <body srcType="report" format="html">
        <include src="headerSection"/>
        <include src="mainSection"/>
    </body>
</mail>
```

9.4.2.4 E-Mail with External File as Body and Report as Attachment

The contents of the body for this email will be an external file, and the report will go along as an attachment. The path to the file is expressed differently for Windows and UNIX.

9.4.2.4.1 Windows

```
<mail id="XQRSN" to="accounting@mycompany.com" subject="Salaries"
    <body srcType="file">
        <include src="c:\mail\body.html"/>
    </body>
    <attach format="pdf" name="salaries.pdf" srcType="report">
        <include src="report"/>
    </attach>
</mail>
```

9.4.2.4.2 UNIX

```
<mail id="XQRSN" to="accounting@mycompany.com" subject="Salaries"
    <body srcType="file">
        <include src="/mail/body.html"/>
    </body>
    <attach format="pdf" name="salaries.pdf" srcType="report">
        <include src="report"/>
    </attach>
</mail>
```

9.4.2.5 E-Mail with Whole Report and Grouped Sections Attached

In this example, recipients receive one e-mail with multiple attachments: one attachment for each group instance and an additional attachment that contains the entire report. If the report is grouped on department_id and there are four departments, recipients will receive five attachments: one for each department and one whole report.

```
<mail id="grx90" to="sales@mycompany.com">
    <body srcType="text">
    Attached you will find the summary report and breakdown by department of
    weekly totals.
    </body>
    <attach format="rtf" name="myAttach.rtf" srcType="report">
        <include src="report"/>
    </attach>
    <foreach>
        <attach format="pdf" name="myattach.pdf" srcType="report"
        instance="this">
            <include src="mainSection"/>
        </attach>
    </foreach>
</mail>
```

9.4.2.6 E-Mail to Relevant Manager and Department

In this example, the manager for department 10 gets department 10's report; the manager for department 20 gets department 20's report; and so on. For this tag set to be valid, the variable must refer to a column that is included in the "repeat on" group used with the attached section. That is, if the section repeats on G_ department_id, *manager* must be a column in that group.

```
<foreach>
    <mail id="mgr1090" to="&amp;&lt;manager&gt;@mycompany.com">
        <attach format="pdf" name="attach.pdf" srcType="report" instance="this">
```

```
            <include src="mainSection"/>
        </attach>
    </mail>
</foreach>
```

9.4.3 file examples

Whenever you burst and distribute grouped reports to files, be sure to specify filenames with variable values based on the repeating group or some other variable information. Otherwise, you run the risk of having each successive file that is created overwrite the previously created file. For example, if you specify an output filename of department.pdf, and you output separate instances of each department's report, the second department.pdf file will overwrite the first department.pdf file; the third will overwrite the second; an so on. You will end up with only one report, that of the final department to be output. Instead, with grouped reports that you want to output separately according to each group instance, use variable values to specify filenames, for example: `name="department_&<department_id>.pdf"`.

The examples in this section include:

- File for Whole Report
- File for Combined Report Sections
- File for Each Group of Combined Sections
- File for Each Report Group Instance

9.4.3.1 File for Whole Report

This example will yield one file named report.pdf that contains the entire report.

9.4.3.1.1 Windows

```
<file id="a1" name="c:\reports\report.pdf" format="pdf">
    <include src="report"/>
</file>
```

9.4.3.1.2 UNIX

```
<file id="a1" name="/reports/report.pdf" format="pdf">
    <include src="report"/>
</file>
```

9.4.3.2 File for Combined Report Sections

This example will yield one file named sections.pdf that contains a report consisting of the header section and the main section of the report.

```
<file id="a2" name="sections.pdf" format="pdf">
    <include="headerSection"/>
    <include="mainSection"/>
</file>
```

9.4.3.3 File for Each Group of Combined Sections

In this example, a separate file will be created for each repeating group. Each file will contain a report that combines the relevant group main and trailer sections. The main and trailer sections must repeat on the same group, and the variable file name must refer to a column contained within the "repeat on" group. That is, if the report repeats on department_id, and you have four departments, 10 through 40, then one file will contain the main and trailer sections of department 10; the next will contain the main and trailer sections of department 20; and so on. The variable value under *name* must refer to a column that is within the G_department_id group.

```
<foreach>
    <file id="file9" name="department_&amp;&lt;department_id&gt;.pdf"
    instance="this">
        <include src="mainSection"/>
        <include src="trailerSection"/>
    </file>
</foreach>
```

9.4.3.4 File for Each Report Group Instance

In this example, assuming the report is grouped on department_id and there are four departments, 10 through 40, you will end up with four files respectively named: department_10.pdf, department_20.pdf, department_30.pdf, and department_40.pdf.

```
<foreach>
    <file id="a20" name="department_&amp;&lt;department_id&gt;.pdf"
    instance="this">
        <include src="report"/>
    </file>
</foreach>
```

9.4.4 printer examples

The examples in this section include:

- Print Whole Report
- Print Two Sections of a Report
- Print Grouped Report
- Print Combined Sections for Each Group Instance
- Print Relevant Instance of a Report to Its Relevant Printer

The way printer names are specified, differs between Windows and UNIX. Each example demonstrates both ways.

9.4.4.1 Print Whole Report

In this example, the entire report will be sent to the specified printer.

9.4.4.1.1 Windows

```
<printer id="a80" name="\\neptune\prtr20">
    <include src="report"/>
</printer>
```

9.4.4.1.2 UNIX

```
<printer id="a80" name="10th_floor_printer">
    <include src="report"/>
</printer>
```

9.4.4.2 Print Two Sections of a Report

In this example, two sections of a report will be sent to the printer.

9.4.4.2.1 Windows

```
<printer id="a1" name="\\neptune\prtr20">
    <include src="headerSection"/>
    <include src="mainSection"/>
</printer>
```

9.4.4.2.2 UNIX

```
<printer id="a1" name="10th_floor_printer">
    <include src="headerSection"/>
    <include src="mainSection"/>
</printer>
```

9.4.4.3 Print Grouped Report

In this example, one report will be printed. The report will be grouped by, for example, department_id. For this to work, all sections of the report must repeat on the same group.

9.4.4.3.1 Windows

```
<foreach>
    <printer id="prt20" name="\\neptune\prtr20" instance="all">
        <include src="report"/>
    </printer>
</foreach>
```

9.4.4.3.2 UNIX

```
<foreach>
    <printer id="prt20" name="10th_floor_printer" instance="all">
        <include src="report"/>
    </printer>
</foreach>
```

9.4.4.4 Print Combined Sections for Each Group Instance

This example will yield a number of print jobs: one for each group instance. The combined sections must repeat on the same group. If the report repeats on department_id, and you have four departments, 10 through 40, you will end up with four print jobs: one for department 10; one for department 20; and so on. The main and trailer sections must both repeat on department_id.

9.4.4.4.1 Windows

```
<foreach>
    <printer id="prt20" name="\\neptune\prtr20" instance="this">
        <include src="mainSection"/>
        <include src="trailerSection"/>
    </printer>
</foreach>
```

9.4.4.4.2 UNIX

```
<foreach>
    <printer id="prt20" name="10th_floor_printer" instance="this">
        <include src="mainSection"/>
        <include src="trailerSection"/>
    </printer>
```

```
</foreach>
```

9.4.4.5 Print Relevant Instance of a Report to Its Relevant Printer

For this example to work, the "repeat on" group must contain a column of printer names appropriate to the host platform (e.g., the printer_name column must contain an appropriate printer alias on UNIX and a printer server / name combination on Windows). For example, if the report is grouped by department_id, then G_department_id must also have a printer_name column. Assuming the printer_names are tied to departments, then department 10's report would be printed on department 10's printer; department 20's report would be printed on department 20's printer; and so on.

```
<foreach>
    <printer id="a60" name="&amp;printer_name" instance="this">
        <include src="mainSection"/>
    </printer>
</foreach>
```

Each group instance equals a separate print job. Each print job goes to the relevant department's printer

9.5 Using a Distribution XML File at Runtime

The method for using a distribution XML file at runtime is essentially the same whether you use it in a URL or a command line. Use the commands:

```
distribute=yes destination=filename.xml
```

Where *filename* is the name of the distribution XML file. You are required to specify either the relative or absolute path of the XML file. For example, for Windows, you might specify:

```
distribute=yes destination=c:\ORACLE_HOME\reports\distribution\filename.xml
```

For UNIX, you might specify:

```
distribute=yes destinations=ORACLE_HOME/reports/distribution/filename.xml
```

The paths in these examples are used for illustrative purposes only. There is no requirement for where you store your distribution XML files. You can store them wherever you like.

Note: For detailed information on running reports from command lines and URLs and using the cgicmd.dat file, see Chapter 8, "Running Report Requests".

9.6 XSL Transformation for Custom/Pluggable Destinations

The distribution.xsl file is an XML style sheet, located on both Windows and UNIX at *ORACLE_HOME*\reports\conf\distribution.xsl. You can modify this file by adding a template for translating your destype tag format to the required format defined in the distribution.dtd file.

Use the distribution.xsl file to transform user-defined custom tags in the distribution XML file to a format required by Reports runtime. Reports can understand only the generic *destype* tag structure for any pluggable destination. The user can specify the custom destination in accordance with the generic *destype* tag structure for a pluggable destination. Alternatively, for ease of use, the user can specify a custom, more specific tag structure. These tags are unknown to the distribution.dtd, so they need to be mapped to the generic destype tag structure as specified in the distribution.dtd.

The following examples illustrate a fax destination. The user can specify the destination as per the generic tag structure in the distribution XML file as follows:

```
<destype id="faxdest" name="fax">
    <property name="number" value="123456789"/>
    <include src="report"/>
</destype>
```

A more user-friendly example:

```
<fax id="faxdest" number="123456789">
    <include src="report"/>
</fax>
```

Reports runtime cannot process the <fax> tag structure as illustrated here because the <fax> tag is not a standard destination specified in the distribution.dtd file. The following tag structure must therefore be converted to the generic format as shown in the first example.

To achieve this, you must specify a template for the fax destination in the distribution.xsl file. The template will be used to convert the <fax> tag

structure to the generic *destype* structure. Your `distribution.xsl` entry might look like this:

```
<xsl:output doctype-system="distribution.dtd"/>

<xsl:template match = "/">
    <xsl:apply-templates match = "destinations" />
</xsl:template>

<xsl:template match="destinations">
    <destinations>

    <!--
    The Standard mail/file/printer/destype and foreach must be copied to the
    transformed xml. The foreach tag must be copied only if it is specified with
    file/mail/printer/destype tags.
    -->
        <xsl:copy-of select="mail"/>
        <xsl:copy-of select="file"/>
        <xsl:copy-of select="printer"/>
        <xsl:copy-of select="destype"/>
        <xsl:copy-of select="foreach"/>

        <!-- apply template for the sample FAX destination -->
        <xsl:apply-templates match = "fax" />

    </destinations>
</xsl:template>

<!--
    Sample Transformation Template for a FAX destination specified in the
    distribution.xml file

    <fax id="FAXDEST" number="123456789">
        <include src="report"/>
    </fax>
-->

<xsl:template match="fax">
    <!-- create a new destype element -->
    <xsl:element name="destype">
        <!--
        create an ID attribute and copy the value from the ID given for the fax
        destination
        -->
```

```
    <xsl:attribute name="id">
        <xsl:value-of select="@id"/>
    </xsl:attribute>

    <!-- create a Name attribute with a fax as it's value -->
    <xsl:attribute name="name">fax</xsl:attribute>
    <!--
    create a Property Attribute with name / value attribute pairs property tag
    is created for number attribute. Similarly create more property tags for any
    other attribute you add to the FAX destination
    -->
    <xsl:element name="property">
        <xsl:attribute name="name">number</xsl:attribute>
        <xsl:attribute name="value">
            <xsl:value-of select="@number"/>
        </xsl:attribute>
    </xsl:element>

        <!-- copy the include tag as it is -->
        <xsl:copy-of select="include"/>
    </xsl:element>
    <!-- end of template -->
</xsl:template>

</xsl:stylesheet>
```

Note: All you need to do after you modify the XSL file is save it back to the same location under the same file name. Reports will automatically look for this XSL file when resolving distributions.

10

Customizing Reports with XML

XML customizations enable you to modify reports at runtime without changing the original report. With the addition of the `CUSTOMIZE` command to your runtime command line, you can call a customization file to add to or change a report's layout or data model. One XML customization file can perform all of these tasks or any combination of them. You can even use Reports XML to build a report data model for inclusion in a custom JSP-based report.

By creating and applying different XML customizations, you can alter the report output on a per user or per user group basis. You can use the same report to generate different output depending upon the audience.

When you apply an XML customization to a report, you have the option of saving the combined definition to a file. As a result, you can use XML customizations to make batch updates to existing reports. You can quickly update a large number of reports without having to open each file in the Reports Builder.

Oracle9*i*AS Reports Services extends the possible types of Reports XML customizations by enabling you to create an entire Reports data model in XML. This includes the creation of multiple data sources, linking between data sources, and group hierarchies within each data source. Data model support via Reports XML customization means that any data model that can be created with the Reports Builder can now be created by specifying XML. Additionally, all properties that can be set against data model objects can now be set using XML.

This chapter discusses the ways you can use XML to customize reports on the fly and to build data models. It includes the following sections:

- Customization Overview
- Creating XML Customizations
- Creating XML Data Models

- Using XML Files at Runtime
- Debugging XML Report Definitions

This chapter lists and provides examples of only some of the elements available in the reports.dtd file. This is the data type definition file that lists all elements and attributes associated with Reports XML. If you want more information on elements and attributes than this chapter provides, you can look in three additional sources:

- The reports.dtd file lists all possible Reports XML elements and attributes and, where present, the attributes' default values. The reports.dtd file is located in *ORACLE_HOME*\reports\dtd\ on both Windows and UNIX platforms. Many of the sub-elements include symbols that denote usage rules. For example:
  - A plus sign (+) means you can have one or more of this type of element in your XML file.
  - An asterisk (*) means you can have from zero to many of this type of element in your XML file.
  - A question mark (?) means you can have either zero or one of this type of element in your XML file.
  - No mark means the element is required, and you can have one and only one of this type of element in your XML file.

  If multiple sub-elements are enclosed in parentheses and followed by a symbol, the symbol applies to all enclosed sub-elements.
- In the Reports Builder online help, search for the related property or object. For example, for the XML attribute *width*, search the Reports Builder online help for the Width property. Some attributes are taken from Reports internal properties. Internal properties are read-only and cannot be changed. Internal properties are not documented in the online help.
- Build a report that includes the type of customization you are trying to build, save the report as XML, and view the saved file in a text editor. This provides an excellent means of seeing Reports XML in action and provides you with examples of the more complex models you may wish to build.

10.1 Customization Overview

Anything you can create using the Reports Builder, you can also create using Reports XML tags. Consequently you have the capability of performing customization on any conceivable Reports object you may care to.

Note: Although it is possible to create an entire report manually using the Reports XML tags, only manually created customizations and data models are documented and supported.

Creating and applying an XML customization is a three-step process:

1. Create a customization file using Reports XML tags.

   You can create this customization by building a report using the Oracle9*i*AS Reports Builder then saving your report as XML. You can also build the customization manually, with any sort of text editor or a sophisticated XML editor, as long as you include the XML tags that are required for the particular Reports customization.

2. Store the XML customization in a location that is accessible to Oracle9*i*AS Reports Services.

3. Apply the XML customization to another report with the `CUSTOMIZE` command line argument or the PL/SQL built-in `SRW.APPLY_DEFINITION`, or run the XML customization by itself (if it contains a complete report definition) with the `REPORT` (or `MODULE`) command line argument.

Note: For information on using these command line arguments, see Appendix A, "Command Line Arguments".

10.2 Creating XML Customizations

This section provides examples of various report customizations. It includes examples of:

- Required XML Tags
- Changing Styles
- Changing a Format Mask
- Adding Formatting Exceptions
- Adding Program Units and Hyperlinks
- Adding a New Query and Using the Result in a New Header Section

10.2.1 Required XML Tags

Every XML customization must contain the following required tag pair:

```
<report></report>
```

For example, the following is the most minimal XML customization possible:

```
<report name="emp" DTDVersion="9.0.2.0.0">
</report>
```

This XML customization would have a null effect if applied to a report because it contains nothing. It can be parsed because it has the needed tags, but it is useful only as an example of the required tags.

The `<report>` tag indicates the beginning of the report customization, its name, and the version of the Data Type Dictionary (DTD) file that is being used with this XML customization. The `</report>` tag indicates the end of the report customization.

The *report* element's *name* attribute can be any name you wish, either the name of the report the XML file will customize, or any other name.

This example represents a minimal use of the `<report>` tag. The `<report>` tag also has many attributes, most of which are implied and need not be specified. The only required `<report>` attribute is *DTDVersion*.

A full report definition requires both a data model and a layout and therefore also requires the following tags and their contents:

- `<data></data>`
- `<layout></layout>`

The *data* element has no accompanying attributes. The *layout* element has two attributes, both of which are required: *panelPrintOrder* and *direction*. If you use the default values for these attributes (respectively *acrossDown* and *default*), you don't need to specify them. Examples of the *data* and *layout* elements are provided in the following sections.

10.2.2 Changing Styles

The example in this section demonstrates the use of XML to change the fill and line colors used for report fields `F_Mincurrent_pricePersymbol` and `FMaxcurrent_pricePersymbol`.

```
<report name="anyName" DTDVersion="9.0.2.0.0">
```

```
    <layout>
        <section name="main">
            <field name="F_Mincurrent_pricePersymbol"
            source="Mincurrent_pricePersymbol"
            lineColor="black"
            fillColor="r100g50b50"/>
            <field name="F_Maxcurrent_pricePersymbol"
            source="Maxcurrent_pricePersymbol"
            lineColor="black"
            fillColor="r100g50b50"/>
        </section>
    </layout>
</report>
```

We assume in this example that the *section* and *field* elements' name attributes match the names of fields in the Main section of the report this XML file will customize. In keeping with this assumption, the other attributes of the *field* element will be applied only to the fields of the same name in the report's Main section.

10.2.3 Changing a Format Mask

The example in this section demonstrates the use of XML to change the format mask used for a report field `f_trade_date`.

```
<report name="anyName" DTDVersion="9.0.2.0.0">
    <layout>
        <section name="main"
            <field name="f_trade_date"
            source="trade_date"
            formatMask="MM/DD/RR"/>
        </section>
    </layout>
</report>
```

Notice that the *field* element provides its own closure (`/>`). If the *field* element used additional sub-elements, you would close it with `</field>`.

10.2.4 Adding Formatting Exceptions

The example in this section demonstrates the use of XML to add a formatting exception to highlight values greater than 10 in a report's `f_p_e` and `f_p_e1` fields.

```
<report name="anyName" DTDVersion="9.0.2.0.0">
    <layout>
```

```
        <section name="main">
            <field name="f_p_e" source="p_e">
               <exception textColor="red">
                  <condition source="p_e" operator="gt" operand1="10"/>
               </exception>
            </field>
            <field name="f_p_e1" source="p_e">
               <exception textColor="blue">
                  <condition source="p_e" operator="gt" operand1="10"/>
               </exception>
            </field>
        </section>
    </layout>
</report>
```

In this example, the value for *operator* is *gt*, for greater than. Operators include those listed in Table 10–1:

Table 10–1 Values for the operator attribute

Operator	*Usage*
eq	equal
lt	less than
lteq	less than or equal to
neq	not equal to
gt	greater than
gteq	greater than or equal to
btw	between
notBtw	not between
like	like
notLike	not like
null	null
notNull	not null

Notice also that, unlike the previous example, the *field* element in this example uses sub-elements, and, consequently, closes with `</field>`, rather than a self-contained closure (`/>`).

10.2.5 Adding Program Units and Hyperlinks

The example in this section demonstrates the use of XML to add a program unit to a report, which in turn adds a hyperlink from the employee social security number (:SSN) to employee details.

```
<report name="anyName" DTDVersion="9.0.2.0.0">
    <layout>
        <section name="header">
            <field name="F_ssn1"
            source="ssn1"
            formatTrigger="F_ssn1FormatTrigger"/>
        </section>
        <section name="main">
            <field name="F_ssn"
            source="ssn"
            formatTrigger="F_ssnFormatTrigger"/>
        </section>
    </layout>
    <programUnits>
        <function name="F_ssn1FormatTrigger">
        <textsource>
        <![CDATA[
            function F_ssn1FormatTrigger return boolean is
               begin
                  SRW.SET_HYPERLINK('#EMP_DETAILS_&<' || LTRIM(TO_CHAR(:SSN))
                  || '>');
                  return (TRUE);
               end;
        ]]>
        </function>
        <function name="F_ssnFormatTrigger">
        <![CDATA[
            function F_ssnFormatTrigger return boolean is
               begin
                  SRW.SET_LINKTAG('EMP_DETAILS_&<' || LTRIM(TO_CHAR(:SSN)) ||
                  '>');
                  return (TRUE);
               end;
        ]]>
        </textsource>
        </function>
    </programUnits>
</report>
```

A CDDATA tag is used around the PL/SQL to distinguish it from the XML. Use the same tag sequence when you embed HTML in your XML file. In this example, the functions are referenced by name from the *formatTrigger* attribute of the *field* element.

10.2.6 Adding a New Query and Using the Result in a New Header Section

The example in this section demonstrates the use of XML to add a new query to a report and a new header section that makes use of the query result.

```
<report name="ref" DTDVersion="9.0.2.0.0">
    <data>
        <dataSource name="Q_summary">
            <select>select portid ports, locname locations from portdesc
            </select>
        </dataSource>
    </data>
    <layout>
        <section name="header">
            <tabular name="M_summary" template="BLAFbeige.tdf">
               <labelAttribute font="Arial"
                   fontSize="10"
                   fontStyle="bold"
                   textColor="white"/>
               <field name="F_ports"
                   source="ports"
                   label="Port IDs"
                   font="Arial"
                   fontSize="10"/>
               <field name="F_locations"
                   source="locations"
                   label="Port Names"
                   font="Arial"
                   fontSize="10"/>
            </tabular>
        </section>
    </layout>
</report>
```

This example XML can be run by itself because it has both a data model and a complete layout.

Use aliases in your SELECT statements to ensure the uniqueness of your column names. If you do not use an alias, then the default name of the report column is used and could be something different from the name you expect (for example,

portid1 instead of portid). This becomes important when you must specify the *source* attribute of the *field* element, which requires you to supply the correct name of the source column (the field).

The *labelAttribute* element defines the formatting for the field labels in the layout. Because it lies outside of the open and close *field* element, it applies to all the labels in the tabular layout. If you wanted it to pertain to only one of the fields, then you place it inside the `<field></field>` tag pair. If there is both a global and local *labelAttribute* element (one outside and one inside the `<field></field>` tag pair), the local overrides the global.

10.3 Creating XML Data Models

Oracle9*i*AS Reports Services introduces a greater level of sophistication in the types of data models you can create using Reports XML tags. Use Reports XML for:

- Creating Multiple Data Sources
- Linking Between Data Sources
- Creating Group Hierarchies within Each Data Source
- Creating Cross-Product (Matrix) Groups
- Creating Formulas, Summaries, and Placeholders at any Level
- Creating Parameters

This section provides examples of these uses of Reports XML.

In addition to these data model types, Oracle9*i*AS Reports Services provides support for using PL/SQL in your Reports XML. This includes support for local program units, report-level triggers, and attached PL/SQL libraries.

10.3.1 Creating Multiple Data Sources

The `<data>` tag now supports the creation of multiple data sources as well as the new pluggable data sources. Each data source is enclosed within its own `<dataSource>` tag. The data type definition for the *dataSource* element is:

```
<!ELEMENT dataSource
    ((select|plugin|plsql),
    comment?,
    displayInfo?,
    formula*,
    group*)>
```

```
<!ATTLIST dataSource
    name CDATA #IMPLIED
    defaultGroupName CDATA #IMPLIED
    maximumRowsToFetch CDATA #IMPLIED>
```

The following example creates two SQL data sources and names them *Q_1* and *Q_2*. It also creates all the necessary columns for the data sources and the default group—giving the group the specified *defaultGroupName* or defaulting its own name if *defaultGroupName* is not specified.

```
<report name="anyname" DTDVersion="9.0.2.0.0">
    <data>
        <dataSource name="Q_1" defaultGroupName="G_DEPARTMENTS">
            <select>
               select * from departments
            </select>
        </dataSource>
        <dataSource name="Q_2" defaultGroupName="G_EMPLOYEES">
            <select>
               select * from employees
            </select>
        </dataSource>
    </data>
</report>
```

10.3.2 Linking Between Data Sources

In the presence of multiple data sources, it may be desirable to link the data sources together to create the appropriate data model. Reports data model link objects have also been exposed through Reports XML. They support both group- and column-level links. You can specify any number of links to create the required data model.

The data type definition for the *link* element is:

```
<!ELEMENT link EMPTY>
<!ATTLIST link
    name CDATA #IMPLIED
    parentGroup CDATA #IMPLIED
    parentColumn CDATA #IMPLIED
    childQuery CDATA #IMPLIED
    childColumn CDATA #IMPLIED
    condition (eq|lt|neq|gt|gteq|like|notLike) "eq"
    sqlClause (startWith|having|where) "where">
```

The *link* element is placed within a *data* element and can link any two *dataSource* objects defined within the *data* element. For example:

```
<report name="anyname" DTDVersion="9.0.2.0.0">
    <data>
        <dataSource name="Q_1" defaultGroupName="G_DEPARTMENTS">
            <select>
               select * from departments
            </select>
        </dataSource>
        <dataSource name="Q_2" defaultGroupName="G_EMPLOYEES">
            <select>
               select * from employees
            </select>
        </dataSource>
        <link name="L_1" parentGroup="G_DEPARTMENTS"
            parentColumn="DEPARTMENT_ID" childQuery="Q_2"
            childColumn="DEPARTMENT_ID1" condition="eq" sqlClause="where"/>
    </data>
</report>
```

Within the *link* element, the Reports defaulting mechanism recognizes DEPARTMENT_ID1 as an alias to the DEPARTMENT_ID column in the EMPLOYEES table without your having to explicitly create such an alias.

10.3.3 Creating Group Hierarchies within Each Data Source

With Oracle9*i*AS Reports Services, the complete group hierarchy is available to you. You can specify all the columns within each group and break the order of those columns. You can use formulas, summaries, and placeholders to further customize the objects within groups.

The data type definition for the *group* element is:

```
<!ELEMENT group
    (field|exception|rowDelimiter|xmlSettings|displayInfo|dataItem|formula|
    summary|placeholder|filter|comment)*>
<!ATTLIST group
    name CDATA #IMPLIED
    fillColor CDATA #IMPLIED
    lineColor CDATA #IMPLIED
    formatTrigger CDATA #IMPLIED>
```

The following example demonstrates the use of a *group* element to create a break group under a data source.

```
<report name="anyname" DTDVersion="9.0.2.0.0">
    <data>
        <dataSource name="Q_1">
            <select>
               select * from employees
            </select>
            <group name="G_DEPARTMENTS">
               <dataItem name="DEPARTMENT_ID"/>
            </group>
            <group name="G_EMPLOYEES">
               <dataItem="EMPLOYEE_ID"/>
               <dataItem="FIRST_NAME"/>
               <dataItem="LAST_NAME"/>
               <dataItem="JOB_ID"/>
               <dataItem="MANAGER_ID"/>
               <dataItem="HIRE_DATE"/>
               <dataItem="SALARY"/>
               <dataItem="COMMISSION_PCT"/>
            </group>
        </dataSource>
    </data>
</report>
```

10.3.4 Creating Cross-Product (Matrix) Groups

Cross-product groups allow you to define a matrix of any number of groups in the data model. The dimension groups in a cross product may exist in the same data source or may be combined from different data sources to create a matrix. In support of this flexibility, the `<crossProduct>` tag is placed within the `<data>` tag after all the data sources and groups have been created.

The data type definition for the *crossProduct* element is:

```
<!ELEMENT crossProduct
    (xmlSettings|displayInfo|dimension|(formula|summary|placeholder)*|comment)*>
<ATTLIST crossProduct
    name CDDATA #IMPLIED
    mailText CDDATA #IMPLIED>
```

The following example demonstrates the creation of a single-query matrix.

```
<report name="anyname" DTDVersion="9.0.2.0.0">
    <data>
        <dataSource name="Q_1">
            <select>
```

```
            select * from employees
        </select>
        <group name="G_DEPARTMENTS">
            <dataItem name="DEPARTMENT_ID"/>
        </group>
        <group name="G_JOB_ID>
            <dataItem name="JOB_ID"/>
        </group>
        <group name="G_MANAGER_ID">
            <dataItem name="MANAGER_ID"
        </group>
        <group name="G_EMPLOYEE_ID">
            <dataItem name="EMPLOYEE_ID"/>
            <dataItem name="FIRST_NAME"/>
            <dataItem name="LAST_NAME"/>
            <dataItem name="HIRE_DATE"/>
            <dataItem name="SALARY"/>
            <dataItem name="COMMISSION_PCT"/>
        </group>
    </dataSource>
    <crossProduct name="G_Matrix">
        <dimension>
            <group name="G_DEPARTMENTS">
        </dimension>
        <dimension>
            <group name="G_JOB_ID">
        </dimension>
        <dimension>
            <group name="G_MANAGER_ID">
        </dimension>
    </crossProduct>
  </data>
</report>
```

10.3.5 Creating Formulas, Summaries, and Placeholders at any Level

You can place formulas, summaries, and placeholders at any level within the data model. Additionally, you have complete control over all the attributes for each of these objects.

The following example demonstrates the creation of a report-level summary whose source is based on a group-level formula column.

```
<report name="anyname" DTDVersion="9.0.2.0.0">
  <data>
```

```
        <dataSource name="Q_1">
            <select>
               select * from employees
            </select>
            <group name="G_EMPLOYEES">
               <dataItem="EMPLOYEE_ID"/>
               <dataItem name="EMPLOYEE_ID"/>
               <dataItem name="FIRST_NAME"/>
               <dataItem name="LAST_NAME"/>
               <dataItem name="HIRE_DATE"/>
               <dataItem name="SALARY"/>
               <dataItem name="COMMISSION_PCT"/>
               <dataItem name="DEPARTMENT_ID"/>
               <formula name="CF_REMUNERATION" source="cf_1formula"
                  datatype="number" width="20" precision="10"/>
            </group>
        </dataSource>
        <summary name="CS_REPORT_LEVEL_SUMMARY" function="sum" width="20"
           precision="10" reset="report" compute="report"/>
    </data>
    <programUnits>
        <function name="cf_1formula" returnType="number">
            <textSource>
<![CDATA[
function CF_1Formula return Number is
begin
    return (:salary + nvl(:commission_pct,0));
end;
]]>
            </textSource>
        </function>
    </programUnits>
</report>
```

10.3.6 Creating Parameters

In Reports XML, the *parameter* element is placed between open and close *data* elements. The data type definition for the *parameter* element is:

```
<!ELEMENT parameter (comment?|listOfValues?)>
<!ATTLIST parameter
    name CDATA #REQUIRED
    datatype (number|character|date) "number"
    width CDATA "20"
    scale CDATA "0"
```

```
    precision CDATA "0"
    initialValue CDATA #IMPLIED
    inputMask CDATA #IMPLIED
    validationTrigger CDATA #IMPLIED
    label CDATA #IMPLIED
    defaultWidth CDATA #IMPLIED
    defaultHeight CDATA #IMPLIED>
```

The following example demonstrates a dynamic list of values (LOV), an initial value, and a validation trigger.

```
<report name="anyname" DTDVersion="9.0.2.0.0">
    <data>
        <dataSource name="Q_1" defaultGroupName="G_DEPARTMENTS">
            <select>
               select * from departments
            </select>
        </dataSource>
        <parameter name="P_LAST_NAME" datatype="character" precision="10"
            initialValue="SMITH" validationTrigger="p_last_namevalidtrigger"
            defaultWidth="0" defaultHeight="0">
            <listOfValues restrictToList="yes">
               <selectStatement hideFirstColumn="yes">
               <![CDATA[select last_name, 'last_name||'-'||employee_id'
                  from employees]]>
               </selectStatement>
            </listOfValues>
        </parameter>
    </data>
    <programUnits>
        <function name="p_last_namevalidtrigger" returnType="character">
            <textSource>
    <![CDATA[function P_LAST_NAMEValidTrigger return boolean is last_name
    char(20);
    begin
        select count(*) into last_name from employees
        where upper(last_name)=upper(:p_last_name);
            exception when OTHERS then return(FALSE);
            end;
        return(TRUE);
    end;
    ]]>
            </textSource>
        </function>
    </programUnits>
```

```
</report>
```

10.4 Using XML Files at Runtime

Once you have created your Reports XML customization file, you can use it in the following ways:

- You can apply XML report definitions to RDF or other XML files at runtime by specifying the CUSTOMIZE command line argument or the SRW.APPLY_DEFINITION built-in. Refer to "Applying an XML Report Definition at Runtime" for more information.
- You can run an XML report definition by itself (without another report) by specifying the REPORT (or MODULE) command line argument. Refer to "Running an XML Report Definition by Itself" for more information.
- You can use RWCONVERTER to make batch modifications using the CUSTOMIZE command line argument. Refer to "Performing Batch Modifications" for more information.

The following sections describe each of the cases in more detail and provide examples.

10.4.1 Applying an XML Report Definition at Runtime

To apply an XML report definition to an RDF or XML file at runtime, you can use the CUSTOMIZE command line argument or the SRW.APPLY_DEFINITION built-in. CUSTOMIZE can be used with RWCLIENT, RWRUN, RWBUILDER, RWCONVERTER, and URL report requests.

Note: Refer to "Performing Batch Modifications" for more information about using CUSTOMIZE with RWCONVERTER.

10.4.1.1 Applying One XML Report Definition

The following command line sends a job request to Oracle9*i*AS Reports Services and applies an XML report definition, EMP.XML, to an RDF file, EMP.RDF. In this example, the CUSTOMIZE command refers to a file located in a Windows directory path. For UNIX, specify the path according to UNIX standards (i.e., myreports/emp.xml).

```
RWCLIENT REPORT=emp.rdf CUSTOMIZE=\myreports\emp.xml
    USERID=<username>/<password>@<my_db> DESTYPE=file DESNAME=emp.pdf
```

```
    DESFORMAT=PDF SERVER=<server_name>
```

When you use RWRUN, the Oracle9*i*AS Reports Services runtime command, the equivalent command line would be:

```
RWRUN USERID=<username>/<password>@<my_db> REPORT=emp.rdf
   CUSTOMIZE=\myreports\emp.xml DESTYPE=file DESNAME=emp.pdf
   DESFORMAT=PDF
```

When testing your XML report definition, it is sometimes useful to run your report requests with additional arguments to create a trace file. For example:

```
TRACEFILE=emp.log TRACEMODE=trace_replace TRACEOPT=trace_app
```

> **Note:** Unless you care to change the default, it isn't necessary to include a trace in the command line if you have specified a default trace option in the Reports Server configuration file.

The trace file provides a detailed listing of the creation and formatting of the report objects.

10.4.1.2 Applying Multiple XML Report Definitions

You can apply multiple XML report definitions to a report at runtime by providing a list with the CUSTOMIZE command line argument. The following command line sends a job request to Oracle9*i*AS Reports Services that applies two XML report definitions, EMP0.XML and EMP1.XML, to an RDF file, EMP.RDF:

```
RWCLIENT REPORT=emp.rdf
    CUSTOMIZE="(D:\CORP\MYREPORTS\EMP0.XML,D:\CORP\MYREPORTS\EMP1.XML)"
    USERID=<username>/<password>@<my_db> DESTYPE=file DESNAME=emp.pdf
    DESFORMAT=PDF SERVER=<server_name>
```

> **Note:** In this example, the CUSTOMIZE command entry demonstrates a directory path to files stored on a Windows platform. For UNIX, use that platform's standard for specifying directory paths (i.e., forward slashes instead of backward).

If you were using Oracle9*i*AS Reports Services Runtime, then the equivalent command line would be:

```
RWRUN REPORT=emp.rdf
```

```
CUSTOMIZE="(D:\CORP\MYREPOORTS\EMP0.XML,D:\CORP\MYREPORTS\EMP1.XML)"
USERID=<username>/<password>@<my_db> DESTYPE=file DESNAME=emp.pdf
DESFORMAT=PDF
```

10.4.1.3 Applying an XML Report Definition in PL/SQL

To apply an XML report definition to an RDF file in PL/SQL, use the SRW.APPLY_DEFINITION and SRW.ADD_DEFINITION built-ins in the BeforeForm or AfterForm trigger. The following sections provide examples of these built-ins.

10.4.1.3.1 Applying an XML Definition Stored in a File To apply XML that is stored in the file system to a report, use the SRW.APPLY_DEFINITION built-in in the BeforeForm or AfterForm triggers of the report.

On **Windows**:

```
SRW.APPLY_DEFINITION ('\<ORACLE_HOME>\TOOLS\DOC\US\RBBR\COND.XML');
```

On **UNIX**:

```
SRW.APPLY_DEFINITION ('<ORACLE_HOME>/TOOLS/DOC/US/RBBR/COND.XML');
```

When the report is run, the trigger executes and the specified XML file is applied to the report.

10.4.1.3.2 Applying an XML Definition Stored in Memory To create an XML report definition in memory, you must add the definition to the document buffer using SRW.ADD_DEFINITION before applying it using SRW.APPLY_DEFINITION.

The following example illustrates how to build up and apply several definitions in memory based upon parameter values entered by the user. The PL/SQL in this example is used in the AfterParameterForm trigger of a report called videosales_custom.rdf.

The videosales_custom.rdf file contains PL/SQL in its AfterParameterForm trigger that does the following:

- Conditionally highlights fields based upon parameter values entered by the user at runtime.
- Changes number format masks based upon parameter values entered by the user at runtime.

The following tips are useful when looking at this example:

- Each time you use SRW.APPLY_DEFINITION, the document buffer is flushed and you must begin building a new XML report definition with SRW.ADD_DEFINITION.
- Notice the use of the parameters hilite_profits, hilite_costs, hilite_sales, and money_format to determine what to include in the XML report definition. The hilite_profits, hilite_costs, and hilite_sales parameters are also used in the formatting exceptions to determine which values to highlight.
- Because of the upper limit on the size of VARCHAR2 columns (4000 bytes), you might need to spread very large XML report definitions across several columns. If so, then you might have to create several definitions in memory and apply them separately rather than creating one large definition and applying it once.

```
function AfterPForm return boolean is
begin
SRW.ADD_DEFINITION('<report name="vidsales_masks"
author="Generated" DTDVersion="9.0.2.0.0">');
IF :MONEY_FORMAT='$NNNN.00' THEN SRW.ADD_DEFINITION('<layout>');
    SRW.ADD_DEFINITION('<section name="main">');
    SRW.ADD_DEFINITION('<field name="F_TOTAL_PROFIT" source="TOTAL_PROFIT"
    formatMask="LNNNNNNNNNNN0D00"/>');
    SRW.ADD_DEFINITION('<field name="F_TOTAL_SALES" source="TOTAL_SALES"
    formatMask="LNNNNNNNNNNN0D00"/>');
    SRW.ADD_DEFINITION('<field name="F_TOTAL_COST" source="TOTAL_COST"
    formatMask="LNNNNNNNNNNN0D00"/>');
    SRW.ADD_DEFINITION('<field name="F_SumTOTAL_PROFITPerCITY" source="SumTOTAL_
    PROFITPerCITY"
        formatMask="LNNNNNNNNNNN0D00"/>');
    SRW.ADD_DEFINITION('<field name="F_SumTOTAL_SALESPerCITY" source="SumTOTAL_
    SALESPerCITY"
        formatMask="LNNNNNNNNNNN0D00"/>');
    SRW.ADD_DEFINITION('<field name="F_SumTOTAL_COSTPerCITY" source="SumTOTAL_
    COSTPerCITY"
        formatMask="LNNNNNNNNNNN0D00"/>');
    SRW.ADD_DEFINITION('</section>');
    SRW.ADD_DEFINITION('</layout>');
ELSIF :MONEY_FORMAT='$NNNN' THEN SRW.ADD_DEFINITION('<layout>');
    SRW.ADD_DEFINITION('<section name="main">');
    SRW.ADD_DEFINITION('<field name="F_TOTAL_PROFIT" source="TOTAL_PROFIT"
    formatMask="LNNNNNNNNNN0"/>');
    SRW.ADD_DEFINITION('<field name="F_TOTAL_SALES" source="TOTAL_SALES"
    formatMask="LNNNNNNNNNN0"/>');
    SRW.ADD_DEFINITION('<field name="F_TOTAL_COST" source="TOTAL_COST"
```

```
    formatMask="LNNNNNNNNNN0"/>');
    SRW.ADD_DEFINITION('<field name="F_SumTOTAL_PROFITPerCITY" source="SumTOTAL_
    PROFITPerCITY" formatMask="LNNNNNNNNNN0"/>');
    SRW.ADD_DEFINITION('<field name="F_SumTOTAL_SALESPerCITY" source="SumTOTAL_
    SALESPerCITY" formatMask="LNNNNNNNNNN0"/>');
    SRW.ADD_DEFINITION('<field name="F_SumTOTAL_COSTPerCITY" source="SumTOTAL_
    COSTPerCITY" formatMask="LNNNNNNNNNN0"/>');
    SRW.ADD_DEFINITION('</section>');
    SRW.ADD_DEFINITION('</layout>');
END IF;
SRW.ADD_DEFINITION('</report>');
SRW.APPLY_DEFINITION;
SRW.ADD_DEFINITION('<report name="vidsales_hilite_costs" author="Generated"
DTDVersion="9.0.2.0.0">');
IF :HILITE_COSTS <> 'None' THEN SRW.ADD_DEFINITION('<layout>');
    SRW.ADD_DEFINITION('<section name="main">');
    SRW.ADD_DEFINITION('<field name="F_TOTAL_COST" source="TOTAL_COST">');
    SRW.ADD_DEFINITION('<exception textColor="red">');
    SRW.ADD_DEFINITION('<condition source="TOTAL_COST" operator="gt"
    operand1=":hilite_costs"/>');
    SRW.ADD_DEFINITION('</exception>');
    SRW.ADD_DEFINITION('</field>');
    SRW.ADD_DEFINITION('</section>');
    SRW.ADD_DEFINITION('</layout>');
END IF;
SRW.ADD_DEFINITION('</report>');
SRW.APPLY_DEFINITION;
SRW.ADD_DEFINITION('<report name="vidsales_hilite_sales" author="Generated"
DTDVersion="9.0.2.0.0">');
IF :HILITE_SALES <> 'None' THEN SRW.ADD_DEFINITION('<layout>');
    SRW.ADD_DEFINITION('<section name="main">');
    SRW.ADD_DEFINITION('<field name="F_TOTAL_SALES" source="TOTAL_SALES">');
    SRW.ADD_DEFINITION('<exception textColor="red">');
    SRW.ADD_DEFINITION('<condition source="TOTAL_SALES" operator="gt"
    operand1=":hilite_sales"/>');
    SRW.ADD_DEFINITION('</exception>');
    SRW.ADD_DEFINITION('</field>');
    SRW.ADD_DEFINITION('</section>');
    SRW.ADD_DEFINITION('</layout>');
END IF;
SRW.ADD_DEFINITION('</report>');
SRW.APPLY_DEFINITION;
SRW.ADD_DEFINITION('<report name="vidsales_hilite_profits" author="Generated"
DTDVersion="9.0.2.0.0">');
IF :HILITE_PROFITS <> 'None' THEN SRW.ADD_DEFINITION('<layout>');
```

```
    SRW.ADD_DEFINITION('<section name="main">');
    SRW.ADD_DEFINITION('<field name="F_TOTAL_PROFIT" source="TOTAL_PROFIT">');
    SRW.ADD_DEFINITION('<exception textColor="red">');
    SRW.ADD_DEFINITION('<condition source="TOTAL_PROFIT" operator="gt"
    operand1=":hilite_profits"/>');
    SRW.ADD_DEFINITION('</exception>');
    SRW.ADD_DEFINITION('</field>');
    SRW.ADD_DEFINITION('</section>');
    SRW.ADD_DEFINITION('</layout>');
END IF;
SRW.ADD_DEFINITION('</report>');
SRW.APPLY_DEFINITION;
return (TRUE);
end;
```

10.4.2 Running an XML Report Definition by Itself

To run an XML report definition by itself, you send a request with an XML file specified in the REPORT (or MODULE) argument. The following command line sends a job request to Oracle9*i*AS Reports Services to run a report, `emp.xml`, by itself:

```
RWCLIENT USERID=<username>/<password>@<my_db>
    REPORT=C:\CORP\MYREPORTS\EMP.XML
    DESTYPE=file DESNAME=emp.pdf DESFORMAT=PDF
    SERVER=<server_name>
```

When you use RWRUN, the Oracle9*i*AS Reports Services runtime command, the equivalent command line would be:

```
RWRUN USERID=<username>/<password>@<my_db>
  REPORT=C:\CORP\MYREPORTS\EMP.XML
  DESTYPE=file DESNAME=emp.pdf DESFORMAT=PDF
```

When you run an XML report definition in this way, you must specify an XML file extension. You could also apply an XML customization file to this report using the CUSTOMIZE argument.

10.4.3 Performing Batch Modifications

If you have a large number of reports that need to be updated, then you can use the CUSTOMIZE command line argument with RWCONVERTER to perform modifications in batch. Batch modifications are particularly useful when you must make a repetitive change to a large number of reports (for example, changing a field's format mask). Rather than opening each report and manually making the change in

the Reports Builder, you can run RWCONVERTER once and make the same change to a large number of reports at once.

The following example applies two XML report definitions, translate.xml and customize.xml, to three RDF files, INVEN1.RDF, INVEN2.RDF, and MANU.RDF, and saves the revised definitions to new files, INVEN1_NEW.RDF, INVEN2_NEW.RDF, and MANU_NEW.RDF.

```
RWCONVERTER <username>/<password>@<my_db>
    STYPE=rdffile
    SOURCE="(inven1.rdf, inven2.rdf, manu.rdf)"
    DTYPE=rdffile
    DEST="(inven1_new.rdf, inven2_new.rdf, manu_new.rdf)"
    CUSTOMIZE="(D:\APPS\TRANS\TRANSLATE.XML,D:\APPS\CUSTOM\CUSTOMIZE.XML)"
    BATCH=yes
```

Note: In this example, the CUSTOMIZE command entry demonstrates a directory path to files stored on a Windows platform. For UNIX, use that platform's standard for specifying directory paths (i.e., forward slashes instead of backward).

10.5 Debugging XML Report Definitions

The following features are available to help you debug your XML Report files:

- XML Parser Error Messages
- Tracing Options
- RWBUILDER
- Writing XML to a File for Debugging

These features are discussed in the following sections.

10.5.1 XML Parser Error Messages

The XML parser is part of Oracle's XML Development Kit (XDK), which is delivered with the core Oracle Database release. The XML parser is a Java package that checks the validity of XML syntax. The JAR files that contain the XML parser are automatically set up on install and are available to Reports.

The XML parser catches most syntax errors and displays an error message. The error message contains the line number in the XML where the error occurred as well as a brief description of the problem.

For more information on the XML parser, see the Oracle Technology Network (http://otn.oracle.com). Search for *XML parser* or *XDK*. Information is also available in the documentation that came with your Oracle Database.

10.5.2 Tracing Options

When testing your XML report definition, it can be useful to run your report along with additional arguments to create a trace file. For example:

```
RWRUN <username>/<password>@<my_db>
    REPORT=\CORP\MYREPORTS\EMP.XML
    TRACEFILE=emp.log
    TRACEMODE=trace_replace
    TRACEOPT=trace_app
```

The last three arguments in this command line generate a trace file that provides a detailed listing of report processing. The default location for trace file logs is the same on Windows and UNIX platforms:

```
ORACLE_HOME\reports\logs\
```

Note: In this example, the REPORT command entry and the path to the trace log demonstrate directory paths to files stored on a Windows platform. For UNIX, use that platform's standard for specifying directory paths (i.e., forward slashes instead of backward).

10.5.3 RWBUILDER

When designing an XML report definition, it is sometimes useful to open it in the Reports Builder. In Reports Builder, you can quickly determine if the objects are being created or modified as expected. For example, if you are creating summaries in an XML report definition, then opening the definition in the Reports Builder enables you to quickly determine if the summaries are being placed in the appropriate group in the data model.

To open a full report definition in the Reports Builder, use the REPORT (or MODULE) keyword. For example:

```
RWBUILDER USERID=<username>/<password>@<my_db>
    REPORT=C:\CORP\MYREORTS\EMP.XML
```

To open a partial report definition in Oracle9*i*DS Reports Services Builder, use the CUSTOMIZE keyword. For example:

```
RWBUILDER USERID=<username>/<password>@<my_db> REPORT=EMP.RDF
    CUSTOMIZE=C:\MYREPORTS\EMP.XML
```

> **Note:** In this example, the REPORT command entry demonstrates a directory path to files stored on a Windows platform. For UNIX, use that platform's standard for specifying directory paths (i.e., forward slashes instead of backward).

In both cases, the Reports Builder is opened with the XML report definition in effect. You can then use the various views of the Oracle9*i*AS Reports Services Editor to determine if the report is being created or modified as you expected.

10.5.4 Writing XML to a File for Debugging

If you are using SRW.ADD_DEFINTION to build an XML report definition in memory, then it can be helpful to write the XML to a file for debugging purposes. The following example demonstrates a procedure that writes each line that you pass to it to the document buffer in memory and, optionally, to a file that you specify.

```
PROCEDURE addaline (newline VARCHAR, outfile Text_IO.File_Type) IS
BEGIN
  SRW.ADD_DEFINITION(newline);
  IF :WRITE_TO_FILE='Yes' THEN
    Text_IO.Put_Line(outfile, newline);
  END IF;
END;
```

For this example to work, the PL/SQL that calls this procedure would need to declare a variable of type TEXT_IO.File_Type. For example:

```
custom_summary  Text_IO.File_Type;
```

You would also need to open the file for writing and call the addaline procedure, passing it the string to be written and the file to which it should be written. For example:

```
custom_summary := Text_IO.Fopen(:file_directory || 'vid_summ_per.xml', 'w');
addaline('<report name="video_custom" author="Generated"
DTDVersion="9.0.2.0.0">',
    custom_summary);
```

11

Event-Driven Publishing

Modern business processes often require the blending of automation into the work environment through the invocation of behind-the-scenes functions and procedures. Behind-the-scenes tasks can include the automatic production of output such as an invoice that prints automatically when an order is processed, a Web site that is automatically updated with current data, or an automatic e-mail with fresh report output when a transaction is completed.

Automatic output in response to events used to be a fairly complicated effort, particularly if you wished to produce the same results possible through interactive, RAD development tools, such as Oracle9*i* Reports Developer.

To address the requirement of automatic output, Oracle introduced a scheduling mechanism in Oracle9*i*AS Reports Services that enabled the invocation of reports on a scheduled basis without requiring additional user interaction. But this left one requirement unresolved: the ability to automatically run a report in response to an event in the database, such as the insertion of a record or the change of a value.

With the Oracle9*i*AS Reports Services Event-Driven Publishing API, you can automatically run a report in response to an event in the database, such as the insertion of a record or the change of a value. The Event-Driven Publishing API is a PL/SQL API that allows for the automatic submission of jobs to Oracle9*i*AS Reports Services from within the database.

This chapter provides a look at the Event-Driven Publishing API and includes examples of its use. It includes the following sections:

- The Event-Driven Publishing API
- Debugging Applications That Use the Event-Driven Publishing API
- Invoking a Report From a Database Event
- Integrating with Oracle9i Advanced Queuing

11.1 The Event-Driven Publishing API

The Event-Driven Publishing API is a PL/SQL Package that provides the basic functions required for the development of procedures that respond to events in the database. Event-driven jobs are submitted using the HTTP protocol. The server assigns a unique job_ident record to every call, useful for tracking the status of the job.

11.1.1 Elements of the API

The API consists of several key elements:

- The **SRW-Package** contains all relevant functions and procedures for submitting jobs, checking job status, and cancelling jobs, as well as manipulating parameter lists.
- The **ParamList-Type** defines a parameter list. A parameter list is the main vehicle for passing values when submitting a job. A parameter list is required for each job submission. It must contain several key parameters.
- The **ParamList-Object** is required for such features as Advanced Queuing, where a parameter list must be stored in the database so that it may be passed along with a message.

These API elements are discussed in more detail in the following sections.

The API is installed together with Oracle9*i*AS Reports Services Security and Oracle9*i*AS Portal, but neither is required. Installation scripts are also available separately should you want to install the API into a database that does not also hold Oracle Portal:

- **srwAPIins.sql** installs the Events-Driven Publishing API.
- **srwAPIgrant.sql** grants access privileges to the API. Run this script for each user to whom you will grant access to the API. If everyone may have access, you can run this once and grant access to PUBLIC.
- **srwAPIdrop.sql** removes the API.

11.1.2 Creating and Manipulating a Parameter List

A parameter list is a PL/SQL variable of type SRW_PARAMLIST. A variable of this type is an array of 255 elements of type SRW_PARAMETER, which itself consists of two attributes: NAME and VALUE. The API provides procedures for manipulating parameter lists, including:

- Add_Parameter
- Remove_Parameter
- Clear_Parameter_List

These procedures allow you to manipulate your parameter lists. They are discussed briefly in this section. You'll find more information in the Oracle9*i*AS Reports API documentation.

Note: Look for upcoming information about Reports APIs on the Oracle Technology Network: http://otn.oracle.com.

11.1.2.1 Add_Parameter

Whenever you use a parameter list for the first time, it must be initialized before you can add parameters to it. For example:

```
DECLARE
    myPlist SRW_PARAMLIST;
BEGIN
    myPlist := SRW_PARAMLIST(SRW_PARAMETER('',''));
    srw.add_parameter(myPlist,'myParameter','myValue');
END;
```

Both attributes of a parameter (NAME and VALUE) are of type VARCHAR2 and may not exceed a length of 80 characters for the NAME and 255 characters for the value.

The ADD_PARAMETER function has a third—optional—attribute, called MODE. MODE determines whether a parameter will be overwritten or an error raised in the event that a parameter with the same name already exists. To specify that an error will be raised in the event of duplicate names, use the constant CHECK_FOR_ EXISTANCE. This is the default value for the MODE attribute. To specify that a parameter will be overwritten in the event of duplicate names, use the constant OVERWRITE_IF_EXISTS.

11.1.2.2 Remove_Parameter

Use REMOVE_PARAMETER to remove a parameter from a parameter list. Call the procedure, and pass the parameter list from which you want to remove a parameter along with the name of the parameter you want to remove.

For example:

```
DECLARE
    myPlist SRW_PARAMLIST;
BEGIN
    myPlist := SRW_PARAMLIST(SRW_PARAMETER('',''));
    srw.add_parameter(myPlist,'myParameter','myValue');
    srw.remove_parameter(myPlist,'myParameter');
END;
```

11.1.2.3 Clear_Parameter_List

To remove ALL parameters from your list, use CLEAR_PARAMETER_LIST. For example:

```
DECLARE
    myPlist SRW_PARAMLIST;
BEGIN
    myPlist := SRW_PARAMLIST(SRW_PARAMETER('',''));
    srw.add_parameter(myPlist,'myParameter','myValue');
    srw.clear_parameter_list(myPlist);
END;
```

This will remove all parameters from your list.

11.1.3 How to Submit a Job

A parameter list contains all vital parameters for submitting a job. The job type determines which parameters are required on the list to enable the Reports Server to process the request.

The listed parameters are the same ones that you must specify when you submit a job from a browser to the Reports Servlet. In such a case, if the job is a report you will need at least the following parameters but may have more:

- **GATEWAY** provides the URL to the Reports Servlet you will use to process the request.
- **SERVER** identifies the Reports Server to be used in conjunction with the servlet.
- **REPORT** identifies the report file to be run.
- **USERID** identifies the name and user ID of the person running the report.
- **AUTHID** provides authorization information in the event you are running against a secured server.

Each request returns a job_ident record that holds the information required to identify the job uniquely. This information is stored in variable of type SRW.JOB_IDENT. Be aware that this is a PACKAGE-TYPE and must be referenced SRW.JOB_IDENT; while the parameter list is an OBJECT-TYPE and must be referenced SRW_PARAMLIST.

For example:

```
 DECLARE
    myPlist SRW_PARAMLIST;
    myIdent SRW.Job_Ident;
 BEGIN
    myPlist := SRW_PARAMLIST(SRW_PARAMETER('',''));
    srw.add_parameter(myPlist,'GATEWAY','http://…');
    srw.add_parameter(myPlist,'SERVER','mySVR');
    srw.add_parameter(myPlist,'REPORT','myReport.RDF');
    srw.add_parameter(myPlist,'USERID','me/secret');
    myIdent := srw.run_report(myPlist);
END;
```

The API method RUN_REPORT takes a parameter list that contains all vital information as input (via ADD_PARAMETER), creates and submits the request, and returns the job_ident record.

The job_ident record contains the following parameters:

- MyIdent.GatewayURL
- MyIdent.ServerName
- MyIdent.JobID
- MyIdent.AuthID

These parameters are needed by the SRW.REPORT_STATUS function to get status information for a submitted job.

11.1.4 How to Check for Status

The Event-Driven Publishing API provides a two-way communication with the Reports Server. You submit a job to the server, and you can query the status of this job from the server using the SRW.REPORT_STATUS function.

This function will return a record of type SRW.STATUS_RECORD that holds the same information you would see in the job-status display if you were using the Reports Servlet's SHOWJOBS command.

For example:

```
DECLARE
    myPlist SRW_PARAMLIST;
    myIdent SRW.Job_Ident;
    myStatus SRW.Status_Record;
BEGIN
    myPlist := SRW_PARAMLIST(SRW_PARAMETER('',''));
    srw.add_parameter(myPlist,'GATEWAY','http://…');
    srw.add_parameter(myPlist,'SERVER','mySVR');
    srw.add_parameter(myPlist,'REPORT','MyReport.RDF');
    srw.add_parameter(myPlist,'USERID','me/secret');
    myIdent := srw.run_report(myPlist);
    myStatus := srw.report_status(myIdent);
END;
```

You can use the returned status record for fetching information about the status of your job.

11.1.5 Using the Servers' Status Record

The status record contains processing information about your job. It contains the same information found in the server queue (SHOWJOBS). Additionally, it contains information about the files produced for finished jobs and the lineage for scheduled jobs.

The most important information in the status record is the current job status and the status text, used in turn to check for runtime errors and their causes.

You can use timing information to determine if a job is subject to cancellation because it has exceeded its predicted time for completion.

One way to use the status record is to cancel a job. The Event-Driven Publishing API offers a method for cancelling a job that has been submitted to the server. This might be handy if you want to remove a job that has exceeded its allowed time to run or if you simply have scheduled jobs you want to cancel.

To cancel a job, use the following procedure:

```
DECLARE
    myPlist SRW_PARAMLIST;
    myIdent SRW.Job_Ident;
    myStatus SRW.Status_Record;
BEGIN
    myPlist := SRW_PARAMLIST(SRW_PARAMETER('',''));
    srw.add_parameter(myPlist,'GATEWAY','http://…');
```

```
    srw.add_parameter(myPlist,'SERVER','mySVR');
    srw.add_parameter(myPlist,'REPORT','myReport.RDF');
    srw.add_parameter(myPlist,'USERID','me/secret');
    myIdent := srw.run_report(myPlist);
    srw.cancel_report(myIdent);
END;
```

As evident in this example, you cancel a report by calling the CANCEL_REPORT procedure (srw.cancel_report) and passing it the job_ident record of the job you want to cancel. The procedure takes an optional parameter list to enable you to pass any additional parameters you might need.

11.2 Debugging Applications That Use the Event-Driven Publishing API

Because these processes all run behind the scenes, there is no actual place where debugging information is produced during normal execution. Therefore, the API has two procedures that toggle a special debugging mode that produces extensive debugging information via DBMS_OUTPUT:

- SRW.START_DEBUGGING
- SRW.STOP_DEBUGGING

To switch on debugging mode simply call SRW.START_DEBUGGING and to stop it call SRW.STOP_DEBUGGING. The debugging-mode must be started immediately before you run your actual logic. It stays on as long as the current instance of the package is loaded.

One way you can display this information is by setting SERVEROUT to ON in SQL*PLUS before you run your script.

In addition to this method of debugging, the API has a set of pre-defined exceptions to be used for error handling. You'll find examples of these exceptions in the srw_test.sql script provided with your Oracle9*i*AS Reports Services installation. Additionally, see the Reports API reference documentation for a detailed explanation of these exceptions.

Note: Look for upcoming information about Reports APIs on the Oracle Technology Network: http://otn.oracle.com.

11.3 Invoking a Report From a Database Event

Database triggers are the primary mechanism for invoking reports using the Event-Driven Publishing API. The Oracle database allows you to define various scopes of triggers that fire in response to various events. To submit a database-driven job, you use the code described in the previous sections within a database trigger.

There are many ways to use event-driven publishing. One way is to create security protocols using a trigger that fires whenever a grant is done or a user logs on or off. Another way is to create automated processes that respond to certain types of changes to data in a table. For example, a database trigger could fire when the status of an expense report changes to DONE; in turn, a report could automatically be sent to an employee's manager.

For example:

```
CREATE TRIGGER EXP_REP_TRG
    AFTER INSERT OR UPDATE on EXP_REP FOR EACH ROW
    myPlist SRW_PARAMLIST;
    myIdent SRW.Job_Ident;
BEGIN
    IF (:new.ExpStat = 'DONE') THEN
        myPlist := SRW_PARAMLIST(SRW_PARAMETER('',''));
        srw.add_parameter(myPlist,'GATEWAY','http://…');
        srw.add_parameter(myPlist,'SERVER','fooSVR');
        srw.add_parameter(myPlist,'REPORT','foo.RDF');
        srw.add_parameter(myPlist,'USERID','foo/bar');
        srw.add_parameter(myPlist,'ExpenseID',:new.ExpID);
        myIdent := srw.run_report(myPlist);
    END IF;
END;
```

This trigger will fire after each update on the EXP_REP table. In the event the status changes to DONE, the report request is run.

If you want your request to run against a key specified in the cgicmd.dat file, specify the CMDKEY parameter in lieu of the REPORT parameter. If the key contains user ID information, you can omit the USERID parameter as well. For example:

```
CREATE TRIGGER EXP_REP_TRG
    AFTER INSERT OR UPDATE on EXP_REP FOR EACH ROW
    myPlist SRW_PARAMLIST;
    myIdent SRW.Job_Ident;
BEGIN
    IF (:new.ExpStat = 'DONE') THEN
```

```
        myPlist := SRW_PARAMLIST(SRW_PARAMETER('',''));
        srw.add_parameter(myPlist,'GATEWAY','http://…');
        srw.add_parameter(myPlist,'SERVER','fooSVR');
        srw.add_parameter(myPlist,'CMDKEY','keyvalue');
        srw.add_parameter(myPlist,'ExpenseID',:new.ExpID);
        myIdent := srw.run_report(myPlist);
    END IF;
END;
```

Additionally, if you have defined an advanced distribution model via a distribution XML file, you can specify that file with the DIST parameter. For example:

```
CREATE TRIGGER EXP_REP_TRG
    AFTER INSERT OR UPDATE on EXP_REP FOR EACH ROW
    myPlist SRW_PARAMLIST;
    myIdent SRW.Job_Ident;
BEGIN
    IF (:new.ExpStat = 'DONE') THEN
        myPlist := SRW_PARAMLIST(SRW_PARAMETER('',''));
        srw.add_parameter(myPlist,'GATEWAY','http://…');
        srw.add_parameter(myPlist,'SERVER','fooSVR');
        srw.add_parameter(myPlist,'REPORT','foo.RDF');
        srw.add_parameter(myPlist,'USERID','foo/bar');
        srw.add_parameter(myPlist,'DISTRIBUTE','YES');
        srw.add_parameter(myPlist,'DESTINATION','filename.xml');
        srw.add_parameter(myPlist,'ExpenseID',:new.ExpID);
        myIdent := srw.run_report(myPlist);
    END IF;
END;
```

This is one way to move this kind of logic from your application into the database and use the database as a central storage for business processes.

Note: You'll find additional examples of the Event-Driven Publishing API in action in the demo script srw_test.sql, included with your Oracle9*i*AS Reports Services installation.

11.4 Integrating with Oracle9*i* Advanced Queuing

Oracle Advanced Queuing is a means for building an asynchronous request/response mechanism around a so-called queue and two processes: ENQUEUE, which puts MESSAGES into a queue, and DEQUEUE, which reads the queue.

Advanced queuing provides sophisticated mechanisms for distributing messages across queues and for queue subscription. These mechanisms are all built on top of these basic elements (ENQUEUE, DEQUEUE, and MESSAGES).

With the Event-Driven Publishing API you can use these queues to store and transmit report jobs. You can even build your own queuing mechanism if the one provided with Oracle9*i*AS Reports Services does not fit your needs.

11.4.1 Creating a Queue That Holds Messages of Type SRW_PARAMLIST

A queue is a table in the database that holds, along with several administrative columns, an object column that represents a message. In our case the message is the parameter list.

The `dbms_AQadm` package, provided with Advanced Queuing, contains all the administrative functions required for setting up an advanced queuing system.

Use `dbms_AQadm.Create_Queue_Table` to create the physical table in the database. You must pass it a name for the table and a name for the object type that will define the message for this queue.

For example:

```
…
execute dbms_AQadm.Create_Queue_Table
    (queue_Table=>'queuename._tab',
    queue_Payload_Type=>'SRW_ParamList_Object',
    compatible=>'9.0');
```

In earlier examples, we created the object type SRW_PARAMLIST_OBJECT that encapsulates the SRW_PARAMLIST type in object notation so it can be used as a message.

After creating the queue table, you must create the queue with `dbms_AQadm.Create_Queue` and start the queue with `dbms_AQadm.Start_Queue`.

For example:

```
…
execute dbms_AQadm.Create_Queue
    (Queue_Name=>'queuename',Queue_Table=>'queuename._tab');
prompt … starting queue
execute dbms_AQadm.Start_Queue
    (Queue_Name=>'queuename');
…
```

Note: You'll find a complete example for setting up, creating, and starting a simple queue in the demo file `srwAQsetup.sql`, included with your Oracle9*i*AS Reports Services installation.

Having created and started the queue, what you need now is a procedure that creates a message in this queue and a procedure that reads out the queue and submits the job to the server. These are discussed in the following sections.

11.4.2 Creating the Enqueuing Procedure

The enqueuing procedure is responsible for putting a message into the queue. This procedure can be part of your application, called by a database-trigger, or provided via an external mechanism. In this section, we will provide an example of creating a stored procedure that puts a simple message in this queue.

Because our message is the parameter list itself, the procedure is fairly easy. We use the same code we used in earlier sections to create a parameter list. In addition to the variables we used, we define an `object` variable to hold the message we will put into the queue.

```
…
  plist_object SRW_ParamList_Object;
…
```

After creating the parameter list we create the actual message object using the object constructor.

```
…
plist_object := SRW_ParamList_Object(plist);
…
```

Then we enqueue the message using the enqueue procedure provided by Advanced Queuing.

```
…
dbms_aq.enqueue(queue_name => 'myQueue',
    enqueue_options => enqueue_options,
    message_properties => message_properties,
    payload => PList_Object,
    msgid => message_handle);
…
```

The message is put into the queue. Because we did not set up any message distribution, the message will stay in the queue until it is fetched by a dequeuing-procedure, which is discussed in the next section.

Note: For the exact syntax of `dbms_aq.enqueue` refer to the Advanced Queuing API Reference document.

You'll find additional examples in the `srwAQsetup.sql` file included with your Oracle9*i*AS Reports Services installation.

Look for upcoming information about Reports APIs on the Oracle Technology Network: http://otn.oracle.com.

11.4.3 Creating the Dequeuing Procedure

A dequeuing procedure reads out all available messages in a queue and processes them. In our case, we want to read out the message and submit a job to the server using the parameter list that was attached to the message.

To accomplish this, we follow this example:

```
BEGIN
    dequeue_options.wait := 1;
loop
    DBMS_AQ.DEQUEUE(queue_name => 'myQueue',
        dequeue_options => dequeue_options,
        message_properties => message_properties,
        payload => PList_Object,
        msgid => message_handle);
    COMMIT;
    plist := plist_object.params;
    r_jid := SRW.run_report(plist);
end loop;
exception when aq_timeout then
    begin
        NULL;
    end;
END;
```

This code example will read out the queue until all messages have been processed. Time allowed for processing is determined by the time-out defined in the second line of code. This time-out defines the amount of seconds the dequeue procedure should wait for a message before creating a time-out exception.

The DBMS_AQ.DEQUEUE built-in is provided by Advanced Queuing for reading out messages. It puts the payload of the message, the object that holds the information, into the object defined by the payload parameter.

Using plist, we extract the information from the payload object. As mentioned before, our object holds a parameter list. It is stored in the attribute PARAMS inside the object. The extracted parameter list is then handed over to SRW.RUN_REPORT for submitting the job.

If you want to avoid the need for invoking this dequeuing procedure by hand, you can run it as a job inside the database.

Part III

National Language Support and Bidirectional Support

Part III provides information about Reports-related National Language Support settings and bidirectional support. It includes the following chapter:

- Chapter 12, "NLS and Bidirectional Support"

12

NLS and Bidirectional Support

When you design reports to be deployed to different countries, you must consider such things as character sets and text reading order. Oracle9*i*AS Reports Services includes the support you need to address any issues related to these considerations: National Language Support (NLS) for character sets and bidirectional support for text reading order.

Oracle NLS makes it possible to design applications that can be deployed in several different languages. Oracle supports most European, Middle Eastern, and Asian languages. NLS enables you to:

- Use international character sets (including multibyte character sets)
- Display data according to the appropriate language and territory conventions
- Extract strings that appear in your interface and translate them

Bidirectional support enables you to display data in either a left-to-right or right-to-left orientation, depending on the requirements of your audience.

This chapter provides a look at NLS architecture, including NLS settings relevant to Reports; explains how to specify character sets in a JSP; and offers information on bidirectional, Unicode, and translation support available through Oracle9*i*. It includes the following main sections:

- NLS Architecture
- NLS Environment Variables
- Specifying a Character Set in a JSP or XML File
- Bidirectional Support
- Unicode
- Translating Applications

12.1 NLS Architecture

Oracle NLS architecture consists of two parts:

- Language-Independent Functions
- Language-Dependent Data

12.1.1 Language-Independent Functions

Language-independent functions handle manipulation of data in an appropriate manner, depending on the language and territory of the runtime operator. Data is automatically formatted according to local date and time conventions.

12.1.2 Language-Dependent Data

With language-dependent data, you can isolate your data. This enables your application to deal only with translating strings that are unique to your application.

Because the language-dependent data is separate from the code, the operation of NLS functions is governed by the data supplied at runtime. New languages can be added and language-specific application characteristics can be altered without requiring code changes. This architecture also enables language-dependent features to be specified for each session.

12.2 NLS Environment Variables

NLS environment variables are automatically set to default values during Oracle9*i*AS installation.

Note: On a given Oracle9*i*AS Reports Services host machine, you can specify only one language. If you are providing application server services to a multilingual audience, you must have a separate host machine for each language.

Table 12–1 lists and describes NLS-related environment variables that are particularly relevant to Oracle9*i*AS Reports Services.

Note: For more information on all NLS environment variables, see the *Oracle9i Globalization Support Guide* on the Oracle Technology Network (http://otn.oracle.com).

Table 12–1 Environment variables particularly related to Oracle9iAS Reports Services

Variable	Description
NLS_LANG	Relevant to Oracle9*i*AS Reports Services. The language settings used by Oracle9*i*AS Reports Services.
DEVELOPER_NLS_LANG	The language for the Oracle9*i*DS Reports Builder.
USER_NLS_LANG	The language for the Oracle9*i*AS Reports Runtime component.

12.2.1 NLS_LANG Environment Variable

The NLS_LANG environment variable specifies the language, territory, and character set settings to be used by Oracle9*i*AS Reports Services. Specifically:

- The language for messages displayed to the user
- The default format masks used for DATE and NUMBER data types
- The sorting sequence
- The character set

Note: This environment variable is set automatically when you install Oracle9*i*AS. Refer to Section 12.2.1.1, "Defining the NLS_LANG Environment Variable" for more information about changing the environment variable after installing Oracle9*i*AS.

The syntax for NLS_LANG is:

```
NLS_LANG=<language_territory>.<charset>
```

The values are defined as follows:

- `language`

Specifies the language and its conventions for displaying messages (including error messages) as well as day and month names. If language is not specified, then the value defaults to American.

- `territory`

  Specifies the territory and its conventions for default date format, decimal character used for numbers, currency symbol, and calculation of week and day numbers. If territory is not specified, then the value defaults to America.

- `charset`

  Specifies the character set in which data is displayed. This should be a character set that matches your language and platform. This argument also specifies the character set used for displaying messages.

Table 12–2 lists commonly used language, territory, and character values for NLS_LANG:

Table 12–2 Commonly used NLS_LANG values

Language	Language_Territory.Character Set
American	AMERICAN_AMERICA.US7ASCII
Arabic	ARABIC_UNITED ARAB EMIRATES.AR8ISO8859P6
Brazilian Portuguese	BRAZILIAN PORTUGUESE_BRAZIL.WE8DEC
Bulgarian	BULGARIAN_BULGARIA.CL8ISO8859P5
Canadian French	CANADIAN FRENCH_CANADA.WE8ISO8859P1
Catalan	CATALAN_CATALONIA.WE8ISO8859P1
Croatian	CROATIAN_CROATIA.EE8ISO8859P2
Czech	CZECH_CZECH REPUBLIC.WE8ISO8859P1
Danish	DANISH_DENMARK.WE8ISO8859P1
Dutch	DUTCH_THE NETHERLANDS.WE8ISO8859P1
Egyptian	ARABIC_UNITED ARAB EMIRATES.AR8ISO8859P6
English (American)	See American
English (United Kingdom)	ENGLISH_UNITED KINGDOM.WE8DEC
Estonian	ESTONIAN_ESTONIA.BLT8MSWIN1257
Finnish	FINNISH_FINLAND.WE8ISO8859P1

Table 12–2 Commonly used NLS_LANG values

Language	Language_Territory.Character Set
French	FRENCH_FRANCE.WE8ISO8859P1
German	GERMAN_GERMANY.WE8ISO8859P1
Greek	GREEK_GREECE.EL8ISO8859P7
Hebrew	HEBREW_ISRAEL.IW8ISO8859P8
Hungarian	HUNGARIAN_HUNGARY.EE8ISO8859P2
Icelandic	ICELANDIC_ICELAND.WE8ISO8859P1
Indonesian	INDONESIAN_INDONESIA.WE8ISO8859P1
Italian	ITALIAN_ITALY.WE8DEC
Japanese	JAPANESE_JAPAN.JA16EUC
Korean	KOREAN_KOREA.KO16KSC5601
Latin America Spanish	LATIN AMERICAN SPANISH_AMERICA.WE8DEC
Latvian	LATVIAN_LATVIA.NEE8ISO8859P4
Lithuanian	LITHUANIAN_LITHUANIA.NEE8ISO8859P4
Mexican Spanish (see also Spanish)	MEXICAN SPANISH_MEXICO.WE8DEC
Norwegian	NORWEGIAN_NORWAY.WE8DEC
Polish	POLISH_POLAND.EE8ISO8859P2
Portuguese	PORTUGUESE_PORTUGAL.WE8DEC
Romanian	ROMANIAN_ROMANIA.EE8ISO8859P2
Russian	RUSSIAN_CIS.RU8PC855
Simplified Chinese	SIMPLIFIED CHINISE_CHINA.ZHS16CGB231280
Slovak	SLOVAK_SLOVAKIA.EE8ISO8859P2
Spanish (see also Mexican Spanish)	SPANISH_SPAIN.WE8DEC
Swedish	SWEDISH_SWEDEN.WE8DEC
Thai	THAI_THAILAND.TH8TISASCII
Traditional Chinese	TRADITIONAL CHINESE_TAIWAN.ZHT32EUC

Table 12–2 Commonly used NLS_LANG values

Language	Language_Territory.Character Set
Turkish	TURKISH_TURKEY.WE8ISO8859P9
Ukrainian	UKRAINIAN_UKRAINE.CL8ISO8859P5
Vietnamese	VIETNAMESE_VIETNAM.VN8VN3

Your NLS_LANG setting should take into account regional differences between countries that use (basically) the same language. For example, if you want to run in French (as used in France), then you set the NLS_LANG environment variable:

```
NLS_LANG=FRENCH_FRANCE.WE8ISO8859P1
```

If you want to run in French, but this time as used in Switzerland, you would set the NLS_LANG environment variable:

```
NLS_LANG=FRENCH_SWITZERLAND.WE8ISO8859P1
```

12.2.1.1 Defining the NLS_LANG Environment Variable

You define the NLS_LANG environment variable in the same way you define other environment variables on your Windows or UNIX operating system.

12.2.1.1.1 Windows To define the NLS_LANG environment variable on Windows, do the following:

4. Open the Windows registry.

> **Note:** Back up your registry before you edit it.

5. Expand the **HKEY_LOCAL_MACHINE** node, then expand the **SOFTWARE** node.
6. Click **ORACLE** to display the Oracle environment variables in the right panel of the Registry Editor.
7. Double-click the NLS_LANG environment variable.
8. Type the new value for NLS_LANG in the **Value** data text box.
9. Click **OK**.

12.2.1.1.2 UNIX To define the NLS_LANG environment variable on the UNIX platform, set it in the shell script `rwrun.sh`, located in your *`ORACLE_HOME`*`/bin` directory.

12.2.1.2 Character Sets

The character set component of the NLS environment variables specifies the character set in which data is represented in your environment. When data is transferred from a system using one character set to a system using another character set, it is processed and displayed correctly on the second system, even though some characters might be represented by different binary values in the character sets.

12.2.1.2.1 Character Set Design Considerations If you are designing a multilingual application, or even a single-language application that runs with multiple character sets, you need to determine the character set most widely used at runtime and then generate with the NLS environment variable set to that particular character set.

If you design and generate an application in one character set and run it in another character set, performance can suffer. Furthermore, if the runtime character set does not contain all the characters in the generate character set, then question marks appear in place of the unrecognized characters.

Portable Document Format (PDF) supports multibyte character sets.

12.2.1.2.2 Font Aliasing on Windows Platforms There might be situations where you create an application with a specific font but find that a different font is being used when you run that application. You would most likely encounter this when using an English font (such as MS Sans Serif or Arial) in environments other than Western European. This occurs because Oracle9*i*AS Reports Services checks to see if the character set associated with the font matches the character set specified by the language environment variable. If the two do not match, Oracle9*i*AS Reports Services automatically substitutes the font with another font whose associated character set matches the character set specified by the language environment variable. This automatic substitution assures that the data being returned from the database gets displayed correctly in the application.

Note: If you enter local characters using an English font, then Windows does an implicit association with another font.

There might be cases, however, where you do not want this substitution to take place. You can avoid this substitution by mapping all desired fonts to the WE8ISO8859P1 character set in the font alias file (`uifont.ali`). For example, if you are unable to use the Arial font in your application, you can add the following line to your font alias file (located at *ORACLE_HOME*`\TOOLS\COMMON\`):

```
ARIAL.....=ARIAL.....WE8ISO8859P1
```

Each line in the `uifont.ali` file takes the following syntax:

```
<Face>.<Size>.<Style>.<Weight>.<Width>.<CharSet>=<Face>.<Size>.<Style>.<Weight>.
<Width>.<CharSet>
```

In this example, you're saying that any ARIAL font should be mapped to the same value, but with the WE8ISO8859P1 character set.

Refer to Section 12.2, "NLS Environment Variables" for more information about the language environment variables.

12.2.1.3 Language and Territory

While the character set ensures that the individual characters needed for each language are available, support for national conventions provides correct localized display of data items.

The specified language determines the default conventions for the following characteristics:

- Language for server messages
- Language for day and month names and their abbreviations (specified in the SQL functions `TO_CHAR` and `TO_DATE`)
- Symbol equivalents for AM, PM, AD, and BC
- Default sorting sequence for character data when `ORDER BY` is specified (`GROUP BY` uses a binary sort unless `ORDER BY` is specified)
- Writing direction (both right to left and left to right)
- Affirmative and negative response strings

For example, if the language is set to French, then the following messages in English are converted to French:

```
English:
ORA-00942: table or view does not exist
FRM-10043: Cannot open file.
```

```
French:
ORA-0092: table ou vue inexistante
FRM-10043: Ouverture de fichier impossible
```

The specified territory determines the conventions for the following default date and numeric formatting characteristics:

- Date format
- Decimal character and group separator
- Local currency symbol
- ISO currency symbol
- Week start day
- Credit and debit symbol
- ISO week flag
- List separator

For example, if the territory is set to France, then the numbers are formatted using a comma as the decimal character.

12.2.2 DEVELOPER_NLS_LANG and USER_NLS_LANG Environment Variables

If you must use two sets of resource and message files at the same time, then two other language environment variables are available. These can be used after Oracle9*i*AS installation is completed.

- DEVELOPER_NLS_LANG
- USER_NLS_LANG

The syntax for DEVELOPER_NLS_LANG and USER_NLS_LANG is the same as for the NLS_LANG environment variable. That is:

```
DEVELOPER_NLS_LANG=<language_territory>.<charset>
USER_NLS_LANG=<language_territory>.<charset>
```

Use these environment variables in lieu of the NLS_LANG environment variable in the following situations:

- You prefer to use the Reports Builder in English, but you are developing an application for another language. DEVELOPER_NLS_LANG and USER_NLS_

LANG environment variables allow you to use different language settings for the Reports Builder and Reports Runtime.

- You are creating an application to run in a language for which a local language version of Oracle9*i*DS Reports Builder is not currently available.

If these environment variables are not specifically set, then NLS_LANG default values will be used.

12.3 Specifying a Character Set in a JSP or XML File

If you are producing HTML with your JSP, then you may need to add a character set to your JSP file using the following syntax (this one specifies a Japanese character set):

```
<META http-equiv="Content-Type" content="text/html;charset=shift_jis”>
```

Additionally, if you plan on outputting a report to XML, you may wish to include a character set in the XML Prolog Value property in the Builder's Property Inspector, following this syntax:

```
<?xml version=”1.0” encoding=”shift_jis” ?>
```

In both instances, the values expressed for the character set should call a character set that is compatible with the one specified for the host environment. The values for character sets used on the Web are different from the values expressed in the NLS_LANG environment variable. Table 12–3 lists commonly used values for the `charset` or `encoding` parameter:

Note: The values for `charset` and `encoding` are not case sensitive. You can enter them in lower- or uppercase.

Note: to set the character set in a .rdf file that you plan to use to generate HTML, you must ensure that the Before Report Escape property includes the following:

```
charset="text/html;charset=&encoding"
```

`&encoding` is then replaced at runtime with the appropriate setting.

Table 12–3 Valid values for a charset or encoding parameter

Language	Valid Character Set(s)
Afrikaans	iso-8859-1, windows-1252
Albanian	iso-8859-1, windows-1252
Arabic	iso-8859-6
Basque	iso-8859-1, windows-1252
Bulgarian	iso-8859-5
Byelorussian	iso-8859-5
Catalan	iso-8859-1, windows-1252
Croatian	iso-8859-2
Czech	iso-8859-2
Danish	iso-8859-1, windows-1252
Dutch	iso08859-1, windows-1252
English	iso-8859-1, windows-1252
Esperanto	iso-8859-3 (not widely supported in browsers)
Estonian	iso-8859-15
Faroese	iso-8859-1, windows-1252
Finnish	iso-8859-1, windows-1252
French	iso-8859-1, windows-1252
Galician	iso-8859-1, windows-1252
German	iso-8859-1, windows-1252
Greek	iso-8859-1
Hebrew	iso-8859-8
Hungarian	iso-8859-2
Icelandic	iso-8859-1, windows-1252
Inuit languages	iso-8859-10 (not widely supported in browsers)
Irish	iso-8859-1
Italian	iso-8859-1
Japanese	shift_jis, iso-2202-jp, euc-jp

Table 12–3 Valid values for a charset or encoding parameter

Language	Valid Character Set(s)
Korean	euc-kr
Lapp	iso-8859-10 (not widely supported in browsers)
Latvian	iso-8859-13, windows-1257
Lithuanian	iso-8859-13, windows-1257
Macedonian	iso-8859-5
Maltese	iso-8859-3 (not widely supported in browsers)
Norwegian	iso-8859-1, windows-1252
Polish	iso-8859-2
Portuguese	iso-8859-1, windows-1252
Romanian	iso-8859-2
Russian	koi-8-r, iso-8859-5
Scottish	iso-8859-1, windows-1252
Serbian	iso-8859-5
Slovak	iso-8859-2
Slovenian	iso-8859-2
Spanish	iso-8859-1, windows-1252
Swedish	iso-8859-1, windows-1252
Turkish	iso-8859-9, windows-1254
Ukrainian	iso-8859-5

12.4 Bidirectional Support

Bidirectional support enables you to design applications in Middle Eastern and North African languages whose natural writing direction is right to left. Bidirectional support enables you to control:

- Layout direction, which includes displaying items with labels at the right of the item and correct placement of check boxes and radio buttons
- Reading order, which includes text direction (e.g., right to left or left to right)

- Alignment, which includes switching point-of-origin from upper left to upper right
- Initial keyboard state, which controls whether local or Roman characters are produced automatically when the user begins data entry in forms (the end user can override this setting)

When you are designing bidirectional applications, you might want to use the NLS environment variables DEVELOPER_NLS_LANG and USER_NLS_LANG rather than NLS_LANG. For example, if you want to use an American interface while developing an Arabic application in a Windows environment, then set these environment variables as follows:

```
DEVELOPER_NLS_LANG=AMERICAN_AMERICA.AR8MSWIN1256
USER_NLS_LANG=ARABIC_UNITED ARAB EMIRATES.AR8MSWIN1256
```

Note that, in this example, the DEVELOPER_NLS_LANG environment variable uses an Arabic character set. Refer to "NLS Environment Variables" for more information about environment variables.

12.5 Unicode

Unicode is a global character set that allows multilingual text to be displayed in a single application. This enables multinational corporations to develop a single multilingual application and deploy it worldwide.

Global markets require a character set that:

- Allows a single implementation of a product for all languages, yet is simple enough to be implemented everywhere
- Contains all major living scripts
- Supports multilingual users and organizations
- Enables worldwide interchange of data via the Internet

12.5.1 Unicode Support

Oracle9*i*AS Reports Services provides Unicode support. If you use Unicode, you are able to display multiple languages, both single-byte languages such as Western Europe, Eastern Europe, Bidirectional Middle Eastern, and multibyte Asian languages such as Chinese, Japanese, and Korean (CJK) in the same application.

Use of a single character set that encompasses all languages eliminates the need to have various character sets for various languages. For example, to display a multibyte language such as Japanese, the NLS_LANG environment variable must be set to the following:

```
NLS_LANG=JAPAN_JAPANESE.JA16SJIS
```

To display a single-byte language such as German, NLS_LANG must be set to the following:

```
NLS_LANG=GERMAN_GERMANY.WE8ISO8859P1
```

The obvious disadvantage of this scheme is that applications can only display characters from one character set at a time. Mixed character set data is not possible.

With the Unicode character set, you can set the character set portion of NLS_LANG to UTF8 instead of a specific language character set. This allows characters from different languages and character sets to be displayed simultaneously. For example, to display Japanese and German together on the screen, the NLS_LANG variable must be set to one of the following:

```
NLS_LANG=JAPAN_JAPANESE.UTF8
```

```
NLS_LANG=GERMAN_GERMANY.UTF8
```

Unicode capability gives the application developer and end user the ability to display multilingual text in a form. This includes text from a database containing Unicode, multilingual text, text in graphical user interface (GUI) objects (for example, button labels), text input from the keyboard, and text from the clipboard. Oracle9*i*AS Reports Services currently supports Unicode on Windows.

Note: If you develop applications for the Web, then you can use Unicode because of the Unicode support provided by Java through the browser.

12.5.2 Unicode Font Support

Oracle9*i*AS Reports Services relies on the operating system for the font and input method for different languages. To enter and display text in a particular language, you must be running a version of the operating system that supports that language. Font support is limited but not restricted to the operating system font.

Windows NT release 4.0 provides True Type Big Fonts. These fonts contain all the characters necessary to display or print multilingual text. If you try to type, display,

or print multilingual text and see unexpected characters, then you are probably not using a Big Font. Big Fonts provided by Microsoft under Windows NT release 4.0 are as follows:

- Arial
- Courier New
- Lucida Console
- Lucida Sans Unicode
- Times New Roman

Third-party Unicode fonts are also available.

12.5.3 Enabling Unicode Support

To enable Unicode support, set the NLS_LANG environment variable as follows:

```
NLS_LANG=<language_territory>.UTF8
```

Refer to "NLS Environment Variables" for more information about environment variables.

12.5.4 Using ALTER SESSION

You can use the SQL command ALTER SESSION to override some NLS defaults, if you are connected to an Oracle database via the USERID keyword or have connected via the Connect dialog in the Reports Builder. For example, suppose you are building a report against an Oracle database that will publish data to different geographic locations. You might want to change the currency symbol, thousands grouping, and decimal indicator that the Oracle database uses when it formats a currency field depending on a user parameter. You could accomplish this task in several ways, but one method is to alter the Oracle database session NLS_LANG variable in a Before Report trigger.

Note: ALTER SESSION does not apply to pluggable data source queries (for example, JDBC and XML).

Note: For more information on the ALTER SESSION command, see the *Oracle9i SQL Reference*, available on the Oracle Technology Network (http://otn.oracle.com).

12.6 Translating Applications

In any Oracle9*i*AS Reports Services application, you see many types of messages, including:

- Error messages from the database
- Runtime error messages produced by Oracle9*i*AS Reports Services
- Messages and boilerplate text defined as part of the application

If the NLS environment variable is set correctly and the appropriate message files are available, then translation of messages for the first two items is done for you. To translate messages and boilerplate text defined as part of the application, you can use the Oracle translation tool, TranslationHub, and you might also find it useful to use PL/SQL Libraries for strings of code.

Note: You'll find information about using TranslationHub on the Oracle9*i*DS documentation CD and on the Oracle Technology Network (http://otn.oracle.com).

Manual translation is required for constant text within a PL/SQL block because that text is not clearly delimited, but is often built up from variables and pieces of strings. To translate these strings, you can use PL/SQL libraries to implement a flexible message structure.

You can use attachable PL/SQL libraries to implement a flexible message function for messages that are displayed programmatically by the built-in routine SRW.MESSAGE, or by assigning a message to a display item from a trigger or procedure. The library can be stored on the host and dynamically attached at runtime. At runtime, based on a search path, you can pull in the attached library. For example, a library might hold only the Italian messages:

```
FUNCTION nls_appl_mesg(inexe_no NUMBER)
RETURN CHAR
IS
   msg CHAR(80):
BEGIN
   IF      index_no = 1001 THEN
      msg := 'L' 'impiegato che Voi cercate non esiste...';
   ELSEIF  index_no = 1002 THEN
      msg := 'Lo stipendio non puo essere minore di zero.':
   ELSEIF  ...
   .
```

```
   .
   ELSE
      msg := 'ERRORE: Indice messaggio inesistente.';
   END IF;
   RETURN msg;
END;
```

A routine like this could be used anywhere a character expression would normally be valid. For example, to display text with the appropriately translated application message, you might include the following code:

```
Change_Alert_Message('My_Error_Alert', nls_appl_mesg(1001));
n := Show_Alert('My_Error_Alert');
```

To change the application to another language, simply replace the PL/SQL library containing the `nls_appl_mesg` function with a library of the same name containing the `nls_appl_mesg` function with translated text.

Part IV

Performance

Part IV provides information on managing, monitoring, and tuning your Oracle9*i*AS Reports Services environment. It includes the following chapters:

- Chapter 13, "Managing and Monitoring Oracle9iAS Reports Services"
- Chapter 14, "Tuning Oracle9iAS Reports Services"

13

Managing and Monitoring Oracle9*i*AS Reports Services

Oracle Enterprise Manager (OEM), included with Oracle9*i*AS, provides managing and monitoring services to Oracle9*i*AS Reports Services.

Use OEM to:

- Start, stop, and restart Reports Servers
- View and manage the Reports job queues (scheduled, current, failed, and finished)
- Monitor server performance
- View and change Reports Server configuration files
- View and link to all members of a server cluster

> **Note:** For more information on Oracle Enterprise Manager, see *Oracle9i Application Server Administrator's Guide*, available on the Oracle9*i*AS documentation CD.

Oracle9*i*AS installation automatically identifies Reports Servers and registers them with OEM. All you do is start OEM and start managing. If you add Reports Servers to your environment after you have installed Oracle9*i*AS, you must manually add the new Reports Server(s) to OEM's `targets.xml` file.

This chapter describes the managing and monitoring capabilities of OEM as they relate to Oracle9*i*AS Reports Services and tells you how to add a Reports Server to OEM's `targets.xml` file. It includes the following main sections:

- Navigating to Reports Services Information in OEM

- Starting, Stopping, and Restarting Reports Servers
- Viewing and Managing Reports Job Queues
- Monitoring Server Performance
- Viewing and Changing Reports Server Configuration Files
- Viewing and Linking to Server Cluster Members
- Adding a Reports Server to OEM

Note: The Reports Server pages in OEM include context sensitive online help topics that offer information about the items that appear on each page. In OEM, click the **Help** link to display help.

13.1 Navigating to Reports Services Information in OEM

To navigate to Reports Services Information in OEM:

Note: Before you launch the Enterprise Manager Console, you must first install and configure a management server. For instructions on how to do this, see Chapter 3 of the *Oracle Enterprise Manager Configuration Guide*, available on the Oracle9*i*AS documentation CD and on the Oracle Technology Network (http://otn.oracle.com).

---

1. Launch the Enterprise Manager Console. For more information on how to launch the console, refer to your Enterprise Manager documentation.
2. In the Console, click the **Oracle9*i*AS** node.
3. On the Oracle9*i*AS home page, click a Reports Server node to open its OEM main page.

13.2 Starting, Stopping, and Restarting Reports Servers

Once a Reports Server is registered in OEM, you can go through OEM to stop, start, and restart a given server.

Note: Reports Servers are automatically registered with OEM during installation of Oracle9*i*AS. If you add any Reports Servers after installing Oracle9*i*AS, you must register the new server(s) manually in OEM's `targets.xml` file. For more information, see Section 13.7, "Adding a Reports Server to OEM".

To start, stop, or restart a Reports Server:

1. In OEM, navigate to the Reports Server you want to manage.
2. On the Reports Server's main OEM page:
   - Click the **Start** button to start the server.
   - Click the **Stop** button to stop the server.
   - Click the **Restart** button to restart the server.

   These buttons appear on a Reports Server's main OEM page according to the server's current state:
   - When the server is down, the **Start** and **Stop** buttons display.
   - When the server is up, the **Restart** and **Stop** buttons display.

13.3 Viewing and Managing Reports Job Queues

OEM provides a page for viewing and managing Reports job queues. Each queue, Current, Scheduled, Failed, and Finished Jobs, has its own page. With some of the information, you can drill down to a greater level of detail. For example, on the Finished Jobs page, you can drill down to trace information or a Web view of report output (obtained from the Reports Server cache).

The following sections describe:

- Viewing and Managing the Current Jobs Queue
- Viewing and Managing the Scheduled Jobs Queue
- Viewing and Managing the Finished Jobs Queue
- Viewing and Managing the Failed Jobs Queue

13.3.1 Viewing and Managing the Current Jobs Queue

The Reports Current Jobs queue lists all jobs currently running on a particular Reports Server.

Use OEM for:

- Viewing a Report Server's Current Jobs Queue
- Cancelling a Current Job

13.3.1.1 Viewing a Report Server's Current Jobs Queue

To view a Current Jobs Queue:

1. In OEM, navigate to the Reports Server you want to manage.
2. On the Reports Server's main page, scroll down to the **Response and Load** section and click the number next to **Current Jobs**.

   If there are no current jobs in the Current Jobs Queue, there will be no link, and you will not be able to view the empty queue.

Table 13–1 lists and describes information provided in the Current Jobs Queue.

Table 13–1 Information provided in the Current Jobs Queue

Item	Description
Select	Use this radio button to select a particular job. On the Current Jobs Queue page, this function is most useful when you wish to cancel a job. Click the **Select** radio button next to a job you wish to cancel, then click the **Cancel** button near the top of the page.
Id	This displays a unique job identifier assigned to this job by the Reports Server. This number is strictly under the server's control and cannot be reset by a user.
Job Name	If you specified a job name in the command line you used to run this job, that name is listed here; otherwise, it is the name of the job provided for the "report=" or "module=" parameter of the job request.
Owner	This displays the user ID under which this job is running.
Output Type	Lists the destination type (destype) specified for this job at runtime.
Output Format	Lists the output format (desformat) specified for this job at runtime.

Table 13–1 Information provided in the Current Jobs Queue

Item	Description
Queued At	Lists the date and time this request was placed in the job queue.
Started At	Lists the data and time this job started running.

13.3.1.2 Cancelling a Current Job

To cancel a current job:

1. On the Current Jobs Queue page, click the **Select** radio button next to the job you want to cancel.
2. Click the **Cancel Job** button.

   This button does not display in OEM if there are no currently running jobs.

If you wish to rerun the job, you can do so from the Finished Jobs queue.

13.3.2 Viewing and Managing the Scheduled Jobs Queue

The Reports Scheduled Jobs Queue lists all jobs scheduled to run on a particular Reports Server.

Use OEM for:

- Viewing a Report Server's Scheduled Jobs Queue
- Cancelling a Scheduled Job

13.3.2.1 Viewing a Report Server's Scheduled Jobs Queue

To view a Scheduled Jobs Queue:

1. In OEM, navigate to the Reports Server you want to manage.
2. On the Reports Server's main page, scroll down to the **Response and Load** section and click the number next to **Scheduled Jobs**.

   If there are no scheduled jobs in the Scheduled Jobs Queue, there will be no link, and you will not be able to view the empty queue.

Table 13–2 lists and describes information provided in the Scheduled Jobs Queue.

Table 13–2 Information provided in the Scheduled Jobs Queue

Item	Description
Select	Use this radio button to select a particular job. On the Scheduled Job Queue page, this function is most useful when you wish to cancel a job. Click the **Select** radio button next to a job you wish to cancel, then click the **Cancel** button near the top of the page.
Id	This displays a unique job identifier assigned to this job by the Reports Server. This number is strictly under the server's control and cannot be reset by a user.
Job Name	If you specified a job name in the command line you used to run this job, that name is listed here; otherwise, it is the name of the job provided for the "report=" or "module=" parameter of the job request.
Owner	This displays the user ID under which this job is scheduled to run.
Output Type	Lists the destination type (destype) specified for this job.
Output Format	Lists the output format (desformat) specified for this job.
Queued At	Lists the date and time this request was placed in the job queue.
Interval	The frequency at which the job will be run, for example, daily, monthly, and so on.

13.3.2.2 Cancelling a Scheduled Job

To cancel a scheduled job:

1. On the Scheduled Job Queue page, click the **Select** radio button next to the job you want to cancel.
2. Click the **Cancel Job** button.

   This button does not display in OEM if there are no scheduled jobs.

If you wish to rerun the job, you can do so from the Finished Jobs queue.

13.3.3 Viewing and Managing the Finished Jobs Queue

The Reports Finished Jobs queue lists all successfully completed jobs on a particular Reports Server.

Use OEM for:

- Viewing a Report Server's Finished Jobs Queue
- Viewing a Job's Trace File
- Viewing a Result from Cache
- Rerunning a Finished Job

13.3.3.1 Viewing a Report Server's Finished Jobs Queue

To view a Finished Jobs Queue:

1. In OEM, navigate to the Reports Server you want to manage.
2. On the Reports Server's main page, scroll down to the **Response and Load** section and click the number next to **Finished Jobs**.

   If there are no finished jobs in the Finished Jobs Queue, there will be no link, and you will not be able to view the empty queue.

Table 13–3 lists and describes information provided in the Finished Jobs Queue.

Table 13–3 Information provided in the Finished Jobs Queue

Item	Description
Select	Use this radio button to select a particular job. On the Finished Job Queue page, this function is most useful for selecting a job and: ▪ Viewing its output Click the **Select** radio button next to a job you want to view, then click the **View Result** button near the top of the page. ▪ Viewing its trace results, provided that you included a trace command in the runtime command. Click the **Select** radio button next to a job with trace results you want to view, then click the **View Trace** button near the top of the page. ▪ Rerunning it Click the **Select** radio button next to a job you want to rerun, then click the **Rerun Report** button near the top of the page.

Table 13–3 Information provided in the Finished Jobs Queue

Item	Description
Id	This displays a unique job identifier assigned to this job by the Reports Server. This number is strictly under the server's control and cannot be reset by a user.
Job Name	If you specified a job name in the command line you used to run this job, that name is listed here; otherwise, it is the name of the job provided for the "report=" or "module=" parameter of the job request.
Owner	This displays the user ID under which this job was run.
Output Type	Lists the destination type (destype) specified for this job.
Output Format	Lists the output format (desformat) specified for this job.
Queued At	Lists the date and time this request was placed in the job queue.
Started At	Lists the date and time this job started running.
Finished At	Displays the date and time this job completed.
Status	Displays the finished status of the job. In the Finished Jobs Queue, status is always Finished Successfully.

13.3.3.2 Viewing a Job's Trace File

To view a job's trace file:

1. In the **Select** column on a Finished Jobs Queue page, click the radio button next to the finished job whose trace file you want to view.
2. Click the **View Trace** button near the top of the page.

13.3.3.3 Viewing a Result from Cache

To view a job result from the Report Server cache:

1. In the **Select** column on a Finished Jobs Queue page, click the radio button next to the finished job you want to view.
2. Click the **View Result** button near the top of the page.

   The result opens in a second browser window.

13.3.3.4 Rerunning a Finished Job

To rerun a job:

1. In the **Select** column on a Finished Jobs Queue page, click the radio button next to the finished job you want to rerun.
2. Click the **Rerun Report** button near the top of the page.

13.3.4 Viewing and Managing the Failed Jobs Queue

The Reports Failed Jobs queue lists all jobs that were cancelled or terminated with error on a particular Reports Server.

Use OEM for:

- Viewing a Report Server's Failed Jobs Queue
- Viewing a Failed Job's Trace File
- Rerunning a Failed Job

13.3.4.1 Viewing a Report Server's Failed Jobs Queue

To view a Failed Jobs Queue:

1. In OEM, navigate to the Reports Server you want to manage.
2. On the Reports Server's main page, scroll down to the **Response and Load** section and click the number next to **Failed Jobs**.

   If there are no failed jobs in the Failed Jobs Queue, there will be no link, and you will not be able to view the empty queue.

Table 13–4 lists and describes information provided in the Failed Jobs Queue.

Table 13–4 Information provided in the Failed Jobs Queue

Item	Description
Select	Use this radio button to select a particular job. On the Failed Jobs Queue page, this function is most useful for selecting a job and: ▪ Viewing its trace results, provided that you included a trace command in the runtime command. Click the **Select** radio button next to a job with trace results you want to view, then click the **View Trace** button near the top of the page. ▪ Rerunning it Click the **Select** radio button next to a job you want to rerun, then click the **Rerun Report** button near the top of the page.
Id	This displays a unique job identifier assigned to this job by the Reports Server. This number is strictly under the server's control and cannot be reset by a user.
Job Name	If you specified a job name in the command line you used to run this job, that name is listed here; otherwise, it is the name of the job provided for the "report=" or "module=" parameter of the job request.
Owner	This displays the user ID under which this job was run.
Output Type	Lists the destination type (destype) specified for this job.
Output Format	Lists the output format (desformat) specified for this job.
Queued At	Lists the date and time this request was placed in the job queue.
Started At	Lists the date and time this job started running.
Finished At	Displays the date and time this job was cancelled or terminated with error.
Status	Displays the finished status of the job. It informs you whether a job was cancelled by the user or terminated with error. In instances where a job was terminated with error, a brief error message is provided to indicate the cause of termination.

13.3.4.2 Viewing a Failed Job's Trace File

To view a failed job's trace file:

1. In the **Select** column on a Failed Jobs Queue page, click the radio button next to the failed job whose trace file you want to view.
2. Click the **View Trace** button near the top of the page.

13.3.4.3 Rerunning a Failed Job

To rerun a failed job:

1. In the **Select** column on a Failed Jobs Queue page, click the radio button next to the failed job you want to rerun.
2. Click the **Rerun Report** button near the top of the page.

13.4 Monitoring Server Performance

Each Reports Server registered in OEM has its own home page that summarizes general information about the server's status and performance. The sections that provide information about performance are Reports Server, Performance, and Administration:

- The **Reports Server** section is subdivided into General, Configuration, Status, and Response and Load:
  - **General** provides information on the Reports Server's installed version number, whether the server is up or down, and, if it's up, the date and time the server was started. It includes buttons for stopping, starting, and restarting the server. Which buttons show depends on whether the server is up or down.
  - **Configuration** provides the selected Reports Server's cluster name, if it is a member of a cluster; the trace option and mode specified in its configuration file; and the maximum number of jobs that can be held at one time in the Reports Server queue (this includes the total of the current, scheduled, and finished and failed jobs queues). All of these are configurable values.
  - **Status** lists the current number of active engines on the selected Reports Server, the amount of the host machine's CPU and RAM the selected server is currently using, and the average number of milliseconds is takes for the selected Reports Server to process a request from the client.
  - **Response and Load** provides information about the number of current, failed, finished, and scheduled jobs.

- The **Performance** section provides links to detailed performance information. Each link takes you to a different section of the same Performance page:
  - **Response Metrics** provides details about average response time; scheduled, finished, current, and failed jobs in the Job Queue; and the number of jobs transferred from one server to another in a clustered environment.
  - **Engine Information** lists the types and numbers of currently running engines on the selected Reports Server.
  - **System Usage Metrics** provides the percentage of CPU and number of megabytes of RAM currently being used by the selected Reports Server.
- The **Administration** section provides links to detailed information about the selected Reports Server's current configuration properties and views of the server's trace and log files:
  - **Server Configuration** leads to an editable view of the selected Reports Server's configuration file. Here you can alter the file, check file syntax, and save your changes. Changes take effect after the next server restart. (For more information, see Section 13.5, "Viewing and Changing Reports Server Configuration Files". See also Chapter 3, "Configuring Oracle9iAS Reports Services".
  - **Server Trace** leads to the results of any trace you ran on the selected Reports Server. Specify whether you will use the Trace option in the Reports Server's configuration file, available through the **Server Configuration** link.
  - **Server Log** leads to a log of general server events, such as when the selected Reports Server was started and stopped.

13.5 Viewing and Changing Reports Server Configuration Files

To view and change a Reports Server's configuration file through OEM:

1. On a selected Reports Server's home page, click the **Server Configuration** link under the **Administration** heading.
2. Make your changes in the display window.
3. Click the **Check Syntax** button to check your XML syntax.

> **Note:** Clicking this button does not validate the values you enter for configuration elements. For example, if an element requires that you specify a directory path, syntax checking does not validate the accuracy of your path. It just validates the XML syntax.

4. Click the **Save Changes** button to save your changes.

   Changes take effect after the next server startup or restart.

> **Note:** For detailed information on the attributes and values in the Reports Server configuration file, see Chapter 3, "Configuring Oracle9iAS Reports Services".

13.6 Viewing and Linking to Server Cluster Members

When you cluster Reports Servers together, it's reflected on each cluster member's home page in OEM under the **Other Servers Running in This Cluster** heading. Each listed cluster member links to the home page for that member.

Table 13–5 lists and describes the information the **Other Servers ...** section provides for each cluster member:

Table 13–5 Information under the Other Servers Running in this Cluster Heading

Row	Description
Server Name	Lists the names of each of the other Reports Servers that are members of the same cluster that the selected Reports Server belongs to. Click a server name to hyperlink to the cluster member's home page in OEM.
Finished Jobs	Provides the total number of finished jobs currently in the listed Reports Server's Job Queue.
Current Jobs	Provides the total number of currently running jobs in the listed Reports Server's Job Queue.
Scheduled Jobs	Provides the total number of scheduled jobs currently in the listed Reports Server's Job Queue.
Failed Jobs	Provides the total number of jobs for the listed Reports Server that were stopped before completion. This includes jobs that were user-terminated or terminated with error.

Table 13–5 Information under the Other Servers Running in this Cluster Heading

Row	Description
Average Response Time	Lists the average number of milliseconds it takes for the listed Reports Server to process a request from the client.

13.7 Adding a Reports Server to OEM

During Oracle9*i*AS installation, Reports Servers are automatically registered with OEM. If you add a Reports Server after you have installed Oracle9*i*AS, you must manually register the server with OEM if you want OEM to manage it. This section describes how to edit the `targets.xml` file that contains registration information for OEM targets.

> **Note:** A *target* in OEM is a component that OEM manages. For example, a Reports Server that is registered in OEM is considered an OEM target.

To register a target with OEM:

1. Open the `targets.xml` file in your XML editor of choice.

   You'll find `targets.xml` in the following directory path on both Windows and UNIX:

   ORACLE_HOME\sysman\emd\targets.xml

2. Using the following syntax, enter information for the Reports Server you are adding:

```
<target type="oracle_repserv" name="Reports_Server_name"
    <property name="password" value="password_value" encrypted="false"/>
    <property name="server" value="Reports_Server_name"/>
    <property name="servlet" value="http_URL_to_Reports_Servlet/rwservlet"/>
    <property name="userName" value="default_userid"/>
    <property name="oracleHome" value="ORACLE_HOME"/>
    <property name="host" value="domain_of_host_machine"/>
    <compositeMembership>
        <memberOf type="oracle_ias" name="ias-1" association="null"/>
    </compositeMembership>
</target>
```

   For example:

```
<target type="oracle_repserv" name="rep_disun1813"
    <property name="password" value="tiger" encrypted="false"/>
    <property name="server" value="rep_disun1813"/>
    <property name="servlet"
    value="http://machinename-pc2.us.oracle.com/servlet/rwservlet"/>
    <property name="userName" value="scott"/>
    <property name="oracleHome" value="d:\ora11"/>
    <property name="host" value="dlsun1813.us.oracle.com"/>
    <compositeMembership>
        <memberOf type="oracle_ias" name="ias-1" association="null"/>
    </compositeMembership>
</target>
```

Note: The user name and password are for the Reports Services administrator account. In a non-secure environment, the user name and password correspond to the <identifier> element in the Reports Server configuration file, <server_name>.conf. In a secure environment, they correspond to the Reports Services administrator account created in Oracle9*i*AS Portal or Oracle Internet Directory (OID), which belongs to the RW_ADMINISTRATOR group.

Initially, the `password` entry in the `targets.xml` file should be set to `encrypted="false"`. OEM will encrypt the value and reset `encrypted` to `"true"`.

---

3. Save the targets.xml file, and restart OEM to make your changes take effect.

14

Tuning Oracle9*i*AS Reports Services

Oracle9*i*AS Reports Services offers a number of fine-tuning options to adjust the level of service and performance it provides. This chapter discusses tuning considerations and offers you ways to optimize your server environment to fit the needs of your user base. It includes the following topics:

- Using the In-Process Server
- Tuning the Reports Engine
- Clustering Multiple Servers
- Optimizing Cache Strategies
- Monitoring Performance

14.1 Using the In-Process Server

The in-process server is a component of Oracle9*i*AS Reports Services that resides within the Reports Servlet (rwservlet). It requires less maintenance than a stand-alone server because, unlike the stand-alone server, it starts automatically whenever it receives the first request from the client via the Reports Servlet or JSP-based report URL. Additionally, an in-process server cuts down on the communication between processes, increasing the potential for faster performance.

To indicate that you want to run the Reports Server within the same process as the Reports Servlet, use the SERVER_IN_PROCESS parameter in the Reports Servlet configuration file (`rwservlet.properties`).

It takes the following syntax:

```
SERVER_IN_PROCESS=yes
```

Enter *no* (the default value) if you do not want the Reports Server to run within the same process as the Reports Servlet.

The Reports Servlet configuration file is located in the following path on both Windows and UNIX:

ORACLE_HOME\reports\conf\rwservlet.properties

14.2 Tuning the Reports Engine

The Reports Engine has a number of configuration parameters that relate to performance tuning, including:

- initEngine
- maxEngine
- minEngine
- engLife
- maxIdle
- callBackTimeOut

These parameters are attributes of the *engine* element in the Reports Server configuration file, <*server_name*>.conf, located in the following path on both Windows and UNIX:

ORACLE_HOME\reports\conf\<*server_name*>.conf

> **Note:** You'll find detailed information about the Reports Server configuration file in Chapter 3, "Configuring Oracle9iAS Reports Services".

The *engine* element takes the following syntax:

```
<engine id="rwEng" class="oracle.reports.engine.EngineImpl" initEngine="1"
maxEngine="5" minEngine="1" engLife="50" maxIdle="15" callbackTimeOut="10000">
    <property name="sourceDir" value="D:\orawin\reports\server\cache"/>
    <property name="tempDir" value="D:\orawin\reports\server\temp"/>
</engine>
```

In the following subsections, each of the relevant attributes of the *engine* element is discussed separately. But another part of this story is how they work together. For example, the *initEngine* attribute works together with *maxEngine* and *minEngine* to

set the number of Reports Engines that are always available to users, provided the server is up. These are also affected by *engLife*, which specifies how many jobs an engine will process before it shuts down. Additionally, to prevent engines from starting up then shutting down in high-volume situations, you may try setting a relatively high value for *engLife*.

Finally, there is no optimal setting for any of these attributes—the best settings differ based on factors such as how heavy your request load is and whether the request load is constant or fluctuates throughout the day.

The following subsections examine the benefits of high and low attribute values in conjunction with constant and variable request loads and how response time may be affected.

14.2.1 initEngine

The *initEngine* attribute defines the number of engines you want the Reports Server to start at initialization. Its default value is 1. The initEngine attribute is influential only when the server first comes up.

Within the Reports Server configuration file, *initEngine* takes the following syntax:

```
initEngine="1"
```

The more engines you initialize at server startup, the longer the Reports Server takes to start and the more memory it requires to sustain them. Higher numbers of user requests coupled with a constant request load will likely benefit from a higher *initEngine* value. The benefit to be gained from having engines at the ready should outweigh the potential disadvantage of higher memory use. On the other hand, if an engine is not available when a request comes in, time to start up the engine is added to the response overhead.

14.2.2 maxEngine

The *maxEngine* attribute specifies the maximum number of a given type of engine that can run on the server at the same time. If the number of requests exceeds the number specified for *maxEngine*, the spillover requests are sent to the Reports Queue and are run as engines become available. The default value is 1.

Within the Reports Server configuration file, *maxEngine* takes the following syntax:

```
maxEngine="4"
```

14.2.3 minEngine

The *minEngine* attribute specifies the minimum number of a given type of engine that should be kept running in spite of the *maxIdle* value (which would otherwise tell an engine to shut down after *x* number of idle minutes). The default value is 0.

Within the Reports Server configuration file, *minEngine* takes the following syntax:

```
minEngine="0"
```

In an environment with a low request volume or a high tolerance in the user base for less-than-optimal response times, the default value is a good way of preserving memory (by not tying it up with unnecessary engines). In a high request volume environment, where the request load is constant, a higher value should yield quicker response. This is because engines that are already running when requests come in save engine startup time.

Note: Make sure the value you enter for *minEngine* does not exceed the value you entered for *maxEngine* or *initEngine*. A *minEngine* value that is higher than a *maxEngine* or *initEngine* value will result in an error.

14.2.4 engLife

The *engLife* attribute specifies the number of jobs an engine can run before the engine is terminated, and, if necessary, a new engine is started. This feature is available to clean up memory structures. The default value is 50.

Within the Reports Server configuration file, *engLife* takes the following syntax:

```
engLife="120"
```

A low value shuts engines down sooner and consequently frees up memory. In this way, it also allows resources to be recycled. A high value keeps the engine alive for a longer period. This could be valuable in environments that sustain a high-volume of requests throughout the day and where memory use is of lesser concern.

14.2.5 maxIdle

The *maxIdle* attribute specifies the number of minutes of allowable idle time before an engine is shut down, provided the current number of engines is higher than the value specified for *minEngine*.

For example, if *minEngine* is 0, *maxIdle* is 30, and one engine has been running but unused for 30 minutes, that engine will shut down.

The default value is 30 (minutes).

Within the Reports Server configuration file, *maxIdle* takes the following syntax:

```
maxIdle="30"
```

When you specify this value, you must balance your desire to save engine startup time against your desire to prevent unnecessary use of memory (by idle engines).

14.2.6 callBackTimeOut

The *callBackTimeOut* attribute specifies the number of milliseconds of allowable waiting time between when the server calls an engine and the engine calls the server back. The default value is 60000 (milliseconds). If the *callBackTimeOut* value is exceeded before communication is established, an exception is raised, an error message is displayed, and the process is stopped.

Within the Reports Server configuration file, *callBackTimeOut* takes the following syntax:

```
callBackTimeOut="60000"
```

If the server host machine is very fast, you can reduce this number. If it is very slow, you may want to keep this number high to ensure there is sufficient time for the server and the engine to communicate. This value is dependent more on the capabilities of your hardware than on the requirements of your user base.

14.3 Clustering Multiple Servers

A cluster is a virtual grouping of servers into a community for the purpose of sharing request processing efficiently across members of the cluster. Clustering in Oracle9*i*AS Reports Services is peer-level, rather than master/slave. Peer-level clustering means that all members of the cluster take equal responsibility for sharing and processing incoming requests. If one member is shut down, the other members carry on managing the request load. There is no single-point-of-failure, where one machine's malfunction brings the whole system down.

The advantages of clustering are the increased processing power through the pooling of multiple CPUs on multiple machines, the fail-safe environment, and faster response times through request sharing.

Each cluster member machine must be configured in more or less the same way to allow a report to run on each server member in the same way. This means that configuration files should have most of the same settings: a distinction can be drawn between job-related settings and machine-related settings. Job-related settings must be the same from cluster member to cluster member. Job-related settings include settings related to security, data sources, and destination types. Machine-related settings include such attributes as maxEngine, minEngine, maxIdle, initEngine, and the like—these can be different from member to member.

Additionally, for cluster members:

- Server-related environment variables should be set to the same values.
- TNS settings should point to the same databases in the same way.

If you must set your servers up for different languages, you won't want to cluster them together. Additionally, if your machines require different job-related configuration settings, you will not benefit from clustering.

For servers to be members of the same cluster, they must share a cluster name (appended to each server's server name) and have the same public and private keys.

Note: You'll find more information about enrolling a server in a cluster in Chapter 3, "Configuring Oracle9iAS Reports Services".

14.4 Optimizing Cache Strategies

A cache is a temporary storage space for recently accessed data that can return data more rapidly than the data's original storage space. When data is fetched from cache, it saves time otherwise required to arrive at and search the original storage space, and it spares the network the burden of additional traffic.

In Oracle9*i*AS Reports Services you have three opportunities for getting the most out of your cache:

- In the Reports Server configuration file, where you set up the Reports Cache
- Via the command line, where you can specify the rules defining how long a particular report's output can be reused and when it should be removed from cache
- In the JSP file, where you use ojsp tags that make use of the JSP Web Object Cache

These opportunities are discussed in the following subsections:

- Setting Up Cache in the Reports Server Configuration File
- Specifying Cache-Related Options in the Command Line
- Setting Up Caching Options in a JSP

14.4.1 Setting Up Cache in the Reports Server Configuration File

The Reports Server configuration file (ORACLE_HOME\reports\conf\<*server_name*>.conf) contains a *cache* element for specifying the Java class that defines the server's cache as well as the cache size and location.

When you use the default Java class provided with the Reports Server, the *cache* element's syntax is:

```
<cache class="oracle.reports.cache.RWCache">
    <property name="cacheSize" value="50"/>
    <property name="cacheDir" value="ORACLE_HOME\reports\server\cache"/>
</cache>
```

Working in conjunction with the TOLERANCE and EXPIRATION command line options, the Reports Server cache speeds report request response times by storing the latest report results and serving them up to subsequent matching requests. Response is faster because a completed report can be fetched from the cache faster than fresh data can be fetched from the database and the display version of the report can be assembled.

The property relevant to tuning is *cacheSize*. For this value, you enter the number of megabytes of disk space you wish to dedicate to the Reports cache. The default value is 50 (megabytes).

The disk space set aside for cache is available only for caching and cannot be used for any other purpose. The Reports Server keeps track of and protects this space. Outside of restrictions that may have been imposed on a report via the EXPIRATION command, the Reports cache operates on a first-in-first-out principle (FIFO), where, when the cache is full, the oldest data in the cache is overwritten by incoming data.

Higher values allow for storage of more data, but might take longer to search. Lower values provide faster cache hits, but may not be adequate for the amount of data you want to keep on hand, so more duplicate requests are run. Another consideration is the amount of disk space you can reasonably remove from general use. The value you specify for *cacheSize* should balance these considerations and provide you with the optimal result for your hardware resources and the requirements of your user base.

14.4.2 Specifying Cache-Related Options in the Command Line

Oracle9*i*AS Reports Services provides two command line options for defining how long a particular report that is stored in cache can be used for similar requests (TOLERANCE) and how long a report can remain in cache before it is deleted (EXPIRATION). TOLERANCE and EXPIRATION are essential for effective use of cache because, if they are not defined, your opportunities for reusing cached report results is significantly diminished.

Note: If no TOLERANCE or EXPIRATION is specified at runtime (or if their values are 0), the only job output reused from cache is that which matches a request that is already in the Reports Current Job Queue.

14.4.2.1 TOLERANCE

Use TOLERANCE to set the maximum acceptable time for reusing a report's cached output when a duplicate job is detected. Setting the time tolerance on a report reduces the processing time when duplicate jobs are found.

You can use TOLERANCE with the rwclient, rwservlet, and rwcgi commands.

If TOLERANCE is not specified, then Oracle9*i*AS Reports Services reruns the report even if a duplicate report is found in the cache.

If a report is being processed (that is, it's in the current job queue) when an identical job is submitted, then Oracle9*i*AS Reports Services reuses the output of the currently running job even if TOLERANCE is not specified or is set to zero.

The syntax of the TOLERANCE command is:

```
TOLERANCE=[number {unit}] or [{Mon DD, YYYY} hh:mi:ss AM/PM {timezone}]
```

Number can be any whole value from 0 up. *Unit* can be minute(s), hour(s), or day(s). The default unit is minute(s). *Date* is optional. If the date is not specified, today's date is assumed. *Timezone* is also optional. If the time zone is not specified, the Report Server's time zone is assumed.

14.4.2.2 EXPIRATION

Use EXPIRATION to define how long the report output can exist in cache before it is deleted.

You can use EXPIRATION with the rwclient, rwservlet, and rwcgi commands.

If a report is in the current job queue when an identical job is submitted, then the server reuses the output of the currently running job even if EXPIRATION is not specified or is set to zero.

The syntax of the EXPIRATION command is:

```
EXPIRATION=[number {unit}] or [{Month DD, YYYY} hh:mi:ss am/pm {timezone}]
```

Number can be any whole value from 0 up. *Unit* can be minute(s), hour(s), or day(s). The default unit is minute(s). *Date* is optional. If the date is not specified, today's date is assumed. *Timezone* is also optional. If the time zone is not specified, the Report Server's time zone is assumed.

14.4.3 Setting Up Caching Options in a JSP

The Oracle9*i*AS Components for Java 2 Enterprise Edition (OC4J) Web Object Cache is a mechanism that allows Web applications written in Java to capture, store, reuse, post-process, and maintain the partial and intermediate results generated by a dynamic Web page, such as a JSP or servlet. For programming interfaces, it provides a tag library (for use in JSPs) and a Java API (for use in servlets).

This section introduces cache-related ojsp tags and points you to documentation that explains how to use these tags, including how to configure your environment to use them.

The Web Object Cache tag library is a convenient wrapper, using JSP custom tag functionality, for the Web Object Cache API. Use custom tags in a JSP to control caching. The API is called through the underlying tag handler classes.

Although we are referring to these tags as "ojsp" tags, which use the ojsp prefix, you can specify any desired prefix in the *taglib* directive in your JSP file.

You can have one or more of these tags in a given JSP file. The open and close tags surround the data (cache blocks) you want to cache. Multiple cache blocks are distinguished from one another through unique *name* attributes.

Table 14–1 lists and describes the cache tags available through OC4J.

Table 14–1 Cache tag descriptions

Tag	Description
ojsp:cache	This is for general, character-based caching of HTML or XML fragments.

Table 14–1 Cache tag descriptions

Tag	Description
`ojsp:cacheXMLObj`	This is for caching XML objects; its parameters comprise a superset of the `cache` tag parameters. The Web Object Cache is particularly useful when post-processing XML documents.
`ojsp:useCacheObj`	This is for general caching of Java serializable objects. Some of the semantics and syntax are patterned after the standard `jsp:useBean` tag.
`ojsp:cacheInclude`	This tag combines the functionality of the cache tags `cache` and `cacheXMLObj` and the standard JSP `include` tag.
`ojsp:invalidateCache`	Use this tag to explicitly invalidate a cache block through program logic. It is important not to confuse the `ojsp:invalidateCache` tag with the *invalidateCache* attribute of the other cache tags. The attribute is for more limited use—to invalidate the pre-existing cache object.

For detailed information on the OC4J Web Object Cache, see the Oracle Technology Network (http://otn.oracle.com). One useful source is *Oracle9i*AS *Containers for J2EE JSP Tag Libraries and Utilities Reference*. This includes tag syntax and examples as well as information for configuring your environment to use ojsp:cache tags.

14.5 Monitoring Performance

Oracle9*i*AS Reports Services provide a number of means for monitoring performance that enable you to identify areas that might benefit from additional tuning. The most frequently used means for monitoring server performance include:

- The Reports Services-related pages in Oracle Enterprise Manager, discussed in detail in Chapter 13, "Managing and Monitoring Oracle9iAS Reports Services"
- The Oracle Trace feature, available through the Reports Server configuration file, the Reports Servlet configuration file, the Reports Server pages in Oracle Enterprise Manager (OEM), and through trace-related command line options
- The `SHOWJOBS` command line option
- The RW_SERVER_QUEUE table, which provides another window (aside from that available through OEM) into the Reports Server job queues

Server performance monitoring is discussed in the following subsections:

- Monitoring Performance with Oracle Trace
- The SHOWJOBS Command Keyword
- Accessing the RW_SERVER_QUEUE table
- Updating the Database with Queue Activity

14.5.1 Monitoring Performance with Oracle Trace

Oracle Trace is a tool for collecting performance and resource utilization data, such as SQL Parse, Execute, Fetch statistics, and Wait statistics. Whether you are having problems, or simply want to assess server performance for potential tuning opportunities, you can use Oracle Trace to gather information about server performance and use it to inform any decisions you are making concerning your server configuration as well as to inform technical support specialists of recent server activity.

This section provides an overview of Trace, information on the places in Oracle9*i*AS Reports Services where tracing features are available, and points you to easily accessible sources of additional information.

Note: Oracle Trace is discussed fairly extensively in other sections of this manual. This section is a pointer to the locations of that information.

14.5.1.1 Trace Overview

When you generate a trace file via `rwbuilder` or `rwrun`, the resulting trace file contains traces of the whole builder session. Tracing options can be specified from the command line or the Reports Builder configuration file (*`ORACLE_HOME`*`\reports\conf\rwbuilder.conf`).

For example, from the command line, you can specify:

```
tracefile=trace.trc traceopts=trace_all tracemode=trace_replace
```

The trace file location is relative to the current working directory or is an absolute location, if specified as such.

In the Reports Builder configuration file, you can specify:

```
<trace traceFile="trace.trc" traceOpts="trace_all" traceMode="trace_replace"/>
```

The trace file location is relative to the server log directory (*ORACLE_HOME*\reports\logs) or is an absolute location, if specified as such. If the trace file is not specified, the default trace file name is used: <*hostname*>-rwbuilder.trc in the server logs directory.

Command line tracing options override those specified in the Reports Builder configuration file.

When you generate a trace file for the Reports Server, separate trace files are generated for the Reports Server and the Reports Engine(s). Specify server tracing options in the server configuration file (*ORACLE_HOME*\reports\conf\<*server_name*>.conf).

For example:

```
<trace traceFile="trace.trc" traceOpts="trace_all" traceMode="trace_replace"/>
```

With a server trace, trace file location is relative to the server logs directory. If a trace file name is not specified, the default server trace file name is used: <*server_name*>-<*engine_name*><*engine_number*>.trc.

With a servlet trace, tracing options can be specified from the rwservlet configuration file (*ORACLE_HOME*\reports\conf\rwservlet.properties). For example:

```
TRACEOPTS=TRACE_ALL
TRACEFILE=rwservlet.trc
TRACEMODE=TRACE_REPLACE
```

With a servlet trace, trace file location is relative to the server log directory (*ORACLE_HOME*\reports\logs).

14.5.1.2 Additional Sources of Trace Information

Oracle9*i*AS Reports Services offers a number of ways to set up a trace and view trace results:

- You can set default trace values in the Reports Server configuration file (*ORACLE_HOME*\reports\conf*server_name*.conf). This is discussed in Chapter 3, Section 3.2.1.11, "trace".
- If you wish to track and log runtime information on the Reports Servlet and JSPs, use the TRACEOPTS parameter in the servlet configuration file (*ORACLE_HOME*\reports\conf\rwservlet.properties). This is discussed Section 3.3.5, "Setting up Trace Options for the Reports Servlet and JSPs".

- Use Oracle Enterprise Manager to view trace results, either server-wide or for individual jobs. This is discussed in Chapter 13, "Managing and Monitoring Oracle9iAS Reports Services".
- Specify desired trace options for individual jobs in the runtime command line. This is discussed in Appendix A, starting with Section A.4.101, "TRACEOPTS".

Note: How to define a runtime command line via a URL is discussed in Chapter 8, "Running Report Requests".

14.5.2 The SHOWJOBS Command Keyword

The `showjobs` command keyword is useful for displaying a Web view of Reports Server queue status. Use it only with the `rwservlet` command. It uses the following syntax:

Reports_URL`/rwservlet/showjobs?server=`*server_name*`&statusformat=`*desired_format*

Table 14–2 lists and describes valid values for SHOWJOBS parameters.

Table 14–2 Valid values for* `SHOWJOBS` *parameters

Parameter	Valid Values	Description
server	*server_name*	The name of the Reports Server that processed the jobs you wish to view in the job queue. **Required**. However, if you specified a default server in the servlet configuration file (*ORACLE_HOME*`\reports\conf\rwservlet.properties`) you can omit this parameter, and `showjobs` will use the default.

Table 14–2 Valid values for `SHOWJOBS` *parameters*

Parameter	Valid Values	Description
statusformat	html|xml|xmldtd	**Default:** html ▪ Use `html` to view an HTML version of the Current, Past, and Scheduled Jobs queues in your browser. This provides the most visually accessible version of these queues. ▪ Use `xml` to view an XML version of the Current, Past, and Scheduled Jobs queues in your browser. This provides output useful for programmatic passing of job status information, which can be used by other applications. ▪ Use `xmldtd` to view an XML plus in-line data type dictionary (DTD) version of the jobs queues in your browser. This also provides output useful for programmatic passing of job status information, which can be used by other applications.

14.5.3 Accessing the RW_SERVER_QUEUE table

When you move to centralized reporting, your user base may require certain operational information. For example:

- Your users may request the status of a report they have submitted.
- Your administrators may request how many concurrent users there are on the Oracle9*i*AS Reports Server. This is useful for both sizing the environment and ensuring license compliance.

The Oracle9*i*AS Reports Server makes it possible to answer both of these questions by posting the current report queue to the database each time a job request is submitted. This information is inserted into a RW_SERVER_QUEUE table that includes the following data:

- The name of the job
- Who submitted it
- What output format was chosen
- The job's current status
- When it was queued, started, and subsequently finished

Table 14–3 lists and describes the information contained in the RW_SERVER_ QUEUE table:

Table 14–3 Structure of the RW_SERVER QUEUE Table

Column Name	Description
JOB_TYPE	States whether the job listed is CURRENT, PAST, or SCHEDULED.
JOB_ID	System generated job identification number.
JOB_NAME	Job submission name (or file name if no value for JOBNAME is specified).
STATUS_CODE	Current status of job. See Table 14–4, " Job Submission Status Codes" for more information about status codes.
STATUS_MESSAGE	Full message text relating to status code (includes Oracle9*i*AS error messages if report is terminated).
COMMAND_LINE	Complete command line submitted for this job submission.
OWNER	User who submitted the job. On the Web, the default user is the OS user who owns the Web server.
DESTYPE	Format of the report output.
DESNAME	Name of the report output if not going to the Oracle9*i*AS Reports Server cache.
SERVER	Oracle9*i*AS Reports Server to which the report was submitted.
QUEUED	Date and time the job submission was received and queued by the given Oracle9*i*AS Reports Server.
STARTED	Date and time the job submission was run.
FINISHED	Date and time the submitted job completed.
LAST_RUN	Date and time a scheduled job was last run.
NEXT_RUN	Date and time a scheduled job will run.
REPEAT_INTERVAL	Frequency on which to run a job.
REPEAT_PATTERN	Repeat pattern (for example, every minute, every hour, or every day).
CACHE_HIT	Indicates whether the job result was fetched from cache instead of running itself.

Table 14–3 Structure of the RW_SERVER QUEUE Table

Column Name	Description
CACHE_KEY	Indicates the cache key used to compare a request with an already cached result. The key is a string that uniquely indicates a report output result without considering the time the job was run. For example, if two requests have the same key, it means they will both generate the same output if they are running at the same time, although the outputs may be used for different purposes (e.g., send to e-mail or save to a file).

Table 14–4 Job Submission Status Codes

Status Code	Defined PL/SQL Constant	Description for Status Code
0	UNKNOWN	No such status
1	ENQUEUED	Job is waiting in queue
2	OPENING	Server is opening report definition
3	RUNNING	Report is currently running
4	FINISHED	Job submission has completed successfully.
5	TERMINATED_W_ERR	Job has ended with an error
6	CRASHED	Engine has crashed during execution of the job.
7	CANCELED	Job was canceled by user request
8	SERVER_SHUTDOWN	Job was canceled due the Oracle9*i*AS Reports Server shutting down.
9	WILL_RETRY	Job failed and is queued for RETRY
10	SENDING_OUTPUT	Job has completed and is returning output

Users can view this table if you grant them SELECT access. This will enable them to query the job submission of interest and determine the job's current status. You can also give them a view of this data by implementing an Oracle9*i*AS Reports Server Queue screen. You can implement such a screen by creating a report based directly on this table. Doing so displays the queue report as a job submission by the user.

Conversely, the real-time update of the table with the status of job submissions makes it very easy for administrators to know exactly how many concurrent users have requested jobs to be run on the Reports Server.

By counting the number of entries in the RW_SERVER_QUEUE table that have a status code indicating that the job has been queued but not completed, it is possible to return an accurate number of the current active users on the server. For example, you could use the following query:

```
SELECT Count(*)
FROM   RW_SERVER_QUEUE
WHERE  STATUS_CODE IN (1,     -- ENQUEUED
                       2,     -- OPENING
                       3)     -- RUNNING
AND    JOB_TYPE != 'Scheduled'
```

Note: While the table contains the date and time a report was queued, run, and finished, it is not a good idea to use a query based on the fact that a job has a defined QUEUED and STARTED time but no FINISHED value. If a report ends due to an unexpected error, such as invalid input, then the FINISHED column remains NULL. However, the STATUS_CODE and STATUS_MESSAGE both indicate there has been a failure and list the cause of that failure.

14.5.4 Updating the Database with Queue Activity

The Reports Server job queue is implemented through the use of a PL/SQL case API. It functions to update the queue table with the queue information as requests are made. This implementation is defined in the following path:

ORACLE_HOME\reports\admin\sql\RW_SERVER.SQL

To implement the queue, take the following required steps:

1. Load the `rw_server.sql` file to a database (this file is included with your Oracle9*i*AS Reports Services installation: *ORACLE_HOME*\reports\admin\sql).

   This creates a schema that owns the report queue information and has execute privileges on the server queue API.

2. Set the REPOSITORYCONN attribute of the *jobStatusRepository* element in the Oracle9*i*AS Reports Server configuration file (located in *ORACLE_HOME*\reports\conf*server_name*.conf) to the connection string of the schema that owns the queue data.

When the server starts, it connects as the defined user and logs job submissions.

Note: If the Oracle9*i*AS Reports Server and the Oracle database have been installed on a single, stand-alone Windows NT machine, then the definition of REPOSITORYCONN can prevent the automatic startup of the Oracle9*i*AS Reports Services as Windows NT boots up. Because the Oracle database service might not have been started, this prevents the Oracle9*i*AS Reports Server from performing the required login. Once the Oracle database has started, the Oracle9*i*AS Reports Server can be started manually.

Part V

Appendices

Part V contains appendices that provide additional, detailed information about functioning in the Oracle9*i*AS Reports Services environment. It includes information about Reports commands and their associated command line arguments as well as details about Reports-related environment variables.

Part V contains the following appendices:

- Appendix A, "Command Line Arguments"
- Appendix B, "Reports-Related Environment Variables"

A

Command Line Arguments

This appendix contains descriptions and examples of command line arguments that can be used with one or more of the following commands: `rwclient`, `rwrun`, `rwbuilder`, `rwconverter`, `rwservlet`, `rwcgi`, and `rwserver`. Each argument description includes a table that indicates which commands can use which argument keywords.

> **Note:** For examples of using command line arguments in your runtime URL, see Chapter 8, "Running Report Requests".

The following topics are discussed in this appendix:

- Command Overview
- Command Line Syntax
- General Usage Notes
- Command Line Arguments

A.1 Command Overview

This section provides a brief description of the commands whose keywords/arguments are discussed in this appendix, including:

- rwclient
- rwrun
- rwbuilder
- rwconverter

- rwservlet
- rwcgi
- rwserver

Each command description includes a list of the keywords that can be used with it. In the command line, you must use the keyword along with its argument. When you enter a command line, you can use the keywords in any order.

A.1.1 rwclient

The `rwclient` command parses and transfers a command line to the specified (or default) Reports Server.

Keywords used with rwclient

The brackets surrounding each keyword in this list are there to create a separation between keywords and has no other significance.

```
rwclient [ACCESSIBLE] [ARRAYSIZE] [AUTHID] [AUTOCOMMIT] [BCC] [BLANKPAGES]
[BUFFERS] [CACHELOB] [CC] [CELLWRAPPER] [CMDFILE] [COPIES] [CUSTOMIZE]
[DATEFORMATMASK] [DELIMITED_HDR] [DELIMITER] [DESFORMAT] [DESNAME] [DESTINATION]
[DESTYPE] [DISTRIBUTE] [EXPIRATION] [EXPRESS_SERVER] [FROM] [IGNOREMARGIN]
[JOBNAME] [JOBTYPE] [LONGCHUNK] [MODE] [MODULE|REPORT] [NONBLOCKSQL]
[NOTIFYFAILURE] [ONFAILURE] [ONSUCCESS] [ORIENTATION] [PAGESIZE] [PDFCOMP]
[PDFEMBED] [READONLY] [REPLYTO] [REPORT|MODULE] [ROLE] [RUNDEBUG] [SCHEDULE]
[SERVER] [SUBJECT] [TOLERANCE] [TRACEMODE] [TRACEOPTS] [USERID]
```

A.1.2 rwrun

The `rwrun` command runs a report using the Oracle9*i*AS Reports Services in-process server. When you run a .rep file, the PL/SQL is already compiled and will not be recompiled. If you are running an .rdf file, the PL/SQL is automatically recompiled, if necessary. It becomes necessary if the report wasn't compiled and saved from the Reports Builder or the platform or version on which you were running the report is incompatible with the platform on which it was last compiled and saved.

Keywords used with rwrun

The brackets surrounding each keyword in this list are there to create a separation between keywords and has no other significance.

```
rwrun [ACCESSIBLE] [ARRAYSIZE] [AUTHID] [AUTOCOMMIT] [BCC] [BLANKPAGES]
[BUFFERS] [CACHELOB] [CC] [CELLWRAPPER] [CMDFILE] [COPIES] [CUSTOMIZE]
```

[DATEFORMATMASK] [DELIMITED_HDR] [DELIMITER] [DESFORMAT] [DESNAME] [DESTINATION] [DESTYPE] [DISTRIBUTE] [EXPRESS_SERVER] [FROM] [IGNOREMARGIN] [LONGCHUNK] [MODE] [MODULE|REPORT] [NONBLOCKSQL] [NOTIFYFAILURE] [ONFAILURE] [ONSUCCESS] [ORIENTATION] [PAGESIZE] [PAGESTREAM] [PDFCOMP] [PDFEMBED] [PRINTJOB] [READONLY] [REPLYTO] [REPORT|MODULE] [ROLE] [RUNDEBUG] [SAVE_RDF] [SUBJECT] [TRACEFILE] [TRACEMODE] [TRACEOPTS] [USERID]

A.1.3 rwbuilder

The `rwbuilder` command invokes the Reports Builder. When you include a `REPORT|MODULE` keyword with the `rwbuilder` command at the command prompt, then press Enter, the Reports Builder opens with the specified report highlighted in the Reports Builder Navigator. When no report is specified, the Reports Builder opens with a Welcome dialog offering you the choice of opening an existing report or creating a new one.

Keywords used with rwbuilder

The brackets surrounding each keyword in this list are there to create a separation between keywords and has no other significance.

rwbuilder [ACCESSIBLE] [ARRAYSIZE] [AUTOCOMMIT] [BLANKPAGES] [BUFFERS] [CACHELOB] [CMDFILE] [EXPRESS_SERVER] [LONGCHUNK] [MODULE|REPORT] [NONBLOCKSQL] [ONFAILURE] [ONSUCCESS] [PAGESIZE] [PRINTJOB] [READONLY] [REPORT|MODULE] [RUNDEBUG] [SAVE_RDF] [TRACEFILE] [TRACEMODE] [TRACEOPTS] [USERID] [WEBSERVER_DEBUG] [WEBSERVER_DOCROOT] [WEBSERVER_PORT]

A.1.4 rwconverter

The `rwconverter` command enables you to convert one or more report definitions or PL/SQL libraries from one storage format to another. For example, you can use `rwconverter` to:

- Combine a report file with an XML file to create a new report

  Note: When a report is converted to a template, only objects in the report's header and trailer sections and margin area are used in the template. Objects in the main section are ignored.

- Convert a report stored in an .rdf file to .rep or .rex files, or a .tdf file
- Convert a report stored in a .rex file to an .rdf or .tdf file

- Convert a library stored in the database to a .pld or .pll file
- Convert a library stored in a .pld file into a database library or a .pll file
- Convert a library stored in a .pll file into a database library of a .pld file

> **Note:** When you convert a report that has an attached library, convert the .pll files attached to the report before converting the .rdf/.rex file.

- Batch create a PL/SQL script that batch registers reports in Oracle9*i*AS Portal

In some cases, `rwconverter` will automatically compile the report's PL/SQL as part of the conversion process. Provided your conversion destination is not a .rex file, `rwconverter` automatically compiles PL/SQL under the following conditions:

- Converting a .rep file
- Using a .rex file as the source
- Using a report created on another platform as the source

In all other situations, you must compile the report's PL/SQL yourself (e.g., using **Program > Compile > All** in the Reports Builder).

> **Note:** Fonts are mapped when a report is opened by the Reports Builder or Reports Runtime, not during the conversion.

Keywords used with rwconverter

The brackets surrounding each keyword in this list are there to create a separation between keywords and has no other significance.

```
rwconverter [BATCH] [CMDFILE] [CUSTOMIZE] [DEST] [DTYPE] [DUNIT] [FORMSIZE]
[NOTIFYFAILURE] [OVERWRITE] [PAGESIZE] [SOURCE] [STYPE] [USERID] [P_OWNER] [P_
SERVERS] [P_NAME] [P_DESCRIPTION] [P_PRIVILEGE] [P_AVAILABILITY] [P_TYPES] [P_
FORMATS] [P_PRINTERS] [P_PFORMTEMPLATE] [P_TRIGGER]
```

A.1.5 rwservlet

The `rwservlet` command translates and delivers information between HTTP and the Reports Server.

Note: When you use the `rwservlet` command to run a JSP, you can use all keywords applicable to `rwservlet`. For more information on running a JSP with `rwservlet`, see Chapter 8, "Running Report Requests".

Keywords used with rwservlet

The brackets surrounding each keyword in this list are there to create a separation between keywords and has no other significance.

```
rwservlet [ACCESSIBLE] [ARRAYSIZE] [AUTHID] [AUTOCOMMIT] [BCC] [BLANKPAGES]
[BUFFERS] [CACHELOB] [CC] [CELLWRAPPER] [CMDKEY] [CONTENTAREA] [COPIES]
[CUSTOMIZE] [DATEFORMATMASK] [DELAUTH] [DELIMITED_HDR] [DELIMITER] [DESFORMAT]
[DESNAME] [DESTINATION] [DESTYPE] [DISTRIBUTE] [EXPIRATION] [EXPIREDAYS]
[EXPRESS_SERVER] [FROM] [GETJOBID] [GETSERVERINFO] [HELP] [IGNOREMARGIN]
[ITEMTITLE] [JOBNAME] [JOBTYPE] [KILLJOBID] [LONGCHUNK] [MODE] [MODULE|REPORT]
[NONBLOCKSQL] [NOTIFYFAILURE] [NOTIFYSUCCESS] [NOTIFYFAILURE] [ONFAILURE]
[ONSUCCESS] [ORIENTATION] [OUTPUTFOLDER] [OUTPUTPAGE] [PAGEGROUP] [PAGESIZE]
[PAGESTREAM] [PARAMFORM] [PARSEQUERY] [PDFCOMP] [PDFEMBED] [READONLY]
[REPLACEITEM] [REPLYTO] [REPORT|MODULE] [ROLE] [RUNDEBUG] [SCHEDULE] [SERVER]
[SHOWENV] [SHOWENV] [SHOWMAP] [SHOWJOBS] [SHOWMYJOBS] [SITENAME] [STATUSFORMAT]
[STATUSFOLDER] [STATUSPAGE] [SSOCONN] [SUBJECT] [TOLERANCE] [TRACEMODE]
[TRACEOPTS] [URLPARAMETER] [USERID]
```

Note: The following keywords are commands rather than keyword-value pairs, i.e., commands are entered by themselves without a corresponding value: showenv, showjobs, showmap, showmyjobs, showjobid, killjobid, parsequery, showauth, delauth, getjobid, and getserverinfo. Refer to the syntax description for each of these keywords for more information.

A.1.6 rwcgi

Like `rwservlet`, the `rwcgi` command translates and delivers information between HTTP and the Reports Server. Between `rwservlet` and `rwcgi`, the `rwservlet` command is the recommend choice. Reports CGI is maintained only for backward compatibility.

Keywords used with rwcgi

The brackets surrounding each keyword in this list are there to create a separation between keywords and has no other significance.

```
rwcgi [ACCESSIBLE] [ARRAYSIZE] [AUTHID] [AUTOCOMMIT] [BCC] [BLANKPAGES]
[BUFFERS] [CACHELOB] [CC] [CELLWRAPPER] [CONTENTAREA] [COPIES] [CUSTOMIZE]
[DATEFORMATMASK] [DELIMITED_HDR] [DELIMITER] [DESFORMAT] [DESNAME] [DESTINATION]
[DESTYPE] [DISTRIBUTE] [EXPIRATION] [EXPIREDAYS] [EXPRESS_SERVER] [FROM]
[IGNOREMARGIN] [ITEMTITLE] [JOBNAME] [JOBTYPE] [LONGCHUNK] [MODE]
[MODULE|REPORT] [NONBLOCKSQL] [NOTIFYFAILURE] [NOTIFYSUCCESS] [NOTIFYFAILURE]
[ONFAILURE] [ONSUCCESS] [ORIENTATION] [OUTPUTFOLDER] [OUTPUTPAGE] [PAGEGROUP]
[PAGESIZE] [PAGESTREAM] [PARAMFORM] [PDFCOMP] [PDFEMBED] [READONLY]
[REPLACEITEM] [REPLYTO] [REPORT|MODULE] [ROLE] [RUNDEBUG] [SCHEDULE] [SERVER]
[SITENAME] [STATUSFOLDER] [STATUSPAGE] [SUBJECT] [TOLERANCE] [TRACEMODE]
[TRACEOPTS] [USERID]
```

A.1.7 rwserver

The `rwserver` command invokes the Reports Server. The Reports Server processes client requests, which includes ushering them through its various services, such as authentication and authorization checking, scheduling, caching, and distribution (including distribution to custom—or pluggable—output destinations). Reports Server also spawns runtime engines for generating requested reports, fetches completed reports from the Reports cache, and notifies the client that the job is ready.

Keywords used with rwserver

The brackets surrounding each keyword in this list are there to create a separation between keywords and has no other significance.

```
[AUTHID] [AUTOSTART] [BATCH] [SERVER] [SHUTDOWN] [TRACEOPTS] [INSTALL]
[UNINSTALL]
```

A.2 Command Line Syntax

Following is the syntax for a command line, where `keyword=value` is a valid command line argument:

```
rwclient REPORT|MODULE=runfile USERID=username/password@database
[ [keyword=]value|(value1, value2, ...) ] SERVER=server_name
```

Keywords must be specified and can be used in any order following the command.

A.3 General Usage Notes

- All file names and paths specified in the client command line refer to files and directories on the server machine, except for any file specified for the command file keyword (CMDFILE=). In this case, the CMDFILE specified will be read and appended to the original command line (of which CMDFILE is a part) before being sent to the Reports Server. The runtime engine will not re-read the CMDFILE.
- If you don't specify a path for a keyword value that includes a file name, The Reports Server will try to find the file from the REPORTS_PATH environment variable.
- If the command line contains CMDFILE=, then the command file is read and appended to the original command line before being sent to Oracle9*i*AS Reports Server. The Oracle9*i*AS Reports Engine does not reread the command file. (See CMDFILE.)

A.4 Command Line Arguments

A.4.1 ACCESSIBLE

Table A–1 indicates which commands can use the ACCESSIBLE keyword.

***Table A–1 Commands that can use* ACCESSIBLE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	yes	no	yes	yes	no

Description Use ACCESSIBLE to specify whether accessibility-related features offered through Reports are enabled (YES) or disabled (NO) for PDF output. No means it isn't.

Syntax ACCESSIBLE={YES|NO}

Values YES means accessibility feature is enabled for Reports PDF output.

Default NO

A.4.2 ARRAYSIZE

Table A–2 indicates which commands can use the ARRAYSIZE keyword.

***Table A–2 Commands that can use* ARRAYSIZE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	yes	no	yes	yes	no

Description Use ARRAYSIZE to specify the size (in kilobytes) for use with ORACLE array processing. Generally, the larger the array size, the faster the report will run.

Syntax ARRAYSIZE=*n*

Values A number from 1 through 9999 (note no comma is used with thousands). This means that Oracle9*i*AS Reports Runtime can use this number of kilobytes of memory per query in your report.

Default 10

Usage Notes ARRAYSIZE can also be used with jobs run as JSPs.

A.4.3 AUTHID

Table A–3 indicates which commands can use the AUTHID keyword.

***Table A–3 Commands that can use* AUTHID**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	yes

Description Use AUTHID to specify the user name and, optionally, the password to be used to authenticate users to the restricted Oracle9*i*AS Reports Server. User authentication ensures that the users making report requests have access privileges to run the requested report.

Syntax AUTHID=u*sername*[*/password]*

Values Any valid user name and, optionally, password created in Oracle9*i*AS Portal. See your DBA to create new users accounts in Oracle9*i*AS Portal.

Default None

Usage Notes AUTHID can also be used with jobs run as JSPs.

If you have a single sign-on environment, then the Oracle Single Sign-on Server will perform the authentication step and pass only the user name to the Reports Server in AUTHID.

A.4.4 AUTOCOMMIT

Table A–4 indicates which commands can use the AUTOCOMMIT keyword.

***Table A–4 Commands that can use* AUTOCOMMIT**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	yes	no	yes	yes	no

Description Use AUTOCOMMIT to specify whether database changes (for example, CREATE) should be automatically committed to the database. Some non-Oracle databases (for example, SQL Server) require that AUTOCOMMIT=YES.

Syntax AUTOCOMMIT={YES|NO}

Values YES or NO

Default NO

Usage Notes AUTOCOMMIT can also be used with jobs run as JSPs.

A.4.5 AUTOSTART

Table A–5 indicates which commands can use the AUTOSTART keyword.

***Table A–5 Commands that can use* AUTOSTART**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	no	no	no	yes

Description Use AUTOSTART to specify that the Reports Server will automatically start after initial installation and after a reboot, without requiring a user logon.

Syntax AUTOSTART={YES|NO}

Values YES or NO

Default NO

Usage Notes The AUTOSTART keyword is only recognized on Microsoft Windows platforms.

A.4.6 BATCH

Table A–6 indicates which commands can use the BATCH keyword.

***Table A–6 Commands that can use* BATCH**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	yes	no	no	yes

Description With rwconverter, BATCH suppress all terminal input and output in order to convert reports / libraries without user intervention. With rwserver, BATCH turns the server window dialog off (YES) or on (NO) to display or suppress process messages.

For all relevant commands, the BATCH option tells the server to run in no-UI mode. How it is used across commands is similar in that no UI is produced by the application when running from a command line that includes BATCH=YES. For example, for rwserver this allows the server to be run from scripts and remote agents so that no server dialog comes up while it is running.

Syntax BATCH={YES|NO}

Default NO

A.4.7 BCC

Table A–7 indicates which commands can use the BCC keyword.

***Table A–7 Commands that can use* BCC**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use BCC to specify e-mail recipient(s) of a blind courtesy copy.

> **Note:** A blind copy is one in which the names of specified recipients are not visible (published) to other recipients.

Syntax `BCC=someone@foo.com OR BCC="someone@foo.com,sometwo@foo.com"`

Values Any one or more valid e-mail addresses.

Default None

Usage Notes To specify more than one e-mail address, enclose the list of addresses in quotation marks and separate each address in the list with a comma.

Related keywords include BCC, CC, FROM, REPLYTO, and SUBJECT. Note that DESNAME is used to specify the main recipient(s) of the e-mail.

BCC can also be used with jobs run as JSPs.

A.4.8 BLANKPAGES

Table A–8 indicates which commands can use the BLANKPAGES keyword.

***Table A–8 Commands that can use* BLANKPAGES**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	yes	no	yes	yes	no

Description Use BLANKPAGES to specify whether to suppress blank pages when you print a report. Use this keyword when there are blank pages in your report output that you do not want to print.

Syntax `BLANKPAGES={YES|NO}`

Values YES means print all blank pages. NO means do not print blank pages.

Default YES

Usage Notes BLANKPAGES is especially useful if your logical page spans multiple physical pages (or panels), and you wish to suppress the printing of any blank physical pages.

A.4.9 BUFFERS

Table A–9 indicates which commands can use the BUFFERS keyword.

***Table A–9 Commands that can use* BUFFERS**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	yes	no	yes	yes	no

Description Use BUFFERS to specify the size of the virtual memory cache in kilobytes. You should tune this setting to ensure that you have enough space to run your reports, but not so much that you are using too much of your system's resources.

Syntax BUFFERS=*n*

Values A number from 1 through 9999 (note that thousands are not expressed with any internal punctuation, e.g., a comma or a decimal point). For some operating systems, the upper limit might be lower.

Default 640

Usage Notes If this setting is changed in the middle of your session, then the change does not take effect until the next time the report is run.

BUFFERS can also be used with jobs run as JSPs.

A.4.10 CACHELOB

Table A–10 indicates which commands can use the CACHELOB keyword.

***Table A–10 Commands that can use* CACHELOB**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	yes	no	yes	yes	no

Description Use CACHELOB to specify whether to cache retrieved Oracle8 large object or objects in the temporary file directory on the Reports Server (specified in the environment variable REPORTS_TMP or by the *tempDir* property of the *engine* element in the Reports Server configuration file, <*server_name*>.conf>; note that a *tempDir* setting overrides a REPORTS_TMP setting.).

Syntax `CACHELOB=NO`

Values YES means to cache the LOB in the temporary file directory. NO means to not cache the LOB in the temporary file directory.

Default YES

Usage Notes

- You can only set this option on the command line.
- If the location of the temporary file directory on the server does not have sufficient available disk space, then it is preferable to set this value to NO. Setting the value to NO, however, might decrease performance, as the LOB might need to be fetched from the database multiple times.
- `CACHELOB` can also be used with jobs run as JSPs.

A.4.11 CC

Table A–12 indicates which commands can use the `CC` keyword.

***Table A–11 Commands that can use* `CC`**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use CC to specify e-mail recipient(s) of a courtesy copy.

Syntax `CC=someone@foo.com OR CC="someone@foo.com,sometwo@foo.com"`

Values Any one or more valid e-mail addresses.

Default None

Usage Notes To specify more than one e-mail address, enclose the list of addresses in quotation marks and separate each address in the list with a comma.

Related keywords include BCC, CC, FROM, REPLYTO, and SUBJECT. Note that DESNAME is used to specify the main recipient(s) of the e-mail.

A.4.12 CELLWRAPPER

Table A–12 indicates which commands can use the CELLWRAPPER keyword.

***Table A–12 Commands that can use* CELLWRAPPER**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use CELLWRAPPER to specify the character or characters that should be placed both before and after the cells in a delimited report output.

Syntax CELLWRAPPER=*value*

Values Any alphanumeric character or string of alphanumeric characters.

" means a double quotation mark is placed on each side of the cell

' means a single quotation mark is placed on each side of the cell

You can also use these reserved values:

tab means a tab is placed on each side of the cell

space means a single space is placed on each side of the cell

return means a new line is placed on each side of the cell

none means no cell wrapper is used

You can also use escape sequences based on the ASCII character set, such as:

\t means a tab is placed on each side of the cell

\n means a new line is placed on each side of the cell

Default None

Usage Notes

- This keyword can only be used if you have specified DESFORMAT=DELIMITED.
- The cell wrapper is different from the actual delimiter. The cell wrapper specifies what character appears around delimited data. The delimiter indicates the boundary or break point between two pieces of data.

A.4.13 CMDFILE

Table A–13 indicates which commands can use the CMDFILE keyword.

***Table A–13 Commands that can use* CMDFILE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	yes	yes	no	no	no

Description Use CMDFILE to call a file that contains one report's command line arguments. The file called must be an ASCII file, either .txt or any other ASCII-type file.

CMDFILE differs from the cgicmd.dat file, in that CMDFILE can contain one command line for one report, where the cgicmd.dat file can contain multiple key-identified commands for multiple reports.

The CMDFILE keyword enables you to run a report without specifying a large number of arguments each time you invoke a run command.

Syntax CMDFILE=*cmdfile*

Values Any valid command file.

Default None

Usage Notes

- With rwservlet and rwcgi, use the *key* argument to refer to a key in the cgicmd.dat file in lieu of using the CMDFILE keyword.
- A command file can reference another command file.
- The syntax for a command line you specify in the command file is identical to that used on the command line.
- Values entered on the command line override values specified in command files. For example, suppose you specify rwclient from the command line with COPIES set to 1 and CMDFILE set to RUNONE (a command file). The RUNONE file also specifies a value for COPIES, but it is set to 2. The value specified for COPIES in the command line (1) overrides the value specified for COPIES in the RUNONE file (2). Only one copy of the report will be generated.
- The value for this keyword might be operating system-specific.

A.4.14 CMDKEY

Table A–14 indicates which commands can use the CMDKEY keyword.

***Table A–14 Commands that can use* CMDKEY**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	no	yes	no	no

Description Use CMDKEY to call a key-identified command line in the cgicmd.dat file. For example:

```
http:// …/reports/rwservlet?cmddkey=key& …
```

> **Note:** You can also use CMDKEY with modules run as JSPs. For more information, see Chapter 8, "Running Report Requests".

Syntax `CMDKEY=key`

Values The name of any key associated with a command line specified in the cgicmd.dat file.

Default None

Usage Notes When you use CMDKEY with `rwservlet`, you can use it in any order in the command line (or the URL, following the question mark). With `rwservlet`, you can use additional command line keywords along with CMDKEY.

CMDKEY can also be used with jobs run as JSPs.

A.4.15 CONTENTAREA

Table A–15 indicates which commands can use the CONTENTAREA keyword.

***Table A–15 Commands that can use* CONTENTAREA**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use CONTENTAREA to specify the content area within Oracle Portal to which report output should be pushed. This keyword is maintained for backward compatibility with earlier versions of Oracle Portal (i.e., 3.0.9). For Oracle9*i*AS Portal, use PAGEGROUP. (See also SITENAME.)

Syntax `CONTENTAREA="Name of Portal content area"`

Values The name of any valid Oracle Portal content area.

Default None

Usage Notes Use of this keyword is required to push Reports output to Oracle Portal. Put quotation marks around the value if the value has any character spaces in it or you are specifying the argument in the cgicmd.dat file.

Relevant keywords include CONTENTAREA, EXPIREDAYS, ITEMTITLE, OUTPUTFOLDER, OUTPUTPAGE, PAGEGROUP, REPLACEITEM, SCHEDULE, SITENAME, STATUSFOLDER, STATUSPAGE.

Oracle Portal objects, such as pages, page groups, and the like, have two names: a display name and an internal name. When you create objects within Oracle Portal that you will use with Oracle Reports output, keep the internal name and the display name the same, following the rules for internal naming specified in the Oracle Portal online help. This way, when you provide a value for a Portal-related keyword in a Reports command line, you will not run into problems with which name to specify.

A.4.16 COPIES

Table A–16 indicates which commands can use the `COPIES` keyword.

***Table A–16 Commands that can use* `COPIES`**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use COPIES to specify the number of copies of the report output to print.

Syntax `COPIES=n`

Values Any valid integer from 1 through 9999 (note that thousands are not expressed with any internal punctuation, e.g., a comma or a decimal point).

Default Taken from the Initial Value property of the COPIES parameter (the Initial Value was defined in the Reports Builder at design time).

Usage Notes

- This keyword is ignored if DESTYPE is not PRINTER.
- If COPIES is left blank on the Runtime Parameter Form, then it defaults to one.

A.4.17 CUSTOMIZE

Table A–17 indicates which commands can use the CUSTOMIZE keyword.

***Table A–17 Commands that can use* CUSTOMIZE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	yes	yes	yes	no

Description Use CUSTOMIZE to specify a Reports XML file to be run against the current report. The Reports XML file contains customizations (for example, changes to the layout or data model) that change the report definition in some way.

Syntax CUSTOMIZE=*filename*.xml | (*filename1*.xml, *filename2.xml*, …)

Values A file name or list of file names that contain a valid XML report definition, with path information prefixed to the file name or file names if necessary. (Affixing paths becomes necessary if the files are not located in a path specified in the REPORTS_PATH registry or SourceDir property for the engine element).

Note: For more information on customizing reports at runtime with XML customization files, see Chapter 10, "Customizing Reports with XML".

Default None

Usage Notes CUSTOMIZE can also be used with jobs run as JSPs.

A.4.18 DATEFORMATMASK

Table A–18 indicates which commands can use the DATEFORMATMASK keyword.

***Table A–18 Commands that can use* DATEFORMATMASK**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use DATEFORMATMASK to specify how date values display in your delimited report output.

Syntax `DATEFORMATMASK=`*`mask`*

Values Any valid date format mask.

Default None

Usage Notes This keyword can only be used if you have specified DESFORMAT=DELIMITED

> **Note:** For valid DATEFORMATMASK values see the Reports Builder online help topic, "Date and Time Format Mask Syntax."

DATEFORMATMASK can also be used with jobs run as JSPs.

A.4.19 DELAUTH

Table A–19 indicates which commands can use the DELAUTH keyword.

***Table A–19 Commands that can use* DELAUTH**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	no	yes	yes	no

Description Use DELAUTH to delete rwservlet or rwcgi userid cookies.

Syntax `http://`*`yourwebserver/yourservletpath`*`/rwservlet/DELAUTH[?]`
`[server=`*`servername`*`][&authid=`*`username/password`*`]`

Values See Syntax.

Default None

Usage Notes Related keywords are SERVER and AUTHID.

A.4.20 DELIMITED_HDR

Table A–20 indicates which commands can use the DELIMITED_HDR keyword.

***Table A–20 Commands that can use* DELIMITED_HDR**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use DELIMITED_HDR to switch off all boilerplate (such as the Report header) when running a report with DESFORMAT=DELIMITED.

Syntax DELIMITED_HDR={YES|NO}

Values YES means to turn off all boilerplate text in the delimited output file. NO means to leave boilerplate text as is in the delimited output file.

Default YES

Usage Notes This keyword can be used only if you have specified DESFORMAT=DELIMITED.

A.4.21 DELIMITER

Table A–21 indicates which commands can use the DELIMITER keyword.

***Table A–21 Commands that can use* DELIMITER**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use DELIMITER to specify the character or characters to use to separate the cells in your report output.

Syntax DELIMITER=*value*

Values Any alphanumeric character or string of alphanumeric characters, such as:

,	means a comma separates each cell
.	means a period separates each cell

You can also use these reserved values:

tab	means a tab separates each cell
space	means a space separates each cell
return	means a new line separates each cell
none	means no delimiter is used

You can also use escape sequences based on the ASCII character set, such as:

\t	means a tab separates each cell
\n	means a new line separates each cell

Default Tab

Usage Notes This keyword can be used only if you have specified DESFORMAT=DELIMITED.

A.4.22 DESFORMAT

Table A–22 indicates which commands can use the DESFORMAT keyword.

***Table A–22 Commands that can use* DESFORMAT**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Specifies the format for the job output. In bit-mapped environments, use DESFORMAT to specify the printer driver to be used when DESTYPE is FILE. In character-mode environments, use it to specify the characteristics of the printer named in DESNAME.

Syntax DESFORMAT=*desformat*

Values Any valid destination format not to exceed 1 kilobyte in length. Examples of valid values for this keyword are listed and described in Table A–23.

Table A–23 Valid values for DESFORMAT

Value	Description
CHARACTER	When the MODE is CHARACTER, the DESFORMAT specifies a printer definition, such as `hpl`, `hplwide`, `dec`, `decwide`, `decland`, `dec180`, `dflt`, or `wide`.
DELIMITED	This report output is sent to a file that can be read by standard spreadsheet utilities, such as Microsoft Excel. If you do not choose a delimiter, then the default delimiter is a TAB.
HTML	This report output is sent to a file that is in HTML format.
HTMLCSS	This report output is sent to a file that includes style sheet extensions.
PDF	This report output is sent to a file that is in PDF format and can be read by a PDF viewer, such as Adobe Acrobat.
POSTSCRIPT	This report output is sent to a file that is in Postscript format.
RTF	Rich Text Format. This report output is sent to a file that can be read by word processors (such as Microsoft Word). When you open the file in MS Word, you must choose **View > Page Layout** to view all the graphics and objects in your report.
XML	This report output is saved as an XML file. This report can be opened and read in an XML-supporting browser, or your choice of XML viewing application.

Default Taken from the Initial Value property of the DESFORMAT parameter (the Initial Value was defined in the Reports Builder at design time). When you run a report via the Reports Builder and DESFORMAT is blank or `dflt`, then the current printer driver (specified in **File > Choose Printer**) is used. If nothing has been selected in **Choose Printer**, then Postscript is used by default.

Usage Notes The value or values for this keyword might be case sensitive, depending on your operating system.

A.4.23 DESNAME

Table A–24 indicates which commands can use the `DESNAME` keyword.

Table A–24 Commands that can use **DESNAME**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use DESNAME to specify the name of the cache, file, printer, Oracle9*i*AS Portal, or e-mail ID (or distribution list) to which the report output will be sent. To send the report output by e-mail, specify the e-mail ID as you do in your e-mail application (any SMTP-compliant application). You can specify multiple user names by separating them with commas, and without spaces. For example:

```
name,name,name
```

Syntax `DESNAME=`*`desname`*

Values Any valid cache destination, file name, printer name, e-mail ID, or OraclePortal, not to exceed 1K in length. For printer names, you can optionally specify a port. For example:

```
DESNAME=printer,LPT1:
DESNAME=printer,FILE:
```

Default Taken from the Initial Value property of the DESNAME parameter (the Initial Value was defined in the Reports Builder at design time). If DESTYPE=FILE and DESNAME is an empty string, then it defaults to `reportname.lis` at runtime.

Usage Notes The argument(s) for this keyword might be case sensitive, depending on your operating system.

A.4.24 DEST

Table A–25 indicates which commands can use the DEST keyword.

Table A–25 Commands that can use **DEST**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	yes	no	no	no

Description Use DEST to specify the name(s) of the converted reports or libraries.

Syntax `DEST={`*`dname`*`|(`*`dname1`*`, `*`dname2`*`, …)|`*`pathname`*`}`

Values Any valid report/library name or filename, or a list of valid report/library names of filenames enclosed in parentheses and separated by commas (e.g., `(qanda, text, dmast)`).

Default If the DEST keyword is not specified, rwconverter uses the following default names:

- If DTYPE is PLDFILE, then the DEST default name is source.pld.
- If DTYPE is PLLFILE, then the DEST default name is source.pll.
- If DTYPE is RDFFILE, then the DEST default name is source.rdf.
- If DTYPE is REPFILE, then the DEST default name is source.rep.
- If DTYPE is REXFILE, then the DEST default name is source.rex.
- If DTYPE is XMLFILE, then the DEST default name is source.xml.
- If DTYPE is REGISTER, then the DEST default name is the name of the SQL*Plus script output file (e.g., output.sql).

Usage Notes

- A list of report/library names of filenames must be enclosed in parentheses with commas separating each entry. For example:

  ```
  (qanda,test,dmast) or (qanda, test, dmast)
  ```

- If you have more destination names than there are source names, the extra destination names are ignored. If you have fewer destination names than there are source names, default names will be used after the destination names run out.
- The value(s) for the DEST keyword may be operating system-specific.
- When DTYPE=REGISTER, multiple destinations are not required. If you list more than one SQL*Plus script file name for DEST, only the first one is recognized. The others are ignored.

A.4.25 DESTINATION

Table A–26 indicates which commands can use the `DESTINATION` keyword.

Table A–26 Commands that can use **DESTINATION**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use the DESTINATION keyword to specify the name of an XML file that defines the distribution for the current run of the report.

Syntax `DESTINATION=`*`filename`*`.xml`

Values The name of an XML file that defines a report or report section distribution.

Default None

Usage Notes To enable the DESTINATION keyword, you must specify DISTRIBUTE=YES on the command line. If both these keywords are specified, DESTYPE, DESNAME, and DESFORMAT are ignored if they are also specified.

Note: For more information about creating advanced distributions, see Chapter 9, "Creating Advanced Distributions".

A.4.26 DESTYPE

Table A–27 indicates which commands can use the DESTYPE keyword.

Table A–27 Commands that can use **DESTYPE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use DESTYPE to specify the type of device that will receive the report output. If you have created your own pluggable destination via the Reports Destination API, this is how the destination you created gets called.

Syntax `DESTYPE={cache|localFile|file|printer|sysout|mail|oraclePortal|`*`name_of_pluggable_destination`*`}`

Values Table A–28 lists and describes the valid values for the DESTYPE keyword.

Table A–28 Valid values for the **DESTYPE** *keyword*

Value	Description
cache	Sends the output directly to Oracle9*i*AS Reports cache.
localFile	Valid only for `rwclient`, `rwcgi`, and `rwservlet`. Sends the output to a file on the client machine, synchronously or asynchronously.
file	Sends the output to the file on the server named in DESNAME.
printer	Sends the output to the printer on the server named in DESNAME. You must have a printer that the Oracle9*i*AS Reports Server can recognize installed and running.
mail	Sends the output to the mail users specified in DESNAME. You can send mail to any mail system that works with SMTP.
OraclePortal	Sends the output to Oracle Portal. Relevant keywords include CONTENTAREA, EXPIREDAYS, ITEMTITLE, OUTPUTFOLDER, OUTPUTPAGE, PAGEGROUP, REPLACEITEM, SCHEDULE, SITENAME, STATUSFOLDER, STATUSPAGE.
sysout	Valid only for rwcgi. Sends the output to the client machine's default output device and forces a synchronous call.
name_of_pluggable_ destination	If you have created your own pluggable destination via the Reports Destination API, this is what you use to call the destination you created.

Default Taken from the Initial Value property of the DESTYPE parameter (the Initial Value was defined in the Reports Builder at design time).

A.4.27 DISTRIBUTE

Table A–29 indicates which commands can use the `DISTRIBUTE` keyword.

Table A–29 Commands that can use **DISTRIBUTE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use DISTRIBUTE to enable or disable distributing the report output to multiple destinations, as specified by the distribution list defined in the report distribution definition (defined in the Reports Builder at design time) or a distribution XML file.

Syntax DISTRIBUTE={YES|NO}

Values YES means to distribute the report to the distribution list. NO means to ignore the distribution list and output the report as specified by the DESNAME, DESTYPE, and DESFORMAT parameters. NO is fundamentally a debug mode to allow running a report set up for distribution without actually executing the distribution.

Default NO

Usage Notes The DISTRIBUTE keyword works in close association with the DESTINATION keyword. DISTRIBUTE must have a value of YES for the DESTINATION keyword to take effect. If both these keywords are specified, DESTYPE, DESNAME, and DESFORMAT are ignored if they are also specified.

Note: For more information about creating advanced distributions, see Chapter 9, "Creating Advanced Distributions".

A.4.28 DTYPE

Table A–30 indicates which commands can use the DTYPE keyword.

***Table A–30 Commands that can use* DTYPE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	yes	no	no	no

Description Use DTYPE to specify the format to which to convert the reports or libraries.

Syntax DTYPE={PLDFILE|PLLFILE|RDFFILE|REPFILE|TDFFILE|XMLFILE|JSPFILE |REGISTER}

Values The following values apply:

- PLDFILE means the converted PL/SQL libraries will be stored in files in ASCII format.
- PLLFILE means the converted PL/SQL libraries will be stored in files containing source code and P-code (compiled PL/SQL).

- RDFFILE means the converted report(s) will be stored in one or more report definition files (files with the `.rdf` extension).
- REPFILE means the converted report(s) will be stored in one or more binary runfiles (files with the `.rep` extension).
- REXFILE means the converted report(s) will be stored in one or more text files (files with the `.rex` extension).
- TDFFILE means the report will be converted to a template file (files with the `.tdf` extension).
- XMLFILE means the converted report(s) will be stored in an XML file (files with the `.xml` extension).
- JSPFILE means the converted report(s) will be stored in a JSP file (files with the `.jsp` extension).
- REGISTER means that a script file is created to load each report specified by SOURCE into Oracle9*i*AS Portal with the RWWWVREG.REGISTER_REPORT function. Each load function is populated with the necessary information to register the report in Oracle9*i*AS Portal. By running the resulting script file in SQL*Plus against the Oracle9*i*AS Portal schema, you can batch register multiple reports in Oracle9*i*AS Portal.

Default REPFILE

Usage Notes When you try to create a .rep file using `rwconverter`, the source report's PL/SQL is automatically compiled. If there are compile errors, an error message is displayed and the .rep file is not created. To avoid this problem, make sure you compile the source report's PL/SQL using **File > Compile**, in the Reports Builder, before you try to create a .rep file.

When converting a report to a template, only objects in the report's header and trailer sections and the margin area are used in the template. Objects in the main section are ignored.

A.4.29 DUNIT

Table A–31 indicates which commands can use the `DUNIT` keyword.

***Table A–31 Commands that can use* `DUNIT`**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	yes	no	no	no

Description Use DUNIT to specify the destination unit of measurement to which the report should be converted. If specified, DUNIT must differ from the SOURCE report's unit of measurement. If unspecified, the SOURCE report's unit of measurement is used.

Syntax `DUNIT={CENTIMETER|CHARACTER|INCH|POINT}`

Values

- CENTIMETER means the converted reports will initially use centimeters as the unit of measurement
- CHARACTER means the converted reports will initially use characters as the unit of measurement.
- INCH means the converted reports will initially use inches as the unit of measurement.
- POINT means the converted reports will initially use points as the unit of measurement

Default Null

A.4.30 EXPIRATION

Table A–32 indicates which command can use the `EXPIRATION` keyword.

***Table A–32 Commands that can use* `EXPIRATION`**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	no	no	no	yes	yes	no

Description Use EXPIRATION to define how long report output can exist in cache before it is deleted.

See Section 8.8, "Reusing Report Output from Cache" (in Chapter 8) for more information on duplicate job detection.

Syntax `EXPIRATION=`*`time_string`*

Default None

Values The time string can be in one of two formats:

- *n*{*unit*}, for a number with an optional unit. The unit can be minute(s), hour(s), or day(s). The default unit is minute(s) if no unit is specified.
- {*Mon DD, YYYY*} *hh*:*mi*:*ss* am|pm {*timezone*}, for a date/time format. Date information is optional. If it isn't specified, *today* is assumed. Time zone is also optional. If it isn't specified, the Reports Server's timezone is used. The date/time is always in a US locale. This format is the same as defined in the Java DateFormat.MEDIUM type.

A.4.31 EXPIREDAYS

Table A–33 indicates which commands can use the EXPIREDAYS keyword.

***Table A–33 Commands that can use* EXPIREDAYS**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	no	yes	yes	no

Description Use EXPIREDAYS to specify the number of days after which the reports output pushed to Oracle Portal should be expired.

Syntax `EXPIREDAYS={PERMANENT|1 day|2 days|3 days|7 days|14 days|31 days|60 days|90 days|120 days}`

Values PERMANENT (does not expire) | 1 day | 2 days | 3 days | 7 days | 14 days | 31 days | 60 days | 90 days | 120 days.

Default None

Usage Notes Use of this keyword is optional when you are pushing Reports output to Oracle Portal. Relevant keywords include CONTENTAREA, EXPIREDAYS, ITEMTITLE, OUTPUTFOLDER, OUTPUTPAGE, PAGEGROUP, REPLACEITEM, SCHEDULE, SITENAME, STATUSFOLDER, STATUSPAGE.

A.4.32 EXPRESS_SERVER

Table A–34 indicates which commands can use the EXPRESS_SERVER keyword.

***Table A–34 Commands that can use* EXPRESS_SERVER**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	yes	no	yes	yes	no

Description Use EXPRESS_SERVER to specify the Express Server to which you want to connect.

Syntax `EXPRESS_SERVER="server=[server]/domain=[domain]/user=[userid]/password=[passwd]"`

Syntax with RAM `EXPRESS_SERVER="server=[server]/domain=[domain]/user=[userid]/password=[passwd]/ramuser=[ramuserid]/rampassword=[rampasswd]/ramexpressid=[ramexpid]/ramserverscript=[ramsscript]/rammasterdb=[ramdb]/ramconnecttype=[ramconn]"`

Values A valid connect string enclosed in double quotes (") where:

`server`	is the Express Server string (for example, `ncacn_ip_tcp:olap2-pc/sl=x/st=x/ct=x/sv=x/`). See below for more details on the server string.
`domain`	is the Express Server domain.
`user`	is the user ID to log on to the Express Server.
`password`	is the password for the user ID.
`ramuser`	is the user ID to log into the RDBMS.
`rampassword`	is the password for the RDBMS.
`ramexpressid`	is the Oracle Sales Analyzer database user ID. This is required for Oracle Sales Analyzer databases only.
`ramserverscript`	is the complete file name (including the full path) of the remote database configuration file (RDC) on the server. This file specifies information such as the location of code and data databases. Using UNC (Universal Naming Convention) syntax allows multiple users to use the same connection to access the data without having to map the same drive letter to that location. UNC syntax is `\\`*`ServerName`*`\`*`ShareName`*`\` followed by any subfolders or files.

`rammasterdb`	is the name of the Relational Access Manager database to attach initially. You must specify only the database file name. This database must reside in a directory that is included in the path list in ServerDBPath for Express Server. You can check the ServerDBPath in the File I/O tab of the Express Configuration Manager dialog box.
`ramconnecttype`	is the type of Express connection. Always specify 0 for a direct connection.

Parameters The server value contains four parameters that correspond to settings that are made in the Oracle Express Connection Editor and stored in connection (XCF) files. All four parameters are required and can be specified in any order. Table A–35 describes the parameters and their settings:

Table A–35 Settings for parameters used with* EXPRESS_SERVER*'s server value

Parameter	Description	Setting
sl	Server Login	-2: Host (Domain Login) -1: Host (Server Login) 0: No authentication required 1: Host (Domain Login) and Connect security 2: Host (Domain Login) and Call security 3: Host (Domain Login) and Packet security 4: Host (Domain Login) and Integrity security 5: Host (Domain Login) and Privacy security **Note:** Windows NT uses all the settings. UNIX systems use only the settings 0, -1, and -2. See the Express Connection Editor Help system for information on these settings.
st	Server Type	:1: Express Server
ct	Connection Type	0: Express connection
sv	Server Version	1: Express 6.2 or greater

Default None

Usage Notes

- You can have spaces in the string if necessary (for example, if the user ID is John Smith) because the entire string is enclosed in quotes.
- If a forward slash (/) is required in the string, then you must use another forward slash as an escape character. For example, if the domain were tools or reports, then the command line should be as follows:

  ```
  EXPRESS_SERVER="server=ncacn_ip_tcp:olap2-pc/sl=0/
  st=1/ct=0/sv=1/domain=tools//reports"
  ```

- You can use single quotes within the string. They are not treated specially because the entire string is enclosed in double quotes.

A.4.33 FORMSIZE

Table A–36 indicates which commands can use the FORMSIZE keyword.

***Table A–36 Commands that can use* FORMSIZE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	yes	no	no	no

Description Use FORMSIZE to specify the size of the Runtime Parameter Form for the converted report in terms of the destination unit of measurement (DUNIT).

> **Note:** For more information on the Runtime Parameter Form, see the PARAMFORM keyword.

Syntax `FORMSIZE=`*`width`* `x` *`height`*

Values Any valid values in the specified unit of measurement.

Default None

Usage Notes For non-character DUNITs, you can use a decimal to specify fractions (e.g., 8.5 x 11).

A.4.34 FROM

Table A–37 indicates which commands can use the FROM keyword.

Table A–37 Commands that can use **FROM**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use FROM to specify the e-mail address of the sender of an e-mail.

Syntax FROM=*someone@foo.com*

Values Any valid e-mail address.

Default *loginid@machine_name*

Usage Notes Related keywords include BCC, CC, FROM, REPLYTO, and SUBJECT. Note that DESNAME is used to specify the main recipient(s) of the e-mail.

A.4.35 GETJOBID

Table A–38 indicates which commands can use the GETJOBID keyword.

Table A–38 Commands that can use **GETJOBID**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	no	yes	yes	no

Description Use GETJOBID to get the result output of the Reports Server job with job ID [*n*].

Syntax http://*yourwebserver*/reports/rwservlet/getjobid[*n*][?]
[server=s*erver_name*][&authid=*username/password*][&statusformat={html|xml|xmldtd}]

Values See Syntax.

Default None

Usage Notes Job must be successfully finished and present in the Reports Server cache. Use SHOWJOBS to see the current list of jobs. The status format can be html, xml, or xmldtd to return status in that format. The default is html.

Related keywords are SERVER, AUTHID, and STATUSFORMAT.

The STATUSFORMAT parameter is only valid for `rwservlet`, not for `rwcgi`.

A.4.36 GETSERVERINFO

Table A–39 indicates which commands can use the `GETSERVERINFO` keyword.

***Table A–39 Commands that can use* `GETSERVERINFO`**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	no	yes	no	no

Description Use `GETSERVERINFO` to display Reports Server information.

Syntax `http://`*`yourwebserver`*`/`*`reports`*`/rwservlet/getserverinfo[?]`
`[server=`*`server_name`*`][&authid=`*`username/password`*`]`

Values See Syntax.

Default None

Usage Notes Related keywords are SERVER and AUTHID.

A.4.37 HELP

Table A–40 indicates which commands can use the `HELP` keyword.

***Table A–40 Commands that can use* `HELP`**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	no	yes	yes	no

Description Use `HELP` to show a help topic that lists the additional commands you can use with the `rwservlet` command.

Syntax `http://`*`yourwebserver`*`/`*`reports`*`/rwservlet/help`

Values See Syntax.

Default None

A.4.38 IGNOREMARGIN

Table A–41 indicates which commands can use the IGNOREMARGIN keyword.

***Table A–41 Commands that can use* IGNOREMARGIN**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	yes	no	yes	yes	no

Description Use IGNOREMARGIN to specify whether Reports ignores the printer's hardware margin and uses one specified in the report definition's physical page margin.

Syntax `IGNOREMARGIN={YES|NO}`

Values YES means Reports will ignore the printer's hardware margin and use the one specified by the report's physical page margin. NO means Reports will add the printer's hardware margin with the report's physical page margin when it prints out the report.

Default NO

A.4.39 INSTALL

Table A–42 indicates which commands can use the INSTALL keyword

Table A–42 Commands that can use INSTALL

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	no	no	no	yes

Description Use INSTALL to configure an instance of the Reports Server on Microsoft Windows as a service. This argument does not work on UNIX platforms.

Syntax `INSTALL REPORTS_SERVER_NAME`

Values A valid name for the Reports Server instance

Default none

Usage Notes If you use the AUTOSTART keyword with INSTALL, the Reports Server service will be started automatically after installation and whenever the system is restarted.

If you use BATCH=YES with INSTALL, then none of the prompts and dialogs that normally display during installation will appear.

A.4.40 ITEMTITLE

Table A–43 indicates which commands can use the `ITEMTITLE` keyword.

***Table A–43 Commands that can use* `ITEMTITLE`**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use ITEMTITLE to specify the display name Oracle Portal should use for report output. The name will display in Oracle Portal and link to Reports output.

Syntax `ITEMTITLE="`*`Your output title`*`"`

Values Any text.

Default The report filename

Usage Notes Use of this keyword is optional when you are pushing Reports output to Oracle Portal. Put quotation marks around the value if the value has any character spaces in it or you are specifying the argument in the cgicmd.dat file.

Relevant keywords include CONTENTAREA, EXPIREDAYS, ITEMTITLE, OUTPUTFOLDER, OUTPUTPAGE, PAGEGROUP, REPLACEITEM, SCHEDULE, SITENAME, STATUSFOLDER, STATUSPAGE.

A.4.41 JOBNAME

Table A–44 indicates which commands can use the `JOBNAME` keyword.

***Table A–44 Commands that can use* `JOBNAME`**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	no	no	no	yes	yes	no

Description Use JOBNAME to specify the name for a job to appear in the Oracle9*i*AS Reports Queue Manager. It is treated as a comment and has nothing to do with running the job. If JOBNAME is not specified, then the Oracle9*i*AS Reports Queue Manager shows the report name as the job name.

Syntax `JOBNAME=`*`string`*

Values Any job name.

Default None

Usage Notes JOBNAME can also be used with jobs run as JSPs.

A.4.42 JOBTYPE

Table A–45 indicates which commands can use the JOBTYPE keyword.

***Table A–45 Commands that can use* JOBTYPE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	no	no	no	yes	yes	no

Description Use JOBTYPE to specify the type of job to be processed by the server. You can enter any type of job, as long as the Reports Server has an engine to process it.

Syntax JOBTYPE={a job for which the Reports Server has an engine}

Default REPORT

A.4.43 KILLJOBID

Table A–46 indicates which commands can use the KILLJOBID keyword.

***Table A–46 Commands that can use* KILLJOBID**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	no	yes	yes	no

Description Use KILLJOBID to kill a Reports Server job with the specified job ID [*n*].

Syntax http://*yourwebserver*/*reports*/rwservlet/killjobid[*n*][?] [server=*server_name*][&authid=*username*/*password*][&statusformat={html|xml|xmldtd}]

Values See Syntax.

Default None

Usage Notes The job must be current (enqueued or scheduled). Use SHOWJOBS to see the current list of jobs. The STATUSFORMAT can be set to html, xml, or xmldtd to return status in that format. The default is html.

Related keywords are SHOWJOBS, SERVER, AUTHID, and STATUSFORMAT.

The STATUSFORMAT parameter is only valid for rwservlet, not for rwcgi.

A.4.44 LONGCHUNK

Table A–47 indicates which commands can use the LONGCHUNK keyword.

***Table A–47 Commands that can use* LONGCHUNK**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	yes	no	yes	yes	no

Description LONGCHUNK is the size (in kilobytes) of the increments in which Oracle9*i*DS Reports Builder retrieves a LONG column value. When retrieving a LONG value, you might want to retrieve it in increments rather than all at once because of memory size restrictions. LONGCHUNK applies only to Oracle databases.

Syntax LONGCHUNK=*n*

Values A number from 1 through 9999 (note that thousands are not expressed with any internal punctuation, e.g., a comma or a decimal point). For some operating systems, the upper limit might be lower.

Default 10

Usage Notes LONGCHUNK can also be used with jobs run as JSPs.

A.4.45 MODE

Table A–48 indicates which commands can use the MODE keyword.

Table A–48 Commands that can use **MODE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use MODE to specify whether to run the report in character mode or bitmap.

Syntax `MODE={BITMAP|CHARACTER|DEFAULT}`

Values The following values apply:

- BITMAP
- CHARACTER
- DEFAULT means BITMAP.

Default DEFAULT

A.4.46 MODULE|REPORT

Table A–49 indicates which commands can use the MODULE|REPORT keyword.

Table A–49 Commands that can use **MODULE|REPORT**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	yes	no	yes	yes	no

Description Use MODULE or REPORT to specify the name of the report to run.

Syntax `REPORT|MODULE=`*`runfile`*

Values Any valid runfile (that is, a file with an extension of REP, RDF, JSP, or XML). If you do not enter a file extension, then the Oracle9*i*AS Reports Runtime searches first for a file with extension REP, then extension RDF, then JSP, and then no extension. Oracle9*i*AS Reports Runtime uses its REPORTS_PATH search order to find the file, if the directory path is not prefixed to the file name.

Default None

A.4.47 NONBLOCKSQL

Table A–50 indicates which commands can use the NONBLOCKSQL keyword.

***Table A–50 Commands that can use* NONBLOCKSQL**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	yes	no	yes	yes	no

Description Use NONBLOCKSQL to specify whether to allow other programs to execute while Oracle9*i*AS Reports Runtime is fetching data from the database.

Syntax `NONBLOCKSQL={YES|NO}`

Values YES means that other programs can run while data is being fetched. NO means that other programs cannot run while data is being fetched.

Default YES

Usage Notes NONBLOCKSQL can also be used with jobs run as JSPs.

A.4.48 NOTIFYFAILURE

Table A–51 indicates which commands can use the NOTIFYFAILURE keyword.

***Table A–51 Commands that can use* NOTIFYFAILURE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	no	no	no	yes	yes	no

Description Use NOTIFYFAILURE to specify the recipient(s) of a notification e-mail should a report request fail. Use this keyword when you configure your Reports Server to use the notification class. For more information, see the notification discussion in Chapter 3, "Configuring Oracle9iAS Reports Services".

Syntax `NOTIFYFAILURE={name1@mycompany.com,name2@mycompany.com}`

Values One or more valid e-mail addresses.

Default None

Usage Notes The default notification e-mail templates that are used for the body of the notification e-mail are included with your installation of Oracle9*i*AS. The NOTIFYFAILURE template is named failnote.txt, and is located at *ORACLE_HOME*\reports\template.

NOTIFYFAILURE can also be used with jobs run as JSPs.

A.4.49 NOTIFYSUCCESS

Table A–52 indicates which commands can use the NOTIFYSUCCESS keyword.

***Table A–52 Commands that can use* NOTIFYSUCCESS**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	no	no	no	yes	yes	no

Description Use NOTIFYSUCCESS to specify the recipient(s) of a notification e-mail should a report request succeed. Use this keyword when you configure your Reports Server to use the notification class. For more information, see the notification discussion in Chapter 3, "Configuring Oracle9iAS Reports Services".

Syntax NOTIFYSUCCESS={name1@mycompany.com,name2@mycompany.com}

Values One or more valid e-mail addresses.

Default None

Usage Notes The default notification e-mail templates that are used for the body of the notification e-mail are included with your installation of Oracle9*i*AS. The NOTIFYSUCCESS template is named succnote.txt, and is located at *ORACLE_HOME*\reports\template.

NOTIFYSUCCESS can also be used with jobs run as JSPs.

A.4.50 NUMBERFORMATMASK

Table A–53 indicates which commands can use the NUMBERFORMATMASK keyword.

***Table A–53 Commands that can use* NUMBERFORMATMASK**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use NUMBERFORMATMASK to specify how number values display in your delimited report output.

Syntax `NUMBERFORMATMASK=`*`mask`*

Values Any valid number format mask.

Default None

Usage Notes This keyword can only be used if you have specified DESFORMAT=DELIMITED.

> **Note:** For valid NUMBERFORMATMASK values see the Reports Builder online help topic, "Number Format Mask Syntax."

NUMBERFORMATMASK can also be used with jobs run as JSPs.

A.4.51 ONFAILURE

Table A–54 indicates which commands can use the ONFAILURE keyword.

***Table A–54 Commands that can use* ONFAILURE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	yes	no	yes	yes	no

Description Use ONFAILURE to specify whether you want a COMMIT or ROLLBACK performed if an error occurs and a report fails to complete.

Syntax `ONFAILURE={COMMIT|ROLLBACK|NOACTION}`

Values COMMIT means perform a COMMIT if a report fails. ROLLBACK means perform a ROLLBACK if a report fails. NOACTION means do nothing if a report fails.

Default ROLLBACK, if a USERID is provided. NOACTION, if called from an external source (for example, Oracle9*i*DS Forms Services) with no USERID provided.

Usage Notes The COMMIT or ROLLBACK for ONFAILURE is performed after the report fails. Other COMMITs and ROLLBACKs can occur prior to this one. For more information, see the READONLY command.

ONFAILURE can also be used with jobs run as JSPs.

A.4.52 ONSUCCESS

Table A–55 indicates which commands can use the ONSUCCESS keyword.

***Table A–55 Commands that can use* ONSUCCESS**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	yes	no	yes	yes	no

Description Use ONSUCCESS to specify that either a COMMIT or ROLLBACK should be performed when a report is finished running.

Syntax `ONSUCCESS={COMMIT|ROLLBACK|NOACTION}`

Values COMMIT means perform a COMMIT when a report is done. ROLLBACK means perform a ROLLBACK when a report is done. NOACTION means do nothing when a report is done.

Default COMMIT, if a USERID is provided. NOACTION, if called from an external source (for example, Oracle9*i*DS Forms Services) with no USERID provided.

Usage Notes The COMMIT or ROLLBACK for ONSUCCESS is performed after the after-report trigger fires. Other COMMITs and ROLLBACKs can occur prior to this one. For more information, see the READONLY command.

ONSUCCESS can also be used with jobs run as JSPs.

A.4.53 ORIENTATION

Table A–56 indicates which commands can use the ORIENTATION keyword.

***Table A–56 Commands that can use* ORIENTATION**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description ORIENTATION controls the direction in which the pages of the report will print.

Syntax `ORIENTATION={DEFAULT|LANDSCAPE|PORTRAIT}`

Values DEFAULT means use the current printer setting for orientation. LANDSCAPE means landscape orientation (long side at top and bottom). PORTRAIT means portrait orientation (short side at top and bottom).

Default DEFAULT

Usage Notes

- If ORIENTATION=LANDSCAPE for a character mode report, then you must ensure that your printer definition file contains a landscape clause.
- Not supported when output to a PCL printer on Motif.

A.4.54 OUTPUTFOLDER

Table A–57 indicates which commands can use the OUTPUTFOLDER keyword.

***Table A–57 Commands that can use* OUTPUTFOLDER**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use OUTPUTFOLDER to specify the name of the Oracle Portal folder to push Reports output into. This keyword is maintained for backward compatibility with earlier versions of Oracle Portal (WebDB 2.2 and Oracle Portal 3.0.9). For Oracle9*i*AS Portal version 2.0, use OUTPUTPAGE.

Syntax `OUTPUTFOLDER=Oracle_Reports_Output`

Values Any valid folder name used in Oracle9*i*AS Portal.

Default Oracle_Reports_Output

Usage Notes The value for this keyword is case sensitive. Use of this keyword is required to push Reports output to Oracle Portal. Put quotation marks around the value if the value has any character spaces in it or you are specifying the argument in the cgicmd.dat file.

Relevant keywords include CONTENTAREA, EXPIREDAYS, ITEMTITLE, OUTPUTFOLDER, OUTPUTPAGE, PAGEGROUP, REPLACEITEM, SCHEDULE, SITENAME, STATUSFOLDER, STATUSPAGE.

Oracle Portal objects, such as pages, page groups, and the like, have two names: a display name and an internal name. When you create objects within Oracle Portal that you will use with Oracle Reports output, keep the internal name and the display name the same, following the rules for internal naming specified in the Oracle Portal online help. This way, when you provide a value for a Portal-related keyword in a Reports command line, you will not run into problems with which name to specify.

A.4.55 OUTPUTPAGE

Table A–58 indicates which commands can use the OUTPUTPAGE keyword.

***Table A–58 Commands that can use* OUTPUTPAGE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use OUTPUTPAGE to specify the name of the Oracle Portal page to push Reports output information into. (For backward compatibility with versions of Oracle Portal earlier than Oracle9*i*AS Portal version 2.0, see OUTPUTFOLDER.)

Syntax `OUTPUTPAGE=Oracle_Reports_Output`

Values Any valid page name used in Oracle Portal.

Default Oracle_Reports_Output

Usage Notes The value for this keyword is case sensitive. Use of this keyword is optional for pushing Reports output to Oracle Portal. If an output page is not specified, Oracle9*i*AS Portal will create a default page named Oracle Reports Output.

Put quotation marks around the value if the value has any character spaces in it or you are specifying the argument in the cgicmd.dat file.

Keywords relevant to pushing Reports output to Oracle Portal include CONTENTAREA, EXPIREDAYS, ITEMTITLE, OUTPUTFOLDER, OUTPUTPAGE, PAGEGROUP, REPLACEITEM, SCHEDULE, SITENAME, STATUSFOLDER, STATUSPAGE.

Oracle Portal objects, such as pages, page groups, and the like, have two names: a display name and an internal name. When you create objects within Oracle Portal that you will use with Oracle Reports output, keep the internal name and the display name the same, following the rules for internal naming specified in the Oracle Portal online help. This way, when you provide a value for a Portal-related keyword in a Reports command line, you will not run into problems with which name to specify.

A.4.56 OVERWRITE

Table A–59 indicates which commands can use the OVERWRITE keyword.

***Table A–59 Commands that can use* OVERWRITE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	yes	no	no	no

Description Use OVERWRITE to specify whether to overwrite existing files with the converted files.

Syntax `OVERWRITE={YES|NO|PROMPT}`

Values

- YES means that rwconverter should automatically overwrite any existing files of the same name.
- NO means not to convert reports and to display a warning message if there are existing files of the same name.
- PROMPT means to prompt you before overwriting any existing files.

Default NO

A.4.57 P_AVAILABILITY

Table A–60 indicates which commands can use the P_AVAILABILITY keyword.

***Table A–60 Commands that can use* P_AVAILABILITY**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	yes	no	no	no

Description P_AVAILABILITY is the name of the availability calendar that determines when the reports specified will be available for processing. This keyword is only used when DTYPE=REGISTER.

Syntax `P_AVAILABILITY=calendar_name`

Values Any valid availability calendar name.

Default none

Usage Notes · The availability calendar must exist in Oracle9*i*AS Portal before running the SQL*PLUS script. If it does not, an invalid package may be created.

A.4.58 P_DESCRIPTION

Table A–61 indicates which commands can use the `P_DESCRIPTION` keyword.

***Table A–61 Commands that can use* `P_DESCRIPTION`**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	yes	no	no	no

Description P_DESCRIPTION is text that provides additional information about the report. This keyword is only used when DTYPE=REGISTER.

Syntax `P_DESCRIPTON=DESCRIPTION_TEXT`

Values Any text string.

Default none

A.4.59 P_FORMATS

Table A–62 indicates which commands can use the `P_FORMATS` keyword. This keyword is only used when DTYPE=REGISTER.

***Table A–62 Commands that can use* `P_FORMATS`**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	yes	no	no	no

Description P_FORMATS is the allowable destination formats for the specified reports. This keyword is only used when DTYPE=REGISTER.

Syntax `P_FORMATS=(HTMLCSS,PDF,...)`

Values Any valid destination type (e.g., HTML), or a list of valid destination types enclosed by parentheses with a comma separating the names (e.g., (HTMLCSS,PDF,RTF)).

Default none

A.4.60 P_NAME

Table A–63 indicates which commands can use the P_NAME keyword.

***Table A–63 Commands that can use* P_NAME**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	yes	no	no	no

Description P_NAME is the report name displayed in Oracle9*i*AS Portal. This keyword is only used when DTYPE=REGISTER.

Syntax `P_NAME=REPORT_NAME`

Values Any report name.

Default If P_NAME is not specified, the PL/SQL function is populated with the report definition file name.

Usage Notes Specify P_NAME only when you want to use the same report name for each report definition file being registered in Oracle9*i*AS Portal. This argument is typically left blank.

The report name cannot be prefaced with numeric characters (e.g., 401K_ report is an invalid file name and my_401K_report is valid).

A.4.61 P_OWNER

Table A–64 indicates which commands can use the P_OWNER keyword.

Table A–64 Commands that can use **P_OWNER**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	yes	no	no	no

Description P_OWNER is the Oracle9*i*AS Portal schema that owns a report's package, which is created when the report definition files are registered. This keyword is only used when DTYPE=REGISTER.

Syntax `P_OWNER=PORTAL_SCHEMA_NAME`

Values Any valid Oracle9*i*AS Portal schema name.

Default The name of the Oracle9*i*AS Portal schema to which you are logged on when you run the SQL*PLUS script file.

A.4.62 P_PFORMTEMPLATE

Table A–65 indicates the commands that can use the P_PFORMTEMPLATE keyword.

Table A–65 Commands that can use **P_PFORMTEMPLATE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	yes	no	no	no

Description P_PFORMTEMPLATE is the name of the Oracle9*i*AS Portal template that determines the style of the Runtime Parameter Form. This keyword is only used when DTYPE=REGISTER.

Syntax `P_PFORMTEMPLATE=TEMPLATE_NAME`

Values Any valid Oracle9*i*AS Portal template name.

Default none

A.4.63 P_PRINTERS

Table A–66 indicates the commands that can use the P_PRINTERS keyword.

Table A–66 Commands that can use **P_PRINTERS**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	yes	no	no	no

Description P_PRINTERS is the allowable printers for the specified reports. This keyword is only used when DTYPE=REGISTER.

Syntax `P_PRINTERS=(PRT1,PRT2,...)`

Values Any valid printer (e.g., PRT1), or a list of valid printers enclosed by parentheses with a comma separating the names (e.g., (PRT1,PRT2,PRT3)).

Default none

Usage Note Access to the printer(s) should already exist in Oracle9*i*AS Portal before running the SQL*Plus script.

A.4.64 P_PRIVILEGE

Table A–67 indicates which commands can use the `P_PRIVILEGE` keyword.

Table A–67 Commands that can use **P_PRIVILEGE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	yes	no	no	no

Description P_PRIVILEGE is the users or roles who have access privileges to run the specified reports. This keyword is only used when DTYPE=REGISTER.

Syntax `P_PRIVILEGE=(SCOTT,JABERS,PMARTIN,...)`

Values Any user name or role that Oracle9*i*AS Portal can recognize (e.g., SCOTT), or a list of user names or roles enclosed by parentheses with a comma separating the names (e.g., (SCOTT,JABERS,PMARTIN)).

Default none

A.4.65 P_SERVERS

Table A–68 indicates which commands can use the P_SERVERS keyword.

***Table A–68 Commands that can use* P_SERVERS**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	yes	no	no	no

Description P_SERVERS is the names of the restricted Reports Servers that can run the report. This keyword is only used when DTYPE=REGISTER.

Syntax `P_SERVERS=(repserver1,repserver2,...)`

Values Any valid TNS name of the Reports Server (e.g., repserver), or a list of valid Reports Server TNS names enclosed by parentheses with a comma separating the names (e.g., (repserver,acct_server,sales_server)).

Default none

Usage Notes ·Access to the Reports Server(s) should already exist in Oracle9*i*AS Portal.

A.4.66 P_TRIGGER

Table A–69 indicates the commands that can use the P_TRIGGERS keyword.

***Table A–69 Commands that can use* P_TRIGGER**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	yes	no	no	no

Description P_TRIGGER is a PL/SQL function that is executed when parameter values are specified on the command line and when users accept the Runtime Parameter Form. The function must return a boolean value (TRUE or FALSE). For example:

```
P_TRIGGER=Is begin IF UPPER(DESTYPE) = 'PRINTER' AND EMPNAME = 'SMITH' THEN
RETURN(TRUE); ELSE RETURN(FALSE); END IF; end;
```

This keyword is only used when DTYPE=REGISTER.

Syntax P_TRIGGER=PLSQL_FUNCTION

Values Any valid PL/SQL function that returns a boolean value.

Default none

A.4.67 P_TYPES

Table A–70 indicates which commands can use the P_TYPES keyword.

***Table A–70 Commands that can use* P_TYPES**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	yes	no	no	no

Description P_TYPES is the allowable destination types for the specified reports. This keyword is only used when DTYPE=REGISTER.

Syntax P_TYPES=(CACHE,MAIL,...)

Values Any valid destination type (e.g., CACHE), or a list of valid destination types enclosed by parentheses with a comma separating the names (e.g., (CACHE,MAIL,PRINTER)).

Default none

A.4.68 PAGEGROUP

Table A–93 indicates which commands can use the PAGEGROUP keyword.

***Table A–71 Commands that can use* PAGEGROUP**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use PAGEGROUP to specify the name of the Oracle Portal page group to push report output to. For WebDB 2.2, use SITENAME instead. For Oracle Portal 3.0, use CONTENTAREA instead. For Oracle9*i*AS Portal version 2.0 and later, use PAGEGROUP.

The page group must be created in Oracle Portal before you can use this parameter.

Syntax PAGEGROUP="*Name of Oracle Portal page group*"

Values Any valid page group used in Oracle Portal.

Default None

Usage Notes Use of this keyword is required to push Reports output to Oracle Portal. Put quotation marks around the value if the value has any character spaces in it or you are specifying the argument in the cgicmd.dat file.

Relevant keywords include CONTENTAREA, EXPIREDAYS, ITEMTITLE, OUTPUTFOLDER, OUTPUTPAGE, PAGEGROUP, REPLACEITEM, SCHEDULE, SITENAME, STATUSFOLDER, STATUSPAGE.

Oracle Portal objects, such as pages, page groups, and the like, have two names: a display name and an internal name. When you create objects within Oracle Portal that you will use with Oracle Reports output, keep the internal name and the display name the same, following the rules for internal naming specified in the Oracle Portal online help. This way, when you provide a value for a Portal-related keyword in a Reports command line, you will not run into problems with which name to specify.

A.4.69 PAGESIZE

Table A–72 indicates which commands can use the PAGESIZE keyword.

***Table A–72 Commands that can use* PAGESIZE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	yes	yes	yes	yes	no

Description Use PAGESIZE to set the dimensions of the physical page (that is, the size of the page that the printer outputs). The page must be large enough to contain the report. For example, if a frame in a report expands to a size larger than the page dimensions, then the report is not run.

Syntax PAGESIZE=*width* x *height*

Values Any valid page dimensions of the form: page width x page height, where page width and page height are more than zero. The maximum width and height depends which unit of measurement was set in the Reports Builder (**Edit >**

Preferences > General tab). For inches, the maximum width and height is 512 inches. For centimeters, it is 1312 centimeters. For picas, it is 36,864 picas.

Default For bitmap, 8.5 x 11 inches. For character mode, 80 x 66 characters. If the report was designed for character mode and is being run or converted on bitmap, then the following formula is used to determine page size if none is specified: (default page size * character page size) / default character page size. For example, if the character page size is 80 x 20, then the bit-mapped page size would be: ((8.5 * 80) / 80) x ((11 * 20) / 66) = (680 / 80) x (220 / 66) = 8.5 x 3.33.

Usage Notes

- On some printers the printable area of the physical page is restricted. For example, the sheet of paper a printer takes might be 8.5 x 11 inches, but the printer might only be able to print on an area of 8 x 10.5 inches. If you define a page width x page height in the Reports Builder that is bigger than the printable area your printer allows, then clipping might occur in your report output. To avoid clipping, you can either increase the printable area for the printer (if your operating system allows it), or you can set the page width x page height to be the size of the printable area of the page.
- Letter size is 8.5 inches x 11 inches. A4 size is 210mm x 297mm, or 8.25 inches x 11.75 inches.
- If you use the PAGESIZE keyword, then its value overrides the page dimensions of the report definition.
- A PAGESIZE value entered on the Runtime Parameter Form overrides any PAGESIZE value entered on the command line.

A.4.70 PAGESTREAM

Table A–73 indicates which commands can use the PAGESTREAM keyword.

***Table A–73 Commands that can use* PAGESTREAM**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description PAGESTREAM enables or disables page streaming for the report when formatted as HTML or HTMLCSS output, using the navigation controls set by either of the following:

- The Page Navigation Control Type and Page Navigation Control Value properties in the Report Property Palette.
- PL/SQL in a Before Report trigger (SRW.SET_PAGE_NAVIGATION_HTML)

Syntax PAGESTREAM={YES|NO}

Values YES means to stream the pages. NO means to output the report without page streaming.

Default NO

A.4.71 PARAMFORM

Table A–74 indicates which commands can use the PARAMFORM keyword.

***Table A–74 Commands that can use* PARAMFORM**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	no	no	no	yes	yes	no

Description Use PARAMFORM to specify whether to display the Runtime Parameter Form when you execute a report via CGI or a servlet. PARAMFORM is used only to supply parameters to paper layout reports, not Web source reports.

Syntax PARAMFORM=YES|NO|HTML

Values YES means the parameter form should be displayed. NO means the parameter form should not be displayed. HTML means the parameter form should be displayed in HTML format.

Default NO

Usage Notes Do not use this keyword when running a report in an Oracle Portal environment. This is because Oracle Portal allows you to set up a Reports runtime parameter form, which may conflict with a form you specify with the PARAMFORM keyword.

A.4.72 PARSEQUERY

Table A–75 indicates which commands can use the PARSEQUERY keyword.

Table A–75 Commands that can use **PARSEQUERY**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	no	yes	yes	no

Description Use PARSEQUERY to parse an `rwservlet` query and display the constructed Reports Server command line.

Syntax `http://yourwebserver/reports/rwservlet/parsequery[?] [server=servername][&authid=username/password]query_string`

Values See Syntax.

Default None

A.4.73 PDFCOMP

Table A–76 indicates which commands can use the PDFCOMP keyword.

Table A–76 Commands that can use **PDFCOMP**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use PDFCOMP to specify whether PDF output should be compressed.

Syntax `PDFCOMP={any value 0 through 9) OR {YES|NO}`

Values Any value 0 though 9 or YES (6) or NO (0). A value of 0 means PDF output will not be compressed. A value of 1 through 9 will compress the PDF output and permit users to control the compression level. A value of YES equals compression level 6. A value of NO means compression level 0.

Default 6

A.4.74 PDFEMBED

Table A–77 indicates which commands can use the PDFEMBED keyword.

Table A–77 Commands that can use **PDFEMBED**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use PDFEMBED to specify whether Reports will embed the Type1 Postscript font file(s) specified in uifont.ali into PDF output.

Syntax `PDFEMBED={YES|NO}`

Values YES means that the PDF driver will embed the font(s) specified in the PDFEMBED parameter of the uifont.ali file into the PDF output. NO means that the font(s) will not be added to PDF output.

Default YES

A.4.75 PRINTJOB

Table A–78 indicates which commands can use the PRINTJOB keyword.

Table A–78 Commands that can use **PRINTJOB**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	yes	no	no	no	no

Description Use PRINTJOB to specify whether the Print Job dialog box should be displayed before running a report.

Syntax `PRINTJOB={YES|NO}`

Values YES or NO

Default NO

Usage Notes

- When a report is run as a spawned process (that is, one executable, such as RWRUN, is called from within another executable, such as RWBUILDER), the Print Job dialog box does not appear, regardless of PRINTJOB.
- When DESTYPE=MAIL, the Print Job dialog box does not appear, regardless of PRINTJOB.

A.4.76 READONLY

Table A–79 indicates which commands can use the READONLY keyword.

***Table A–79 Commands that can use* READONLY**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	yes	no	yes	yes	no

Description Use READONLY to request read consistency across multiple queries in a report. When accessing data from Oracle, read consistency is accomplished by a SET TRANSACTION READ ONLY statement.

> **Note:** Refer to *Oracle9i* SQL documentation (available on the Oracle Technology Network: http://otn.oracle.com) for more information on SET TRANSACTION READ ONLY.

Syntax `READONLY={YES|NO}`

Values YES requests read consistency. NO means do not provide read consistency.

Default NO

Usage Notes

- This keyword is only useful for reports using multiple queries. Oracle automatically provides read consistency, without locking, for single query reports.
- In the Report trigger order of execution, SET TRANSACTION READ ONLY must be set up before the data fetch occurs.
- READONLY can also be used with jobs run as JSPs.

A.4.77 REPLACEITEM

Table A–80 indicates which commands can use the REPLACEITEM keyword.

***Table A–80 Commands that can use* REPLACEITEM**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use REPLACEITEM to specify that the current report output being pushed to Oracle Portal should replace an earlier version of the same item stored in the same output target.

Syntax `REPLACEITEM={YES|NO}`

Values YES specifies that earlier report output should be replaced. NO means do not replace previous version. In this case, a link to the new output will be added to a list of links to previous versions of the same report.

Default None

Usage Notes Use of this keyword is optional. Relevant keywords include CONTENTAREA, EXPIREDAYS, ITEMTITLE, OUTPUTFOLDER, OUTPUTPAGE, PAGEGROUP, REPLACEITEM, SCHEDULE, SITENAME, STATUSFOLDER, STATUSPAGE.

A.4.78 REPLYTO

Table A–81 indicates which commands can use the REPLYTO keyword.

***Table A–81 Commands that can use* REPLYTO**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use REPLYTO to specify the e-mail address to which replies should be sent when the sender wants replies to go to someone other than the sender (specified by the FROM keyword).

Syntax `REPLYTO=someone@foo.com`

Values Any valid e-mail address.

Default None

Usage Notes Related keywords include BCC, CC, FROM, REPLYTO, and SUBJECT. Note that DESNAME is used to specify the main recipient(s) of the e-mail.

A.4.79 REPORT|MODULE

See MODULE | REPORT.

A.4.80 ROLE

Table A–82 indicates which commands can use the ROLE keyword.

***Table A–82 Commands that can use* ROLE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use ROLE to specify the database role to be checked for the report at runtime.

Syntax `ROLE={rolename[/rolepassword]}`

Values A valid role and (optionally) a role password.

Default None

Usage Notes ROLE can also be used with jobs run as JSPs.

A.4.81 RUNDEBUG

Table A–83 indicates which commands can use the RUNDEBUG keyword.

***Table A–83 Commands that can use* RUNDEBUG**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	yes	no	yes	yes	no

Description Use RUNDEBUG to turn on error messages / warnings that would otherwise not be displayed. For example, with RUNDEBUG=YES, you might get the error message: Frame 1 overlaps but does not contain Frame 2. This situation may or may not be acceptable, depending on the job being run.

Syntax `RUNDEBUG={YES|NO}`

Values YES means display additional error / warning messages. NO means do not display additional error / warning messages.

Default YES

Usage Notes RUNDEBUG can also be used with jobs run as JSPs.

A.4.82 SAVE_RDF

Table A–84 indicates which commands can use the SAVE_RDF keyword.

***Table A–84 Commands that can use* SAVE_RDF**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	yes	yes	no	no	no	no

Description Use SAVE_RDF to specify a filename for a combined RDF file and XML customization file. This keyword is useful when you combine an existing RDF file with a Reports XML customization file using the CUSTOMIZE keyword, and you wish to save the combination to a new RDF file.

Syntax SAVE_RDF=*filename*.rdf

Values Any valid file name.

Default None

Usage Notes You can use SAVE_RDF with a JSP file, but only the paper layout part, not the Web source.

A.4.83 SCHEDULE

Table A–85 indicates which commands can use the SCHEDULE keyword.

***Table A–85 Commands that can use* SCHEDULE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	no	no	no	yes	yes	no

Description Use SCHEDULE to set the day, time, and frequency a report should be run. The default is to run the report once, now. Time values are expressed

according to a 24-hour day (i.e., one o'clock is expressed 13:00). To eliminate the need for quoting the scheduling command, use underscores (_) instead of spaces. For example, use:

```
SCHEDULE=every_first_fri_of_month_from_15:53_Oct_23,_1999_retry_3_after_1_hour
SCHEDULE=last_weekday_before_15_from_15:53_Oct_23,_1999_retry_after_1_hour
```

Or:

```
SCHEDULE="every first fri of month from 15:53 Oct 23, 1999 retry 3 after 1 hour"
SCHEDULE="last weekday before 15 from 15:53 Oct 23, 1999 retry after 1 hour"
```

Syntax `SCHEDULE=`*`string`*

where the *string* is:

[*FREQ* from] *TIME* [retry {*n*} + after *LEN*]

Table A–86 lists and explains the values used in this string.

Table A–86 Values for string used with the* SCHEDULE *keyword

FREQ	hourly | daily | weekly | monthly | {every LEN | *DAYREPEAT*}} | {last {*WEEKDAYS* | weekday | weekend} before {*n*}+}
LEN	{*n*}+ {minute[s] | hour[s] | day[s] | week[s] | month[s]}
DAYREPEAT	{first | second | third | fourth | fifth} *WEEKDAYS* of month
WEEKDAYS	mon | tue | wed | thu | fri | sat | sun
TIME	now | *CLOCK* [*DATE*]
CLOCK	h:m | h:mm | hh:m | hh:mm
DATE	today | tomorrow | {*MONTHS* {*d* | *dd*} [,year]}
MONTHS	jan | feb | mar | apr | may | jun | jul | aug | sep | oct | nov | dec

Default None

A.4.84 SERVER

Table A–87 indicates which commands can use the SERVER keyword.

Table A–87 Commands that can use **SERVER**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	no	no	no	yes	yes	yes

Description Use SERVER to specify the name of the Reports Server you want to use to run this report.

Syntax `SERVER=servername`

Values The server name or a TNS service entry name if you're using a 6*i* compatible server.

Default The server name specified in the REPORTS_SERVER environment variable for `rwcgi`.

Usage Notes For jobs run with `rwcgi`, you can set the REPORTS_SERVER environment variable on your Web server machine and omit the SERVER keyword to process requests using the default server, or you can include the SERVER keyword to override the default. For jobs run with `rwservlet` or as a JSP, you can omit the SERVER keyword if you have specified a default server in the servlet configuration file, rwservlet.properties; or you can include the SERVER keyword to override the default.

SERVER can also be used with jobs run as JSPs.

A.4.85 SHOWENV

Table A–88 indicates which commands can use the SHOWENV keyword.

Table A–88 Commands that can use **SHOWENV**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	no	yes	yes	no

Description Use SHOWENV to display the `rwserver` configuration file. (rwservlet.properties).

Syntax `http://yourwebserver/reports/rwservlet/showenv[?]`
`[server=servername][&authid=username/password]`

Values See Syntax.

Default None

A.4.86 SHOWJOBS

Table A–89 indicates which commands can use the SHOWJOBS keyword.

***Table A–89 Commands that can use* SHOWJOBS**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	no	yes	yes	no

Description Use SHOWJOBS to display a Web view of Reports Server queue status.

Syntax `http://`*`yourwebserver`*`/`*`reports`*`/rwservlet/showjobs[`*`n`*`][?]`
`[server=`*`server_name`*`][&authid=`*`username/password`*`][&statusformat={html|xml|xmldtd}]`

Values See Syntax.

Default None

Usage Notes The name of the Reports Server must be specified implicitly by environment variable or servlet configuration file, or explicitly in the URL request. The refresh number [*n*] is optional. When it is specified, the report's queue status will be updated every [*n*] seconds. The STATUSFORMAT can be set to html, xml, or xmldtd to return status in that format. The default is html.

Related keywords are SERVER, AUTHID, and STATUSFORMAT.

The STATUSFORMAT parameter is only valid for `rwservlet`, not for `rwcgi`.

A.4.87 SHOWMAP

Table A–90 indicates which commands can use the SHOWMAP keyword.

***Table A–90 Commands that can use* SHOWMAP**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	no	yes	yes	no

Description Use SHOWMAP to display `rwservlet` key mappings.

Syntax `http://yourwebserver/reports/rwservlet/showmap[?] [server=servername][&authid=username/password]`

Values See Syntax.

Default None

A.4.88 SHOWMYJOBS

Table A–89 indicates which commands can use the SHOWMYJOBS keyword.

***Table A–91 Commands that can use* SHOWMYJOBS**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	no	yes	no	no

Description Use SHOWMYJOBS to display the Reports Server queue status for a particular user.

Syntax `http://yourwebserver/reports/rwservlet/showmyjobs[?] [server=server_name][&authid=username/password][&statusformat={html|xml|xmldtd}]`

Values See Syntax.

Default None

Usage Notes The STATUSFORMAT can be set to html, xml, or xmldtd to return status in that format. The default is html.

Related keywords are SERVER, AUTHID, and STATUSFORMAT.

The STATUSFORMAT parameter is only valid for `rwservlet`, not for `rwcgi`.

A.4.89 SHUTDOWN

Table A–92 indicates which commands can use the SHUTDOWN keyword.

***Table A–92 Commands that can use* SHUTDOWN**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	no	no	no	yes

Description Use SHUTDOWN to shut down a previously running server. You must also use AUTHID to supply a user name and password.

Syntax `SHUTDOWN={NORMAL|IMMEDIATE}`

Values NORMAL or IMMEDIATE

Default NORMAL

Usage Notes The user of the SHUTDOWN keyword must be a Reports Administrative user. If the server has security enabled, it will query the security API to determine the user's role eligibility to execute the shutdown (in other words, the user must be a Reports Administrative user). If security is not enabled, then the user must nonetheless be a Reports Administrative user defined for that server.

A.4.90 SITENAME

Table A–93 indicates which commands can use the SITENAME keyword.

***Table A–93 Commands that can use* SITENAME**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use SITENAME to specify the name of the site to push report output to. For Oracle Portal 3.0 users this is the content area name. (See also CONTENTAREA.) This keyword is maintained for backward compatibility with earlier versions of Oracle Portal (and WebDB). For Oracle9*i*AS Portal version 2.0 and later, use PAGEGROUP.

Syntax `SITENAME=`*`sitename`*

Values Any valid site name used in Oracle Portal.

Default None

Usage Notes Use of this keyword is required to push Reports output to Oracle Portal. Put quotation marks around the value if the value has any character spaces in it or you are specifying the argument in the cgicmd.dat file.

Relevant keywords include CONTENTAREA, EXPIREDAYS, ITEMTITLE, OUTPUTFOLDER, OUTPUTPAGE, PAGEGROUP, REPLACEITEM, SCHEDULE, SITENAME, STATUSFOLDER, STATUSPAGE.

Oracle Portal objects, such as pages, page groups, and the like, have two names: a display name and an internal name. When you create objects within Oracle Portal that you will use with Oracle Reports output, keep the internal name and the display name the same, following the rules for internal naming specified in the Oracle Portal online help. This way, when you provide a value for a Portal-related keyword in a Reports command line, you will not run into problems with which name to specify.

A.4.91 SOURCE

Table A–94 indicates which commands can use the SOURCE keyword.

***Table A–94 Commands that can use* SOURCE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	yes	no	no	no

Description Use SOURCE to specify the report/library or list of reports/libraries to be converted. The `rwconverter` command requires that you specify a source report or library.

Syntax `SOURCE={source_name|(source_name1, source_name2, …)}`

Values Any valid report/library name or filename, or a list of valid report/library names or filenames enclosed in parentheses and separated by commas (e.g., `(qanda, test, dmast)`).

Default None

Usage Notes

- SQL wildcard characters (% and _) may be used for reports or libraries that are stored in the database. For example, R% would fetch all reports stored in the database that begin with R. All reports that match will be converted.
- A list of report/library names or filenames must be enclosed in parentheses, with commas separating the names. For example:

  ```
  (qanda,test,dmast) OR (qanda, test, dmast)
  ```

- Wildcard characters are invalid for reports/libraries stored in files (i.e., with extensions of rdf, rep, rex, pld, pll, or xml).
- The value(s) for the SOURCE keyword may be operating system-specific.
- If you are using user-owned Reports Builder tables, reports/libraries from multiple users must be converted for each user individually.
- To convert reports/libraries, you must have created them or been granted access to the ones you did not create. If no userid is prefixed to the report/library name, the userid is assumed to be the current user.
- When DTYPE=REGISTER, you may only want to list report definition files with common parameters, such as destination types and formats, user access, and availability calendars.

A.4.92 SSOCONN

Table A–95 indicates which commands can use the SSOCONN keyword.

***Table A–95 Commands that can use* SSOCONN**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	no	yes	no	no

Description Use SSOCONN to specify one or more connect strings to use to connect to one or more data sources in a single sign-on environment.

Syntax `SSOCONN=key[/type[/conn_str]][,key[/type[/conn_str]]]`

For example:

```
ssoconn=mykey/OracleDB/userid
```

Values The following information describes the variable values expressed in the SSOCONN syntax:

- *key* refers to a connection string value stored in the Oracle Internet Directory (OID).
- *type* is a predefined character string that specifies a Reports data source type. Types provided with your Oracle9*i*AS installation include OracleDB, JDBCPDS (Java Database Connectivity Pluggable Data Source), and ExpressPDS (Oracle Express Pluggable Data Source).

- *conn_str* is the name of the connection string parameter that `rwservlet` will compose to run the report

Default None

Usage Notes

- If multiple data sources are used in the report, use a comma to separate data source connection strings. For example:

  ```
  ssoconn=key1/type1/conn_str,key2/type2/conn_str2,key3/type3/conn_str3
  ```

- Simplified versions of the SSOCONN argument are also available. Table A–96 provides examples.

Table A–96 Simplified versions of the* SSOCONN *argument

Argument	Description
ssoconn=mkey	When only the key name is specified, the default type (Oracle DB) will be used, and the connect parameter is the USERID keyword.
ssoconn=mkey/PDSApp	When both key name and application type are specified, the connection parameter is the USERID keyword.
ssoconn=mkey/PDSApp/P_1	When everything is specified, the specified values are used.

- SSOCONN can also be used with jobs run as JSPs.

Note: For more information about Reports and single sign-on (SSO), see Chapter 7, "Data Source Single Sign-On".

A.4.93 STATUSFORMAT

Table A–89 indicates which commands can use the STATUSFORMAT keyword.

***Table A–97 Commands that can use* STATUSFORMAT**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	no	yes	no	no

Description Use STATUSFORMAT to specify the format for the Web view of Reports Server queue status.

Syntax `Reports_URL/rwservlet/showjobs?server=server_name&statusformat={html|xml|xmldtd}`

Values HTML, XML, or XMLDTD. Use HTML to specify that the Reports queue status output should be in HTML format. Use XML to specify that it should be in XML format. Use XMLDTD to specify that it should be in XML format with in-line Data Type Definition information.

Default HTML

Usage Notes Use STATUSFORMAT in conjunction with the SHOWJOBS and SHOWMYJOBS keywords.

A.4.94 STATUSFOLDER

Table A–98 indicates which commands can use the STATUSFOLDER keyword.

***Table A–98 Commands that can use* STATUSFOLDER**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use STATUSFOLDER to specify the folder to push status information into. If this is omitted, a new folder is created called "Oracle_Reports_Status." This value is retained for backward compatibility with earlier versions of Oracle Portal. For the current version (Oracle9*i*AS Portal version 2.0 and above, see STATUSPAGE.)

Syntax `STATUSFOLDER=Oracle_Reports_Status`

Values Any valid folder name used in Oracle Portal.

Default Oracle_Reports_Status

Usage Notes The value for this keyword is case sensitive. Use of this keyword is optional.

Put quotation marks around the value if the value has any character spaces in it or you are specifying the argument in the cgicmd.dat file.

Oracle Portal objects, such as pages, page groups, and the like, have two names: a display name and an internal name. When you create objects within Oracle Portal that you will use with Oracle Reports output, keep the internal name and the display name the same, following the rules for internal naming specified in the Oracle Portal online help. This way, when you provide a value for a Portal-related keyword in a Reports command line, you will not run into problems with which name to specify.

A.4.95 STATUSPAGE

Table A–99 indicates which commands can use the STATUSPAGE keyword.

***Table A–99 Commands that can use* STATUSPAGE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use STATUSPAGE to specify the page to push job status information into. If this is omitted, a new page is created called "Oracle_Reports_Status." Use this keyword with Oracle9*i*AS Portal version 2.0 and later. For backward compatibility, see STATUSFOLDER.

Syntax `STATUSPAGE=Oracle_Reports_Status`

Values Any valid page name used in Oracle Portal.

Default Oracle_Reports_Status

Usage Notes The value for this keyword is case sensitive. Use of this keyword is optional.

Put quotation marks around the value if the value has any character spaces in it or you are specifying the argument in the cgicmd.dat file.

Oracle Portal objects, such as pages, page groups, and the like, have two names: a display name and an internal name. When you create objects within Oracle Portal that you will use with Oracle Reports output, keep the internal name and the display name the same, following the rules for internal naming specified in the Oracle Portal online help. This way, when you provide a value for a Portal-related keyword in a Reports command line, you will not run into problems with which name to specify.

A.4.96 STYPE

Table A–100 indicates which commands can use the STYPE keyword.

***Table A–100 Commands that can use* STYPE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	yes	no	no	no

Description Use STYPE to specify the format of the report(s) or libraries to be converted.

Syntax `STYPE={PLDFILE|PLLFILE|RDFFILE|REXFILE|XMLFILE|JSPFILE}`

Values Use any one of the following values:

- PLDFILE means the source PL/SQL libraries are stored in files in ASCII format.
- PLLFILE means the source PL/SQL libraries are stored in files containing source code and P-code (compiled PL/SQL).
- RDFFILE means the source report(s) are stored in one or more report definition files (files with the rdf extension).
- REXFILE means the source report(s) are stored in one or more text files (files with the rex extension).
- XMLFILE means the source report(s) are stored in one or more XML files.
- JSPFILE means the source report(s) are stored in one or more JSP files.

Default RDFFILE

Usage Notes When DTYPE=REGISTER, choose RDDFILE, REXFILE, XML, or JSPFILE for STYPE.

A.4.97 SUBJECT

Table A–101 indicates which commands can use the SUBJECT keyword.

***Table A–101 Commands that can use* SUBJECT**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	no	no	yes	yes	no

Description Use SUBJECT to specify the subject line of an e-mail.

Syntax SUBJECT="*any text*"

Values Any text.

Default None

Usage Notes Enclose subjects that contain character spaces in quotation marks. Single-word subjects do not require quotation marks.

Related keywords include BCC, CC, FROM, REPLYTO, and SUBJECT. Note that DESNAME is used to specify the main recipient(s) of the e-mail.

A.4.98 TOLERANCE

Table A–102 indicates which commands can be used with the TOLERANCE keyword.

***Table A–102 Commands that can use* TOLERANCE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	no	no	no	yes	yes	no

Description Use TOLERANCE to set the maximum acceptable time (in minutes) for reusing a report's cached output when a duplicate job is detected. Setting the time tolerance on a report reduces the processing time when duplicate jobs are found.

See Section 8.8, "Reusing Report Output from Cache" (in Chapter 8) for more information on duplicate job detection.

Syntax TOLERANCE=*time_string*

Values The time string can be in one of two formats:

- *n*{*unit*}, for a number with an optional unit. The unit can be minute(s), hour(s), or day(s). The default unit is minute(s) if no unit is specified.
- {*Mon DD, YYYY*} *hh*:*mi*:*ss* am|pm {*timezone*}, for a date/time format. Date information is optional. If it isn't specified, *today* is assumed. Time zone is also optional. If it isn't specified, the Reports Server's timezone is used. The date/time is always in a US locale. This format is the same as defined in the Java DateFormat.MEDIUM type.

Default None

Usage Notes

- If TOLERANCE is not specified, then Oracle9*i*AS Reports Services reruns the report even if a duplicate report is found in the cache.
- If a report is being processed (that is, in the current job queue) when an identical job is submitted, then Oracle9*i*AS Reports Services reuses the output of the currently running job even if TOLERANCE is not specified or is set to zero.

A.4.99 TRACEFILE

Table A–103 indicates which commands can use the `TRACEFILE` keyword.

***Table A–103 Commands that can use* `TRACEFILE`**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	yes	yes	no	no	no	no

Description TRACEFILE is the name of the file in which trace information is logged.

> **Note:** In a runtime environment, or when you are monitoring Reports Services components, you can use all three TRACE keywords. But for server security, in a server environment you can use only TRACEOPTS. This is to prevent files from being written arbitrarily to the Reports Server's file system.
>
> Tracing for the Reports Server is configured in the server configuration file, <server_name>.conf (see Chapter 3). Tracing for the Reports Servlet is configured in the servlet configuration file, rwservlet.properties (see Chapter 3). Tracing for individual jobs is specified from the runtime command line, via the TRACEOPTS command line argument.

Syntax `TRACEFILE=`*`tracefile`*

Values Any valid file name.

Default None

Usage Notes See Section 14.5.1.1, "Trace Overview" for additional information about how tracing works with Reports.

A.4.100 TRACEMODE

Table A–104 indicates which commands can use the TRACEMODE keyword.

***Table A–104 Commands that can use* TRACEMODE**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	yes	no	yes	yes	no

Description TRACEMODE indicates whether new trace information should be appended to existing information in a trace file or overwrite the entire file.

Syntax `TRACEMODE={TRACE_APPEND|TRACE_REPLACE}`

Values TRACE_APPEND adds the new information to the end of the file. TRACE_REPLACE overwrites the file.

Default TRACE_APPEND

Usage Notes See Section 14.5.1.1, "Trace Overview" for additional information about how tracing works with Reports.

A.4.101 TRACEOPTS

Table A–105 indicates which commands can use the TRACEOPTS keyword.

***Table A–105 Commands that can use* TRACEOPTS**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	yes	no	yes	yes	yes

Description TRACEOPTS indicates the tracing information that you want to be logged in the trace file when you run the report.

Syntax `TRACEOPTS={TRACE_ERR|TRACE_PRF|TRACE_APP|TRACE_PLS|TRACE_SQL|TRACE_TMS|TRACE_DST|TRACE_ALL|TRACE_EXC|(TRACE_ERR, TRACE_PLS, …)}`

Values The following values apply:

- A list of options in parentheses means you want all of the enclosed options to be used. For example, `TRACE_OPTS=(TRACE_APP, TRACE_PRF)` means you want TRACE_APP and TRACE_PRF applied.
- TRACE_ALL logs all possible trace information in the trace file.
- TRACE_APP logs trace information on all the report objects in the trace file.
- TRACE_BRK lists all breakpoints in the trace file.
- TRACE_DBG logs debug information.
- TRACE_DST lists distribution lists in the trace file. You can use this information to determine which section was sent to which destination.
- TRACE_ERR lists error messages and warnings in the trace file.
- TRACE_EXC lists Reports Server exceptions.
- TRACE_INF is a catch-all option that dumps any information not covered by the other options into the trace file/
- TRACE_LOG duplicates log information in your trace file. If you have specified a *log* element, in addition to a *trace* element, in your server configuration file, this value will cause information that is sent to the log file to also be sent to the trace file.
- TRACE_PLS logs trace information on all the PL/SQL objects in the trace file.
- TRACE_PRF logs performance statistics in the trace file.
- TRACE_SQL logs trace information on all the SQL in the trace file.
- TRACE_STA provides server and engine state information, such as initialize, ready, run, and shut-down.
- TRACE_TMS enters a timestamp for each entry in the trace file.
- TRACE_WRN lists server warning messages.

Default TRACE_ALL

Usage Notes `TRACEOPTS` can also be used with jobs run as JSPs.

See Section 14.5.1.1, "Trace Overview" for additional information about how tracing works with Reports.

A.4.102 UNINSTALL

Table A–106 indicates which commands can use the UNINSTALL keyword

Table A–106 Commands that can use UINSTALL

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	no	no	no	no	yes

Description Use UNINSTALL to remove an instance of the Reports Server from Microsoft Windows. This process removes the Reports Server service from the system. This keyword is ignored on UNIX.

Syntax `UNINSTALL REPORTS_SERVER_NAME`

Values The name of an existing Reports Server instance

Default none

Usage Notes If you use BATCH=YES with INSTALL, then none of the prompts and dialogs that normally display during the removal process will appear.

A.4.103 URLPARAMETER

Table A–107 indicates which commands can use the URLPARAMETER keyword.

Table A–107 Commands that can use **URLPARAMETER**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	no	no	no	yes	yes	no

Description Use URLPARAMETER to specify the URL that is to be fetched with the URL engine.

Syntax `URLPARAMETER=http://webserver_name/pagename.html`

Values Any valid URL.

Default None

Usage Notes This keyword is relevant when the *jobType* parameter of the *job* element in the Reports Server configuration file is `rwurl`, and a URL engine is in place.

A.4.104 USERID

Table A–108 indicates which commands can use the `USERID` keyword.

***Table A–108 Commands that can use* `USERID`**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
yes	yes	yes	yes	yes	yes	no

Description Use `USERID` only if you're not using single sign-on. Use `USERID` to specify your Oracle user name and password, with an optional database name for accessing a remote database. If the password is omitted, then a database logon form opens automatically before the user is allowed to run the report.

If you want users to log on to the database, then omit the password portion of the `USERID` keyword from the report request. If you want users to log on every time they run report requests, then use the Reports key mapping file, cgicmd.dat, to specify the runtime command, and include the %D argument in the relevant key mapping entry.

Note: For information on using the cgicmd.dat file, see Chapter 8, "Running Report Requests".

Syntax `userid=`*`username`*`[/`*`password`*`][@`*`database`*`]`

Values The logon definition must be in one of the following forms and cannot exceed 512 bytes in length:

`username``[/`*`password`*`]`
`username``[/`*`password`*`][@`*`database`*`]`

Default None

Usage Notes `USERID` can also be used with jobs run as JSPs.

A.4.105 WEBSERVER_DEBUG

Table A–109 indicates which commands can use the WEBSERVER_DEBUG keyword.

***Table A–109 Commands that can use* WEBSERVER_DEBUG**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	yes	no	no	no	no

Description Use WEBSERVER_DEBUG for JSP debugging. It creates the `stderr.log` and `stdout.log` files under the *`docroot/port#`* directory, and leaves JSP temporary files under *`docroot/port#/`*`default` and log files under *`docroot/port#/`*`log` for your inspection.

Syntax `WEBSERVER_DEBUG={YES|NO}`

Values Yes means create debugging files. No means do not create debugging files.

Default NO

Usage Notes Use this keyword only when you're running a job as a JSP. Relevant keywords include WEBSERVER_DEBUG, WEBSERVER_DOCROOT, WEBSERVER_PORT.

A.4.106 WEBSERVER_DOCROOT

Table A–110 indicates which commands can use the WEBSERVER_DOCROOT keyword.

***Table A–110 Commands that can use* WEBSERVER_DOCROOT**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	yes	no	no	no	no

Description Use WEBSERVER_DOCROOT to set the Reports Builder document root directory. All files you reference in your JSP, such as images, HTML, and the like, should be relative to this directory. By setting the document root to your working directory, you avoid having to copy these files around.

Syntax `WEBSERVER_DOCROOT=`*`REPORTS_TMP/docroot`*

For example:

```
WEBSERVER_DOCROOT=c:/temp/docroot
```

Values The directory to the document root folder in your Reports temporary folder.

Default None

Usage Notes Use this keyword only when you're running a job as a JSP. Relevant keywords include WEBSERVER_DEBUG, WEBSERVER_DOCROOT, WEBSERVER_PORT.

A.4.107 WEBSERVER_PORT

Table A–111 indicates which commands can use the WEBSERVER_PORT keyword.

***Table A–111 Commands that can use* WEBSERVER_PORT**

rwclient	rwrun	rwbuilder	rwconverter	rwservlet	rwcgi	rwserver
no	no	yes	no	no	no	no

Description Use WEBSERVER_PORT to specify the port number an internal Web server listens to. You can specify a port number (e.g., 3002) or a range of port numbers (e.g., 3100-3200). If a single port number is specified, Reports tries to start the internal Web server listening on that port. If that port is in use, it tries to get the next available port. If a range of port numbers is specified, Reports tries to look for a free port in that range.

Syntax WEBSERVER_PORT=*port number or range of numbers*

Values Any valid port number or range of port numbers.

Default The default port is 3000. The default range of ports is 3000-3010.

Usage Notes Use this keyword only when you're running a job as a JSP. Relevant keywords include WEBSERVER_DEBUG, WEBSERVER_DOCROOT, WEBSERVER_PORT.

B

Reports-Related Environment Variables

Environment variables are parameters that configure the environment that hosts Oracle9*i*AS Reports Services. The Oracle9*i*AS installer automatically defines default values for relevant environment variables. If you want something other than the default environment, you can edit the environment variable settings:

- For Windows NT, edit environment variables through the Registry Editor (**Start** > **Run** > **Regedit**).
- For UNIX, edit environment variables by revising and running the shell script that defines the initial default values (`reports.sh`). If you do this, be sure to keep a backup of the original, unaltered `reports.sh` file.

Table B–1 lists and describes the environment variables that pertain to Oracle9*i*AS Reports Services.

> **Note:** For more information on all NLS environment variables, see the *Oracle9i Globalization Support Guide* on the Oracle Technology Network (http://otn.oracle.com).

Table B–1 Environment variables relevant to Oracle9iAS Reports Services

Variable	Description
DEVELOPER_NLS_LANG	The language for the Oracle9*i*DS Reports Builder. Chapter 12, "NLS and Bidirectional Support" contains additional detailed information about this environment variable, including a table of valid values.
NLS_CALENDAR	The calendar system used.

Table B–1 Environment variables relevant to Oracle9iAS Reports Services

Variable	Description
NLS_CREDIT	The string used to indicate a positive monetary value.
NLS_CURRENCY	The local currency symbol.
NLS_DATE_FORMAT	The default format mask used for dates.
NLS_DATE_LANGUAGE	The default language used for dates.
NLS_DEBIT	The string used to indicate a negative memory value.
NLS_ISO_CURRENCY	The ISO currency symbol.
NLS_LANG	The language settings used by Oracle9*i*AS Reports Services. Chapter 12, "NLS and Bidirectional Support" contains additional detailed information about this environment variable, including a table of valid values.
NLS_LIST_SEPARATOR	The character used to separate items in a list.
NLS_MONETARY_CHARACTERS	The decimal character and thousands separator for monetary values.
NLS_NUMERIC_CHARACTERS	The decimal character and grouping separator for numeric values.
NLS_SORT	The type of sort used for character data.
REPORTS_COOKIE_EXPIRE	(`rwcgi` only) Determines the expire time of the cookie in minutes. The default value is 30. Cookies save encrypted user names and passwords on the client-side when users log on to a secured Oracle9*i*AS Reports Server to run report requests. When users successfully log on, their browser is sent an encrypted cookie. When a cookie expires, users must re-authenticate to run subsequent requests.
REPORTS_DB_AUTH	(`rwcgi` only) Specifies the database authentication template to be used for logging on to the database. The default value is `dbauth.htm`.
REPORTS_ENCRYPTION_KEY	(`rwcgi` only) Specifies the encryption key used to encrypt the user name and password for the cookie. The encryption key can be any character string. The default value is `reports9.0`.

Table B–1 Environment variables relevant to Oracle9iAS Reports Services

Variable	Description
REPORTS_CGIDIAGBODYTAGS	(`rwcgi` only) For the Oracle9*i*AS Reports CGI. Specifies HTML tags that are inserted as a <BODY…> tag in the RWCGI diagnostic and debugging output. For example, you might want to use this environment variable to set up text and background color or image. Reports CGI is maintained for backward compatibility only.
REPORTS_CGIDIAGHEADTAGS	(`rwcgi` only) For the Oracle9*i*AS Reports CGI. Specifies HTML tags to insert between <HEAD> …</ HEAD> tags in the RWCGI diagnostic and debugging output. For example, you might want to use this environment variable to set up <TITLE> or <META> tags. Reports CGI is maintained for backward compatibility only.

REPORTS_CGIHELP	(`rwcgi` only) For the Oracle9*i*AS Reports CGI. Defines URL and URI of the RWCGI help file that should display when RWCGI is invoked with the following empty request:
	
	`http://<your_webserver>/RWCGI90?`
	
	For example, setting it to `http://www.yahoo.com` goes to that URL; setting it to `myhelpfile.htm` displays the following file:
	
	`http://<your_webserver>/myhelpfile.htm`
	
	If this parameter is not defined, then a default help screen is displayed.
	
	Reports CGI is maintained for backward compatibility only.

REPORTS_CGIMAP	(`rwcgi` only) For the Oracle9*i*AS Reports CGI. Defines fully qualified file name and location of the RWCGI map file if map file configuration is used. For example: *ORACLE_HOME*`\reports\conf\cgicmd.dat` Reports CGI is maintained for backward compatibility only.

Table B–1 Environment variables relevant to Oracle9iAS Reports Services

Variable	Description
REPORTS_CGINODIAG	(`rwcgi` only) Set to YES or NO. For the Oracle9*i*AS Reports CGI. When defined, disables all debugging and diagnostic output, such as help and showmap, from RWCGI. For example, the following request does not work when REPORTS_CGINODIAG is defined: `http://<your_webserver>/rwcgi/help?` Reports CGI is maintained for backward compatibility only.
REPORTS_PATH	Specifies the directories in which Reports components will automatically search for any files they require. Separate directories with semicolons or colons, depending upon the platform. If you specify a path for the *sourceDir* attribute of the *engine* element in the Reports Server configuration file (`<server_name>.conf`), the *sourceDir* value will override the values you set here.
REPORTS_SERVER	(`rwcgi` only) Specifies the default Oracle9*i*AS Reports Server. When this environment variable is set, you can omit the SERVER command line argument in report requests if you want to process them with the default server. Conversely, you can include the SERVER command line argument to override the default you specify here.
REPORTS_SSLPORT	(`rwcgi` only) If you are using SSL and you want to use a port number other than 443, then you can use this variable to set a different port number.The default value is 443.
REPORTS_SYS_AUTH	(`rwcgi` only) Specifies the authentication template for displaying the user name and password request dialog when users run report request to a restricted Oracle9*i*AS Reports Server.
REPORTS_TMP	Specifies the temporary directory where Reports development- and server-related temporary files will automatically (and temporarily) be stored.

Table B–1 Environment variables relevant to Oracle9iAS Reports Services

Variable	Description
RW	Specifies the reports-specific directory within the Oracle Home. For example: Windows: `RW=d:\ORACLE_HOME\reports` UNIX: `RW=ORACLE_HOME/reports`
USER_NLS_LANG	The language for the Oracle9*i*AS Reports Runtime component. Chapter 12, "NLS and Bidirectional Support" contains additional detailed information about this environment variable, including a table of valid values.

C

Batch Registering Reports in Oracle9*i*AS Portal

If you have a number of reports that you wish to register in Oracle9*i*AS Portal, it is often preferable to register them as a group in a batch script rather than individually in the Oracle9*i*AS Portal user interface. Likewise, if you have a large number of reports that you wish to unregister, a batch script is more efficient.

- Batch Registering Report Definition Files
- Batch Removing Report Packages
- PL/SQL Batch Registering Function

C.1 Batch Registering Report Definition Files

To batch register reports in Oracle9*i*AS Portal, you need to perform the following steps:

1. Run RWCONVERTER to Generate a SQL Script
2. Run the Script in SQL*Plus

C.1.1 Run RWCONVERTER to Generate a SQL Script

To generate a SQL script that you can execute in SQL*Plus to register your reports, do the following:

1. From the operating system prompt (DOS or UNIX), enter the RWCONVERTER command with the keywords to batch register the report definition files. For a

description of RWCONVERTER keywords, refer to Appendix A, "Command Line Arguments".

> **Note:** To successfully create a script file with the necessary load functions, you specify the DTYPE, STYPE, SOURCE, and DEST arguments. To create a functional package in Oracle9*i*AS Portal, you will need to specify the P_SERVERS, P_PRIVILEGE, P_TYPES, P_FORMATS in addition to the arguments used to create the script file.

Following is an example RWCONVERTER command line on Microsoft Windows:

```
rwconverter.exe dtype="register" stype="rdffile"
source="(security.rdf,earnings.rdf,acc_pay.rdf)" dest="(output.sql)"
p_owner="Oracle9iAS Portal" p_servers="(repserver,acct_server)"
p_description="restricted report" p_privilege="(SCOTT,JABERS,ACCT)"
p_availability="production" p_types="(Cache,printer)"
p_formats="(HTMLCSS,PDF)" p_printers="(sales_printer,acct_printer)"
p_pformTemplate="public.finance_template"
p_trigger="Is begin IF UPPER(DESTYPE) = ''PRINTER'' AND
EMPNAME = ''SMITH'' THEN RETURN(TRUE); ELSE RETURN(FALSE); END IF; end;"
```

The above command line would generate a SQL script file named output.sql that contains the following:

```
SET SERVEROUTPUT ON

VAR STATUS NUMBER;

EXEC :STATUS := RWWWVREG.REGISTER_REPORT (P_NAME=>'Security',
P_OWNER=>'Oracle9iAS Portal', P_SERVERS=>'repserver,acct_server',
P_FILENAME=>'security.rdf', P_DESCRIPTION=>'restricted report',
P_PRIVILEGE=>'SCOTT,JABERS,ACCT', P_AVAILABILITY=>'production'
P_TYPES=>'Cache,printer', P_FORMATS=>'HTMLCSS,PDF)',
P_PRINTERS=>'sales_printer,acct_printer
P_PFORMTEMPLATE=>'public.finance_template' P_PARAMETERS=>'(P_LASTNAME)
(P_SSN)', P_TRIGGER=>'Is begin IF UPPER(DESTYPE) = ''PRINTER'' AND
EMPNAME = ''SMITH''THEN RETURN(TRUE); ELSE RETURN(FALSE); END IF; end;');

EXEC :STATUS := RWWWVREG.REGISTER_REPORT (P_NAME=>'Earnings',
P_OWNER=>'Oracle9iAS Portal', P_SERVERS=>'repserver,acct_server',
P_FILENAME=>'earnings.rdf', P_DESCRIPTION=>'restricted report',
P_PRIVILEGE=>'SCOTT,JABERS,ACCT', P_AVAILABILITY=>'production'
```

```
P_TYPES=>'Cache,printer)', P_FORMATS=>'HTMLCSS,PDF)',
P_PRINTERS=>'sales_printer,acct_printer',
P_PFORMTEMPLATE=>'public.finance_template',
P_TRIGGER='Is begin IF UPPER(DESTYPE) = ''PRINTER'' AND EMPNAME = ''JABERS''
THEN RETURN(TRUE); ELSE RETURN(FALSE); END IF; end;');

EXEC :STATUS := RWWWVREG.REGISTER_REPORT (P_NAME=>'Acc_pay',
P_OWNER=>'Oracle9iAS Portal', P_SERVERS=>'repserver,acct_server',
P_FILENAME=>'acc_pay.rdf', P_DESCRIPTION=>'restricted report',
P_PRIVILEGE=>'SCOTT,JABERS,ACCT', P_AVAILABILITY=>'production'
P_TYPES=>'Cache,printer', P_FORMATS=>'HTMLCSS,PDF',
p_printers=>'sales_printer,acct_printer',
P_PFORMTEMPLATE=>'public.finance_template'
P_TRIGGER=>'Is begin IF UPPER(DESTYPE) = ''PRINTER'' AND
EMPNAME = ''JABERS''THEN RETURN(TRUE); ELSE RETURN(FALSE); END IF; end;');
```

For more information about the contents of this SQL script file, refer to PL/SQL Batch Registering Function.

2. Check the reports.log file, which is typically written to the current working directory, for errors that may have occurred during the conversion process. If the reports.log file was not generated, then no errors were encountered by RWCONVERTER.

3. You can now optionally edit the system and user parameter values as desired. For example, the first RWWWVREG function in the sample script above generated an additional parameter called P_PARAMETERS. This occurred because the security.rdf file contains two user-defined parameters, P_LASTNAME and P_SSN:

   ```
   P_PARAMETERS=>'(P_LASTNAME)(P_SSN)',
   ```

   In this case, you can optionally define the default, low, and high values, or a list of values for each user parameter if you want to restrict the values the user may enter at runtime. Similarly, if you want to restrict system parameters, such as COPIES, to limit the number of copies a user can make, you do so by using the P_PARAMETERS parameter. The edited P_PARAMETERS keyword might look like the following:

   ```
   P_PARAMETERS=>'(P_LASTNAME, LOV=LASTNAME_LOV)(P_SSN)
   (COPIES, DEFAULT=1,LOW=1,HIGH=2)'
   ```

   This revised code segment imposes the following restrictions on the report:

- The P_LASTNAME user parameter is limited to the values listed in the LASTNAME_LOV list of values.
- A user-supplied value for P_SSN is required to validate the P_LASTNAME value.
- The default value of the COPIES system parameter is one and the number of printed copies must be in a range from 1 to 2.

4. Save and close the output.sql file.

C.1.2 Run the Script in SQL*Plus

To actually register your reports in Oracle9*i*AS Portal, you must run the script generated for you by RWCONVERTER:

1. Start SQL*Plus and log in to the Oracle9*i*AS Portal schema that you want to own the packaged procedures.
2. From the SQL*Plus command prompt, execute the script you created with RWCONVERTER:

   ```
   @ output.sql
   ```

   The script will execute and create packages in Oracle9*i*AS Portal for each report listed in the script with the specified parameters.
3. Log in to Oracle9*i*AS Portal as a user with RW_ADMINISTRATOR privileges.
4. Click the **Corporate Documents** tab.
5. Click **Builder**.
6. Click the **Administer** tab.
7. In the Oracle Reports Security portlet, click **Oracle Reports Security Settings**.
8. In the Reports Definition File Access portlet, enter the P_NAME of one of the reports you batch registered in your SQL script.
9. Click **Edit**. The Manage Component page is displayed.
10. Click **Edit** at the bottom of the page to edit the parameters of the report.
11. Review and edit the parameters as desired.
12. Click **OK**.
13. Click **Close**.

14. Repeat steps 8 through 13 for each report that you batch registered with your script.

C.2 Batch Removing Report Packages

To remove many reports from Oracle9*i*AS Portal at once, do the following:

1. In a text editor, create a SQL script file (e.g., rmv_rdfs.sql) that contains one RWWWVREG.DEREGISTER_REPORT function call for each report definition file package that you want to remove. For example:

```
VAR STATUS NUMBER;
EXEC :STATUS := RWWWVREG.DEREGISTER_REPORT (P_NAME=>'Security');
EXEC :STATUS := RWWWVREG.DEREGISTER_REPORT (P_NAME=>'Earnings');
EXEC :STATUS := RWWWVREG.DEREGISTER_REPORT (P_NAME=>'Acc_pay');
```

> **Note:** P_NAME is the name of the report definition file package you want to remove from Oracle9*i*AS Portal.

2. Start SQL*Plus and log in to the Oracle9*i*AS Portal schema that owns the reports' packaged procedures.
3. From the SQL*Plus command prompt, execute the script you created in the first step:

```
@ rmv_rdfs.sql
```

The script will execute and remove the packages from Oracle9*i*AS Portal for each report listed in the script.

> **Note:** This procedure will not remove the report definition files from the file system. It only unregisters the reports making them unavailable from Oracle9*i*AS Portal. If you want to remove the files, you must delete them from the file system.

C.3 PL/SQL Batch Registering Function

The SQL script that RWCONVERTER generates for you to batch register reports in Oracle9*i*AS consists mainly of calls to the rwwwvreg.register_report function. The syntax of rwwwvreg.register_report is as follows:

```
Function Rwwwvreg.register_report(
```

```
p_owner varchar2,
p_name varchar2,
p_servers varchar2,
p_filename varchar2,
p_description varchar2,
p_privileges varchar2,
p_availability varchar2,
p_types varchar2,
p_formats varchar2,
p_printers varchar2,
p_pdformTemplate varchar2,
p_parameters varchar2,
p_trigger varchar2)
return number;
  --  =0 : succeeded;
  --  !=0 : failed;
```

The table below describes each of the parameters taken by rwwwvreg.register_report.

Table C–1 rwwwvreg.register_report parameters

Parameter	Description
P_OWNER	Is the owner of the schema. The default is the current Oracle9*i*AS Portal schema that you are logged in to when you start the SQL*PLUS script. For example: `P_OWNER=>'Oracle9iAS Portal'`
P_NAME	Is the name used to identify the report in Oracle9*i*AS Portal. P_NAME corresponds to the Name field in the Create Report Definition File Access wizard. For example: `P_NAME=>'Earnings'`

Table C–1 rwwwvreg.register_report parameters

Parameter	Description
P_SERVERS	Is the names of the Reports Servers on which the report definition files defined in the P_FILENAME parameter have access privileges. The list of Reports Servers is comma delimited. P_FILENAME corresponds to the Reports Servers field in the Create Report Definition File Access wizard and the Edit Report Definition File page. For example: `P_SERVERS=>'repserver,acct'` **Note:** The Reports Servers you list for P_SERVERS must already be registered in Oracle9*i*AS Portal. For more information, refer to Chapter 5, "Controlling User Access".
P_FILENAME	Is the name of the report definition file that is being registered. P_FILENAME corresponds to the Oracle Reports File Name in the Create Report Definition File Access wizard and the Edit Report Definition File page. For example: `P_FILENAME=>'earnings.rdf'`
P_DESCRIPTION	Is a description of the report. P_DESCRIPTION corresponds to the Description field in the Create Report Definition File Access wizard and the Edit Report Definition File page. For example: `P_DESCRIPTION=>'restricted report'`
P_PRIVILEGE	Is the users or roles given privileges to run the report definition file defined in P_FILENAME. This list is comma delimited. P_PRIVILEGE corresponds to the Grantee list on the Access tab of the Manage Component page for the report. Note that you must uncheck Inherit Privileges from Portal DB Provider in order to see the Grantee list. For example: `P_PRIVILEGE=>'SCOTT,JABERS,PORTAL90'`

Table C–1 rwwwvreg.register_report parameters

Parameter	Description
P_AVAILABILITY	Is the name of the availability calendar that determines when the report definition file defined in the P_FILENAME parameter will be available for processing. P_AVAILABILITY corresponds to the Availability Calendar Name field in the Create Report Definition File Access wizard and the Edit Report Definition File page. For example: `P_AVAILABILITY=>'production'` **Note:** The availability calendar must already exist in Oracle9*i*AS Portal. For more information about creating an availability calendar, see Chapter 5, "Controlling User Access".
P_TYPES	Is the destination types to which the report definition file defined in the P_FILENAME parameter can be sent (e.g., cache, printer). This list is comma delimited. P_TYPES corresponds to the Types multiple select box in the Create Report Definition File Access wizard and the Edit Report Definition File page. For example: P_TYPES=>'CACHE,printer'
P_FORMATS	The destination formats to which the report definition file defined in the P_FILENAME parameter can be sent (e.g., HTML, PDF). This list is comma delimited. P_FORMATS corresponds to the Formats multiple select box in the Create Report Definition File Access wizard and the Edit Report Definition File page. For example: P_FORMATS=>'HTMLCSS,PDF'

Table C–1 rwwwvreg.register_report parameters

Parameter	Description
P_PRINTERS	The printers to which the report definition file defined in the P_FILENAME parameter can print. This list is comma delimited. P_PRINTERS corresponds to the Printers multiple select box in the Create Report Definition File Access wizard and the Edit Report Definition File page. For example: P_PRINTERS=>'sales_printer,acct_printer' **Note:** The printers you list for P_PRINTERS must already be registered in Oracle9*i*AS Portal. For more information, refer to Chapter 5, "Controlling User Access".
P_PFORMTEMPLATE	Is the parameter form template that determines the page style of the Runtime Parameter Form. P_PFORMTEMPLATE corresponds to the Parameter Form Template field in the Create Report Definition File Access wizard and the Edit Report Definition File page. For example: `P_PFORMTEMPLATE=>'public.finance_template'`

Table C–1 rwwwvreg.register_report parameters

Parameter	Description
P_PARAMETERS	Is the user and system parameters' default, high, and low values, or list of values name. **Note:** The P_PARAMETERS parameter does not have a corresponding RWCONVERTER argument. Hence, if you want to batch import user parameter values, ranges, or lists of values, you must manually edit the SQL script generated by RWCONVERTER. P_PARAMETERS corresponds to the (parameter) Name, LOV, Low Value, and High Value fields in the Create Report Definition File Access wizard and the Edit Report Definition File page. The default corresponds to the value set in the Runtime Parameter Form for the specified parameter. For example: `P_PARAMETERS=>'(P_LASTNAME, LOV=LASTNAME_LOV) (P_SSN)(COPIES, DEFAULT=1,LOW=1,HIGH=2)'` where: P_LASTNAME, P_SSN, and COPIES are parameter names. LOV is the name of the list of values. DEFAULT is the default value. LOW is the low value in a range of values. HIGH is the high value in a range of values.

P_TRIGGER	Is the validation trigger written in PL/SQL that returns a boolean statement (e.g., true (succeeded) or false (failed)).
	
	P_TRIGGER corresponds to the text box in the Create Report Definition File Access wizard and the Edit Report Definition File page.
	
	For example:
	
	`P_TRIGGER=>'Is begin`
	`IF UPPER(DESTYPE) = ''PRINTER'' AND`
	`  EMPNAME = ''SMITH''`
	`THEN`
	`  RETURN(TRUE);`
	`ELSE`
	`  RETURN(FALSE);`
	`END IF;`
	`end;'`

Index

A

access controls, 5-6 to 5-21
  availability calendar, combined, 5-10
  availability calendar, simple, 5-7
  printer, 5-12
  report, 5-16
  server, 5-14
accessible command keyword, A-7
Advanced Queuing, 11-2, 11-9, 11-10, 11-11, 11-12
  dbms_AQadm package, 11-10
  dbms_aq.dequeue, 11-13
  DEQUEUE, 11-9
  ENQUEUE, 11-9
  MESSAGES, 11-9
ALTER_SESSION, 12-15
API
  cache, 3-7
  clients, 3-23
  debugging events, 11-7
  destinations, 3-13
  engine, 3-9
  events, 3-15, 11-1 to 11-13
  notification, 3-16
  pluggable destinations, 4-2, 9-1, 9-33
  repository, 3-18
  security, 3-11
  Web Object Cache, 14-9
architecture
  destination, 4-3
  NLS, 12-2
  Reports Services, 1-1
  single sign-on, 7-2
arraysize command keyword, A-8
attach distribution element, 9-11
  format attribute, 9-12
  instance attribute, 9-12
  name attribute, 9-12
  srcType attribute, 9-12
attributes, using variables with, 9-2
authentication cookies, 3-35
authid command keyword, 2-5, A-8
authid parameter, events, 11-4
autocommit command keyword, A-9
autostart command keyword, 2-2, A-9
availability calendar, 5-7 to 5-12
  combined, 5-10
  simple, 5-7

B

batch command keyword, 2-2, 2-3, 2-5, A-10
batch modifications, XML, 10-21
batch registering reports in Oracle 9iAS Portal, C-1
bcc attribute, mail, 9-9
bcc command keyword, A-10
bidirectional support, 12-1 to 12-17
blankpages command keyword, A-11
body distribution element, 9-10
  format attribute, 9-11
  instance attribute, 9-11
  srcType attribute, 9-11
buffers command keyword, A-12
bursting, 9-6

C

cache, 3-8

FIFO, 14-7
ojsp:cache tag, 14-9
ojsp:cacheInclude tag, 14-10
ojsp:cacheXMLObj tag, 14-10
ojsp:invalidateCache tag, 14-10
ojsp:useCacheObj tag, 14-10
setting up in JSP, 14-9
strategies, 14-6 to 14-10
Web Object Cache, 14-6, 14-9
cache configuration element, 3-7, 8-12, 14-7
cacheSize attribute, 14-7
class attribute, 3-8
cache destype, 4-5
cache key, 8-11
cacheDir, 3-8
cachelob command keyword, A-12
cacheSize attribute, 14-7
cacheSize property, 3-8
caching, 8-11
callBackTimeOut attribute, 3-10, 14-5
cancelling a job, 11-6
case sensitivity, 2-4
cc attribute, mail, 9-8
cc command keyword, A-13
cellwrapper command keyword, A-14
CGI, 1-2, 1-5
backward compatibility, 3-6
URL syntax, 8-4
cgicmd.dat, 3-30, 8-13 to 8-15
adding entries, 8-14
using, 8-15
character set
unicode, 12-13 to 12-15
UTF8, 12-14
character sets, 12-1, 12-7
design considerations, 12-7
font aliasing, 12-7
class attribute
cache, 3-8
destination, 3-14
engine, 3-9
jobStatusRepository, 3-19
notification, 3-16
security, 3-12, 5-4
classPath attribute, engine, 3-9
cluster, 3-34
cluster configuration element, 3-23
clustering servers, 3-23
clusters, 6-1 to 6-6, 14-5 to 14-6
duplicate job detection, 8-12
linking to via OEM, 13-13
overview, 6-1
setting up, 6-2
submitting requests to, 6-6
viewing via OEM, 13-13
cmdfile command keyword, A-15
cmdkey command keyword, A-16
cmdkey parameter, events, 11-8
command keywords
accessible, A-7
arraysize, A-8
authid, 2-5, A-8
autocommit, A-9
autostart, 2-2, A-9
batch, 2-2, 2-3, 2-5, A-10
bcc, A-10
blankpages, A-11
buffers, A-12
cachelob, A-12
cc, A-13
cellwrapper, A-14
cmdfile, A-15
cmdkey, A-16
contentarea, A-16
copies, A-17
customize, 10-1, 10-3, 10-16, 10-17, 10-21, 10-24, A-18
dateformatmask, A-19
delauth, A-19
delimited_hdr, A-20
delimiter, A-20
desformat, A-21
desname, A-22
dest, A-23
destination, A-24
destype, A-25
distribute, A-26
dtype, A-27
dunit, A-28
expiration, 1-7, 14-7, 14-8, A-29

expiredays, A-30
express_server, A-30
formsize, A-33
from, A-33
getjobid, A-34
getserverinfo, A-35
help, A-35
ignoremargin, A-36
install, A-36, A-78
itemtitle, A-37
jobname, A-37
jobtype, A-38
killjobid, A-38
longchunk, A-39
mode, A-39
module, 10-16, 10-23, A-40
nonblocksql, A-41
notifyfailure, A-41
notifysuccess, A-42
numberformatmask, A-42
onfailure, A-43
onsuccess, A-44
orientation, A-44
ourputpage, A-46
outputfolder, A-45
overwrite, A-47
p_availability, A-47
p_description, A-48
p_formats, A-48
p_name, A-49
p_owner, A-49
p_pformtemplate, A-50
p_printers, A-50
p_privilege, A-51
p_servers, A-52
p_trigger, A-52
p_types, A-53
pagegroup, A-53
pagesize, A-54
pagestream, A-55
paramform, A-56
parsequery, A-56
pdfcomp, A-57
pdfembed, A-57
printjob, A-58
readonly, A-59
replaceitem, A-59
replyto, A-60
report, 10-16, 10-23, A-40
role, A-61
rundebug, A-61
save_rdf, A-62
schedule, 8-10, A-62
server, 2-3, 2-5, A-63
showenv, A-64
showjobs, 1-2, 11-5, 14-13, A-65
showmap, A-65
showmyjobs, A-66
shutdown, 2-5, A-66
sitename, A-67
source, A-68
ssoconn, A-69
statusfolder, A-71
statusformat, A-70
statuspage, A-72
stype, A-73
subject, A-73
tolerance, 1-7, 8-12, 8-13, 14-7, 14-8, A-74
tracefile, A-75
tracemode, A-76
traceopts, A-76
urlparameter, A-78
used with rwbuilder, A-3
used with rwcgi, A-6
used with rwclient, A-2
used with rwconverter, A-4
used with rwrun, A-2
used with rwserver, A-6
used with rwservlet, A-5
userid, A-79
webserver_debug, A-80
webserver_docroot, A-80
webserver_port, A-81
command lines, specifying, 8-5
commands
overview, A-1
rwbuilder, 10-23, A-3
rwcgi, A-5
rwclient, 10-16, 10-17, 10-21, A-2
rwconverter, 10-16, 10-21, A-3

rwrun, 10-17, 10-21, 10-23, A-2
rwserver, A-6
rwservlet, A-4
syntax, A-6
compatible configuration element, 3-3, 3-6
version attribute, 3-6
confidential attribute, 4-6, 5-4
configuration, 3-1 to 3-36
considerations, 1-8
OEM, 3-39
proxy information, 3-38
Reports Services security, 5-4
rwservlet.properties, 14-1
URL engine, 3-36
via OEM, 13-12
configuration elements
cache, 3-7, 8-12, 14-7
cluster, 3-23
compatible, 3-3, 3-6
connection, 3-21
destination, 3-13, 4-5
engine, 3-8
identifier, 3-26
job, 3-14
jobStatusRepository, 3-18, 14-17
log, 3-17
notification, 3-15
orbClient, 3-22
persistFile, 3-3, 3-25
pluginParam, 3-27, 9-9
queue, 3-24
security, 3-11, 5-3
server, 3-5
trace, 3-19
configuration files
Reports Server, 3-2
rwserverconf.dtd, 3-4
rwserver.template, 3-3
rwservlet.properties, 3-3
configuration, Reports Server, 1-8
connection configuration element, 3-21
idleTimeOut attribute, 3-22
maxConnect attribute, 3-22
connection strings, 7-5
contentarea command keyword, A-16
cookie, 7-2
cookieexpire, rwservlet.properties, 3-35
copies attribute
printer, 9-19
copies command keyword, A-17
current jobs queue, 13-4
Custom Tag Handler, 1-5
customize command keyword, 10-1, 10-3, 10-21, 10-24, A-18
customize keyword, 10-17

D

DAS, see Delegation Administration Service
data models, creating, 10-9 to 10-16
data sources
creating via XML, 10-9
group hierarchies via XML, 10-11
linking via XML, 10-10
Data Type Dictionary
distribution.dtd, 9-33
data type dictionary
distribution.dtd, 9-2
reports.dtd, 10-2
rwserverconf.dtd, 3-2, 3-4
data types
DATE, 12-3
NUMBER, 12-3
database authentication cookies, 3-35
database triggers, 11-8
dateformatmask command keyword, A-19
day names, language for, 12-8
db_sys_diffauth, rwservlet.properties, 3-31
db_sys_sameauth, rwservlet.properties, 3-31
dbauth, rwservlet.properties, 3-31
dbms_AQadm package, 11-10
dbms_aq.dequeue, 11-13
debugging events, 11-7
delauth command keyword, A-19
Delegated Administration Service, 5-6, 7-2, 7-4
delimited_hdr command keyword, A-20
delimiter command keyword, A-20
DEQUEUE, 11-9
dequeuing, creating procedure, 11-12
desformat command keyword, A-21

desname command keyword, A-22
dest command keyword, A-23
destination
  classes, 4-5
  destypes, 4-5
  valid values, 4-5
destination architecture, 4-3
destination command keyword, A-24
destination configuration element, 3-13, 4-5
  class attribute, 3-14
  destype attribute, 3-14
destination types, 3-13, 4-4
destinations distribution element, 9-4
destype attribute, 3-14, 4-5
  cache, 4-5
  file, 4-5
  mail, 4-5
  oraclePortal, 4-5
  printer, 4-5
destype command keyword, A-25
destype distribution element, 9-19
  id attribute, 9-21
  instance attribute, 9-21
  name attribute, 9-21
DEVELOPER_NLS_LANG, 12-9, 12-13, B-1
diagbodytags, rwservlet.properties, 3-35
diagheadtags, rwservlet.properties, 3-35
diagnostic, rwservlet.properties, 3-31
direction, of language, 12-8
dist parameter, events, 11-9
distribute command keyword, A-26
distribution, 9-1 to 9-33
  bursting, 9-6
distribution elements
  attach, 9-11
  body, 9-10
  destinations, 9-4
  destype, 9-19
  file, 9-16
  foreach, 9-5
  include, 9-13
  mail, 9-7
  printer, 9-18
  property, 9-22
distribution examples, 9-22 to 9-33
  file, 9-28
  foreach, 9-22
  mail, 9-25
  printer, 9-29
distribution overview, 9-1 to 9-2
distribution, using XML file, 9-32
distribution.dtd, 9-2, 9-33
distribution.xsl, 9-33
documentation
  related documents, xxi
  structure of this manual, xx
DTD, see data type dictionary
dtype command keyword, A-27
dunit command keyword, A-28
duplicate job detection, 8-12

E

elements, see distribution, customization, or configuration
encrypted attribute, 3-12, 4-6, 5-4
encryption, 4-6, 5-4
encryptionkey, rwservlet.properties, 3-35
engine configuration element, 3-8, 14-2
  callBackTimeOut attribute, 3-10, 14-5
  class attribute, 3-9
  classPath attribute, 3-9
  engLife attribute, 3-10, 14-4
  id attribute, 3-9
  initEngine attribute, 3-9, 14-3
  maxEngine attribute, 3-10, 14-3
  maxIdle attribute, 3-10, 14-4
  minEngine attribute, 3-10, 14-4
engine, tuning, 14-2
engineId attribute, 3-15
engLife attribute, 3-10, 14-4
ENQUEUE, 11-9
enqueuing, creating procedure, 11-11
environment variables
  DEVELOPER_NLS_LANG, 12-9, B-1
  NLS, 12-15
  NLS_CALENDAR, B-1
  NLS_CREDIT, B-2
  NLS_CURRENCY, B-2
  NLS_DATE_FORMAT, B-2

NLS_DATE_LANGUAGE, B-2
NLS_DEBIT, B-2
NLS_ISO_CURRENCY, B-2
NLS_LANG, 12-3, B-2
NLS_LIST_SEPARATOR, B-2
NLS_MONETARY_CHARACTERS, B-2
NLS_NUMERIC_CHARACTERS, B-2
NLS_SORT, B-2
PATH, 2-4
REPORTS_CGIDIAGBODYTAGS, B-3
REPORTS_CGIDIAGHEADTAGS, B-3
REPORTS_CGIHELP, B-3
REPORTS_CGIMAP, B-3
REPORTS_CGINODIAG, B-4
REPORTS_COOKIE_EXPIRE, B-2
REPORTS_DB_AUTH, B-2
REPORTS_ENCRYPTION_KEY, B-2
REPORTS_PATH, B-4
REPORTS_SERVER, B-4
REPORTS_SSLPORT, B-4
REPORTS_SYS_AUTH, B-4
REPORTS_TMP, B-4
RW, B-5
USER_NLS_LANG, 12-9, B-5
environment variables, editing, B-1
environment variables, NLS, 12-2 to 12-10
error messages, 3-31, 3-33
error messages, XML, 10-22
errortemplate, 3-33
event-driven publishing, 1-8, 11-1 to 11-13
events
authid parameter, 11-4
cancelling a job, 11-6
cmdkey parameter, 11-8
creating a message queue, 11-10
creating dequeuing procedure, 11-12
creating enqueuing procedure, 11-11
debugging, 11-7
dist parameter, 11-9
gateway parameter, 11-4
invoking a report, 11-8
MyIdent.AuthID, 11-5
MyIdent.GatewayURL, 11-5
MyIdent.JobID, 11-5
MyIdent.ServerName, 11-5
ParamList-Object, 11-2
ParamList-Type, 11-2
report parameter, 11-4, 11-8
server parameter, 11-4
SRW_PARAMETER, 11-2
SRW_PARAMLIST, 11-2, 11-5, 11-10
srw_test.sql, 11-7
SRW.ADD_PARAMETER, 11-3
srwAPIdrop.sql, 11-2
srwAPIgrant.sql, 11-2
srwAPIins.sql, 11-2
SRW.CANCEL_REPORT, 11-7
SRW.CLEAR_PARAMETER_LIST, 11-4
SRW.JOB_IDENT, 11-5
SRW-Package, 11-2
SRW.REMOVE_PARAMETER, 11-3
SRW.REPORT_STATUS, 11-5
SRW.START_DEBUGGING, 11-7
SRW.STATUS_RECORD, 11-5
SRW.STOP_DEBUGGING, 11-7
userid parameter, 11-4, 11-8
examples, distribution, 9-22 to 9-33
file, 9-28
foreach, 9-22
mail, 9-25
printer, 9-29
expiration command keyword, 1-7, 14-7, 14-8, A-29
expiredays command keyword, A-30
express_server command keyword, A-30

F

failed jobs queue, 13-9
fail-safe environment via clustering, 14-5
FIFO, 14-7
file destype, 4-5
file distribution element, 9-16
format attribute, 9-17
id attribute, 9-17
instance attribute, 9-17
name attribute, 9-17
fileName attribute, 3-26
finished jobs queue, 13-6
firewall
proxy information, 3-38

font aliasing, 12-7
font mapping, 12-8
font support, NLS, 12-14
fonts, true type big, 12-14
foreach distribution element, 9-5
format attribute
  attach, 9-12
  body, 9-11
  file, 9-17
formsize command keyword, A-33
from attribute, mail, 9-9
from command keyword, A-33

G

gateway parameter, events, 11-4
getjobid command keyword, A-34
getserverinfo command keyword, A-35

H

help command keyword, A-35
help, rwservlet.properties, 3-36
HTML
  in debugging output, 3-35
  in diagnostic output, 3-35
HTTP Secure Sockets Layer, 1-5
HTTP Server, 1-5, 7-3
HTTPS, 1-5
hyperlink job request, 8-9
hyperlinks, adding via XML, 10-7

I

id attribute
  destype, 9-21
  engine, 3-9
  file, 9-17
  mail, 9-8
  notification, 3-16
  orbClient, 3-23
  printer, 9-19
  security, 3-12, 5-3
identifier configuration element, 3-26
idleTimeOut attribute, 3-22
ignoremargin command keyword, A-36
image_url, rwservlet.properties, 3-34
include distribution element, 9-13
  src attribute, 9-15
initEngine attribute, 3-9, 14-3
in-process server, 1-2, 1-5, 3-33, 14-1
install command keyword, A-36, A-78
instance attribute
  attach, 9-12
  body, 9-11
  destype, 9-21
  file, 9-17
  printer, 9-19
itemtitle command keyword, A-37

J

Java 2 Enterprise Edition, 1-5
Java Servlet, 1-1
job cancelling, 11-6
job configuration element, 3-14
  engineId attribute, 3-15
  jobType attribute, 3-15
  securityId attribute, 3-15
job queues
  managing via OEM, 13-3
  viewing via OEM, 13-3
jobname command keyword, A-37
jobs queue
  current, 13-4
  failed, 13-9
  finished, 13-6
  scheduled, 13-5
jobs, running, 8-1 to 8-15
jobStatusRepository configuration element, 3-18, 14-17
  class attribute, 3-19
  repositoryconn attribute, 14-17
jobType attribute, 3-15
jobtype command keyword, A-38
JSP, 1-1, 1-5
  Afrikaans character set, 12-11
  Albanian character set, 12-11
  Arabic character set, 12-11
  Basque character set, 12-11

Bulgarian character set, 12-11
Byelorussian character set, 12-11
Catalan character set, 12-11
Croatian character set, 12-11
Czech character set, 12-11
Danish character set, 12-11
Dutch character set, 12-11
English character set, 12-11
Esperanto character set, 12-11
Estonian character set, 12-11
Faroese character set, 12-11
Finnish character set, 12-11
French character set, 12-11
Galician character set, 12-11
German character set, 12-11
Greek character set, 12-11
Hebrew character set, 12-11
Hungarian character set, 12-11
Icelandic character set, 12-11
images, 3-34
Inuit languages character set, 12-11
Irish character set, 12-11
Italian character set, 12-11
Japanese character set, 12-11
Korean character set, 12-12
Lapp character set, 12-12
Latvian character set, 12-12
Lithuanian character set, 12-12
Macedonian character set, 12-12
Maltese character set, 12-12
NLS, 12-10 to 12-12
Norwegian character set, 12-12
Polish character set, 12-12
Portuguese character set, 12-12
Romanian character set, 12-12
Russian character set, 12-12
Scottish character set, 12-12
Serbian character set, 12-12
setting up cache, 14-9
Slovak character set, 12-12
Slovenian character set, 12-12
Spanish character set, 12-12
specifying character set, 12-10 to 12-12
Swedish character set, 12-12
taglib, 14-9
Turkish character set, 12-12
Ukrainian character set, 12-12
URL syntax, 8-3
using key mapping with, 8-15
Web Object Cache, 14-6, 14-9

K

key map file, 8-13 to 8-15
adding entries, 8-14
benefits, 8-13
enabling, 8-13
mapping URL parameters, 8-14
reloading, 3-30
restricted run with Parameter Form, 8-15
specifying location, 3-30
using, 8-15
KeyMapFile, 3-30
keywords
accessible, A-7
arraysize, A-8
authid, A-8
autocommit, A-9
autostart, A-9
batch, A-10
bcc, A-10
blankpages, A-11
buffers, A-12
cachelob, A-12
cc, A-13
cellwrapper, A-14
cmdfile, A-15
cmdkey, A-16
contentarea, A-16
copies, A-17
customize, 10-1, 10-3, 10-16, 10-17, 10-21, 10-24, A-18
dateformatmask, A-19
delauth, A-19
delimited_hdr, A-20
delimiter, A-20
desformat, A-21
desname, A-22
dest, A-23
destination, A-24

destype, A-25
distribute, A-26
dtype, A-27
dunit, A-28
expiration, 14-7, 14-8, A-29
expiredays, A-30
express_server, A-30
formsize, A-33
from, A-33
getjobid, A-34
getserverinfo, A-35
help, A-35
ignoremargin, A-36
install, A-36, A-78
itemtitle, A-37
jobname, A-37
jobtype, A-38
killjobid, A-38
longchunk, A-39
mode, A-39
module, 10-16, 10-23, A-40
nonblocksql, A-41
notifyfailure, A-41
notifysuccess, A-42
numberformatmask, A-42
onfailure, A-43
onsuccess, A-44
orientation, A-44
outputfolder, A-45
outputpage, A-46
overwrite, A-47
p_availability, A-47
p_description, A-48
p_formats, A-48
p_formtemplate, A-50
p_name, A-49
p_owner, A-49
p_printers, A-50
p_privilege, A-51
p_servers, A-52
p_trigger, A-52
p_types, A-53
pagegroup, A-53
pagesize, A-54
pagestream, A-55
paramform, A-56
parsequery, A-56
pdfcomp, A-57
pdfembed, A-57
printjob, A-58
readonly, A-59
replaceitem, A-59
replyto, A-60
report, 10-16, 10-23, A-40
role, A-61
rundebug, A-61
save_rdf, A-62
schedule, 8-10, A-62
server, A-63
showenv, A-64
showjobs, 11-5, 14-13, A-65
showmap, A-65
showmyjobs, A-66
shutdown, A-66
sitename, A-67
source, A-68
ssoconn, A-69
statusfolder, A-71
statusformat, A-70
statuspage, A-72
stype, A-73
subject, A-73
tolerance, 8-12, 8-13, 14-7, 14-8, A-74
tracefile, A-75
tracemode, A-76
traceopts, A-76
urlparameter, A-78
used with rwbuilder, A-3
used with rwcgi, A-6
used with rwclient, A-2
used with rwconverter, A-4
used with rwrun, A-2
used with rwserver, A-6
used with rwservlet, A-5
userid, A-79
webserver_debug, A-80
webserver_docroot, A-80
webserver_port, A-81
killjobid command keyword, A-38

L

languages
  Middle Eastern, 12-12
  North African, 12-12
languages, see NLS_LANG
LDAP, 5-4, 7-5
listener, 1-5
log configuration element, 3-17
  option attribute, 3-18
longchunk command keyword, A-39

M

mail destype, 4-5
mail distribution element, 9-7
  bcc attribute, 9-9
  cc attribute, 9-8
  from attribute, 9-9
  id attribute, 9-8
  organization attribute, 9-9
  priority attribute, 9-9
  replyTo attribute, 9-9
  returnReciept attribute, 9-9
  subject attribute, 9-9
  to attribute, 9-8
managing Reports Services, 13-1 to 13-15
map
  URL parameters, key map file, 8-14
maxConnect attribute, 3-22
maxEngine attribute, 3-10, 14-3
maxIdle attribute, 3-10, 14-4
maxQueueSize attribute, 3-25
message 401, 7-4
message queue, creating, 11-10
MESSAGES, 11-9
messages, language for, 12-8
Middle Eastern languages, 12-12
minEngine attribute, 3-10, 14-4
mod_oc4j, 1-5
mod_osso, 7-3
modecommand keyword, A-39
module command keyword, 10-23, A-40
module keyword, 10-16
monitoring Reports Services, 13-1 to 13-15
month names, language for, 12-8
multibyte, 12-1, 12-7, 12-13
multilingual text display, 12-14
MyIdent.AuthID, 11-5
MyIdent.GatewayURL, 11-5
MyIdent.JobID, 11-5
MyIdent.ServerName, 11-5

N

name attribute, 3-27
  attach, 9-12
  destype, 9-21
  file, 9-17
  printer, 9-19
name / value pairs, destination, 4-6
National Language Support, see NLS
NLS, 12-1 to 12-17
  ALTER_SESSION, 12-15
  architecture, 12-2
  character sets, 12-7
  DEVELOPER_NLS_LANG, 12-9
  environment variables, 12-2 to 12-10
    NLS_LANG, 12-14
  font support, 12-14
  JSP, 12-10 to 12-12
  language-dependent data, 12-2
  language-independent functions, 12-2
  NLS_LANG, 12-3, 12-14
  translating applications, 12-16
  unicode, 12-13 to 12-15
  USER_NLS_LANG, 12-9
NLS_CALENDAR, B-1
NLS_CREDIT, B-2
NLS_CURRENCY, B-2
NLS_DATE_FORMAT, B-2
NLS_DATE_LANGUAGE, B-2
NLS_DEBIT, B-2
NLS_ISO_CURRENCY, B-2
NLS_LANG, 12-3, 12-14, 12-15, B-2
  American language, 12-4
  Arabic language, 12-4
  Bulgarian language, 12-4
  Canadian French language, 12-4
  Catalan language, 12-4

charset, 12-4
Chinese (Simplified) language, 12-5
Chinese (Traditional) language, 12-5
Croatian language, 12-4
Czech language, 12-4
Danish language, 12-4
defining, 12-6
Dutch language, 12-4
Egyptian language, 12-4
English (American) language, 12-4
English (United Kingdom) language, 12-4
Estonian language, 12-4
Finnish language, 12-4
French language, 12-5
German language, 12-5
Greek language, 12-5
Hebrew language, 12-5
Hungarian language, 12-5
Icelandic language, 12-5
Indonesian language, 12-5
Italian language, 12-5
Japanese language, 12-5
Korean language, 12-5
language, 12-3
language conventions, 12-8
Latvian language, 12-5
Lithuanian language, 12-5
Norwegian language, 12-5
Polish language, 12-5
Portuguese (Brazilian) language, 12-4
Portuguese language, 12-5
Romanian language, 12-5
Russian language, 12-5
Slovak language, 12-5
Spanish (Latin Americal) language, 12-5
Spanish (Mexican) language, 12-5
Spanish language, 12-5
Swedish language, 12-5
syntax, 12-3
territory, 12-4
territory conventions, 12-8
Thai language, 12-5
Turkish language, 12-6
Ukrainian language, 12-6
Vietnamese language, 12-6
NLS_LANG environment variable, 12-14, 12-15
NLS_LIST_SEPARATOR, B-2
NLS_MONETARY_CHARACTERS, B-2
NLS_NUMERIC_CHARACTERS, B-2
NLS_SORT, B-2
nonblocksql command keyword, A-41
North African languages, 12-12
notational conventions, xxii
notification configuration element, 3-15
  class attribute, 3-16
  id attribute, 3-16
notifyfailure command keyword, A-41
notifysuccess command keyword, A-42
numberformatmask command keyword, A-42

O

OC4J, 2-5, 7-3, 14-9
OEM, 13-1 to 13-15
  configuring for Reports Server, 3-39
  launching, 13-2
  managing job queues, 13-3
  navigating to Reports Services, 13-2
  performance monitoring, 13-11
  reconfiguring Reports Server, 13-12
  restarting Reports Servers, 13-2
  starting Reports Servers, 13-2
  stopping Reports Servers, 13-2
  viewing job queues, 13-3
OID, see Oracle Internet Directory
ojsp:cache tag, 14-9
ojsp:cacheInclude tag, 14-10
ojsp:cacheXMLObj tag, 14-10
ojsp:invalidateCache tag, 14-10
ojsp:useCacheObj tag, 14-10
onfailure command keyword, A-43
onsuccess command keyword, A-44
OPMN, 2-5
option attribute, 3-18
Oracle Advanced Queuing, 11-2, 11-9, 11-10, 11-11, 11-12
  dbms_AQadm package, 11-10
  dbms_aq.dequeue, 11-13
  DEQUEUE, 11-9
  ENQUEUE, 11-9

MESSAGES, 11-9
Oracle Containers for J2EE, 1-5
Oracle Containers for Java 2 Enterprise Edition, 7-3
Oracle Delegated Administration Service, 7-6
Oracle Enterprise Manager, see OEM
Oracle HTTP Server, 7-3
Oracle Internet Directory, 5-4, 7-1, 7-4, 7-5, 7-6
Oracle Login Server, 7-1, 7-4
Oracle Program Manager, 2-5
Oracle Single Sign-on, 5-2
Oracle Technology Network, 5-2
Oracle Trace, 14-11 to 14-13
ORACLE_HOME, 3-1
Oracle9iAS HTTP Server, 1-5, 2-1
Oracle9iAS Portal, 5-1 to 5-21
access controls, 5-6
availability calendar
combined, 5-10
simple, 5-7
batch registering reports, C-1
introduction, 5-2
printer access, 5-12
publishing a report portlet, 8-7
report access, 5-16
report requests, 8-6
runtime parameter form, 5-20
RW_ADMINISTRATOR, 5-5
RW_BASIC_USER, 5-5
RW_DEVELOPER, 5-5
RW_POWER_USER, 5-5
server access, 5-14
users and groups, 5-4
Oracle9iAS Reports Engine, clustering architecture,
load balancing, 6-2
Oracle9iAS Reports Services
intended audience, xix
security configuration, 5-4
single sign-on, 5-2
oraclePortal destype, 4-5
orbClient configuration element, 3-22
id attribute, 3-23
publicKeyFile Attribute, 3-23
organization attribute, mail, 9-9
orientation command keyword, A-44
OTN, see Oracle Technology Network
output
Postscript, 2-3
printer, 2-3
output processing, 4-1 to 4-4
output types, 4-1
outputfoldercommand keyword, A-45
outputpage command keyword, A-46
overwrite command keyword, A-47

P

p_availability command keyword, A-47
p_description command keyword, A-48
p_formats command keyword, A-48
p_name command keyword, A-49
p_owner command keyword, A-49
p_pformtemplate command keyword, A-50
p_printers command keyword, A-50
p_privilege command keyword, A-51
p_servers command keyword, A-52
p_trigger command keyword, A-52
p_types command keyword, A-53
pagegroup command keyword, A-53
pagesize command keyword, A-54
pagestream command keyword, A-55
paramater list (events)
manipulating, 11-2 to 11-4
parameter form, 5-20
key map file, 8-15
parameter list (events)
creating, 11-2 to 11-4
paramform command keyword, A-56
ParamList-Object, 11-2
ParamList-Type, 11-2
parsequery command keyword, A-56
PATH, 2-4
pdfcomp command keyword, A-57
pdfembed command keyword, A-57
performance monitoring, 13-11, 14-10 to 14-14
Oracle Trace, 14-11 to 14-13
persisiFile configuration element
fileName attribute, 3-26
persistance, 1-2, 3-25, 8-12
persistFile configuration element, 3-3, 3-25
PL / SQL

ALTER_SESSION, 12-15
SRW_PARAMETER, 11-2
SRW_PARAMLIST, 11-2, 11-5, 11-10
srw_test.sql, 11-7
SRW.ADD_DEFINITION, 10-18, 10-24
SRW.ADD_PARAMETER, 11-3
srwAPIdrop.sql, 11-2
srwAPIgrant.sql, 11-2
srwAPIins.sql, 11-2
SRW.APPLY_DEFINITION, 10-3, 10-16, 10-18
SRW.CANCEL_REPORT, 11-7
SRW.CLEAR_PARAMETER_LIST, 11-4
SRW.JOB_IDENT, 11-5
SRW-Package, 11-2
SRW.REMOVE_PARAMETER, 11-3
SRW.REPORT_STATUS, 11-5
SRW.START_DEBUGGING, 11-7
SRW.STATUS_RECORD, 11-5
SRW.STOP_DEBUGGING, 11-7
translating blocks, 12-16
PL/SQL, and advanced distribution, 9-3
pluggable
cache, 3-7
clients, 3-23
destinations, 3-13, 4-2, 9-1, 9-33
engine, 3-9
events, 3-15
notification, 3-16
repository, 3-18
security, 3-11
pluginParam
used with jobStatusRepository, 3-19
used with notification, 3-16
pluginParam configuration element, 3-27, 9-9
name attribute, 3-27
type attribute, 3-28
used with cache, 3-8
used with destination, 3-14
used with engine, 3-11
used with security, 3-13
Portal, 5-1 to 5-21
access controls, 5-6
availability calendar, combined, 5-10
availability calendar, simple, 5-7
introduction, 5-2
printer access, 5-12
report access, 5-16
report requests, 8-6
runtime parameter form, 5-20
RW_ADMINISTRATOR, 5-5
RW_BASIC_USER, 5-5
RW_DEVELOPER, 5-5
RW_POWER_USER, 5-5
server access, 5-14
users and groups, 5-4
portlet
adding to a page, 8-8
batch registering reports, C-1
creating provider for reports, 8-7
creating report definition file access, 8-7
publishing a report, 8-7
Postscript, output to, 2-3
printer access controls, 5-12
printer destype, 4-5
printer distribution element, 9-18
copies attribute, 9-19
id attribute, 9-19
instance attribute, 9-19
name attribute, 9-19
printer, output to, 2-3
printjob command keyword, A-58
priority attribute, mail, 9-9
private key, 6-5
private key file, generating, 3-23
program units, adding via XML, 10-7
property distribution element, 9-22
provider
creating for reports, 8-7
proxy information
configuring, 3-38
public key, 6-5
public key file, generating, 3-23
publicKeyFile attribute, 3-23

Q

queue configuration element
maxQueueSize attribute, 3-25
queue element, 3-24
queue manager, 1-2, 8-10

queue viewer, 1-2
queues
  current jobs, 13-4
  failed jobs, 13-9
  finished jobs, 13-6
  managing via OEM, 13-3
  RW_SERVER_QUEUE table, 14-14
  scheduled jobs, 13-5
  viewing via OEM, 13-3

R

reading order, 12-1, 12-12
readonly command keyword, A-59
registry, editing, B-1
registry, Windows, 12-6
reload_keymap, rwservlet.properties, 3-30
replaceitem command keyword, A-59
replyTo attribute, mail, 9-9
replyto command keyword, A-60
report access controls, 5-16
report command keyword, 10-23, A-40
report definitions, XML, 10-1
report keyword, 10-16
report parameter, events, 11-4, 11-8
reports
  applying custom XML, 10-16 to 10-22
  batch registering in Oracle9iAS Portal, C-1
  batch removing from Oracle9iAS Portal, C-5
  bursting, 9-6
  caching, 8-11
  command line requests, 8-5
  current jobs queue, 13-4
  debugging custom XML, 10-22 to 10-24
  failed jobs queue, 13-9
  finished jobs queue, 13-6
  hyperlinking to, 8-9
  invoking via events, 11-8
  processing, 1-6
  request methods, 8-5
  request via packaged procedure, 8-6
  requests via Portal, 8-6
  running automatically, 8-10
  scheduled jobs queue, 13-5
  scheduling, 8-10
  URL requests, 8-6
  URL syntax, 8-1, 8-9
  XML customization, 10-3 to 10-9
  XML data models, 10-9 to 10-16
Reports Cache, 1-5
Reports CGI, 1-5
Reports Engine, 1-6, 1-8
Reports JSP, 1-5
Reports Queue Manager, 1-2, 8-10
Reports Queue Viewer, 1-2
Reports Server, 1-5, 7-4
  access controls, 5-14
  adding to OEM, 13-14
  clusters, 6-1 to 6-6, 14-5 to 14-6
  configuration file, 1-8, 3-2
  destination processing, 4-3
  in-process server, 14-1
  monitoring performance, 13-11
  performance monitoring, 14-10 to 14-14
  persistance, 3-25, 8-12
  reconfiguring via OEM, 13-12
  registering destination types, 4-4
  renaming, 6-3
  restarting, 6-6
  restarting via OEM, 13-2
  starting as service, 2-2
  starting as servlet, 2-3
  starting from command line, 2-3
  starting via OEM, 13-2
  status record, 11-6
  stopping via OEM, 13-2
  tuning, 14-1 to 14-18
Reports Services
  about, 1-1
  architecture, 1-1
  cache API, 3-7
  clients API, 3-23
  components, 1-4
  destinations API, 3-13, 4-2, 9-1
  engine API, 3-9
  events API, 3-15
  managing, 13-1 to 13-15
  monitoring, 13-1 to 13-15
  notification API, 3-16
  persistance, 1-2

Reports Engine, 14-2
repository API, 3-18
security API, 3-11
single sign-on, 5-2
starting and stopping, 2-1
Reports Servlet, 1-5, 3-3, 7-3
URL syntax, 8-2
reports, running, 8-1 to 8-15
REPORTS_CGIDIAGBODYTAGS, B-3
REPORTS_CGIDIAGHEADTAGS, B-3
REPORTS_CGIHELP, B-3
REPORTS_CGIMAP, B-3
REPORTS_CGINODIAG, B-4
REPORTS_COOKIE_EXPIRE, B-2
REPORTS_DB_AUTH, B-2
REPORTS_ENCRYPTION_KEY, B-2
REPORTS_PATH, B-4
REPORTS_SERVER, B-4
REPORTS_SSLPORT, B-4
REPORTS_SYS_AUTH, B-4
REPORTS_TMP, B-4
reports.dtd, 10-2
repositoryconn attribute, 14-17
returnReceipt attribute, mail, 9-9
role command keyword, A-61
rundebug command keyword, A-61
runing a report automatically
from Oracle9iAS Portal, 8-9
running a report, 8-1
running a report automatically, 8-10
runtime parameter form, 5-20
runtime URL, 8-1 to 8-15
runtime URL syntax, 8-1
RW environment variable, B-5
RW_ADMINISTRATOR, 5-5
RW_BASIC_USER, 5-5
RW_DEVELOPER, 5-5
RW_POWER_USER, 5-5
RW_SERVER_QUEUE table, 14-14
rw_server.sql, 14-17
rwbuilder command, 10-23, A-3
keywords used with, A-3
rwbuilder.conf, 3-2
rwcgi command, A-5
keywords used with, A-6
rwclient, 8-5
rwclient command, 10-16, 10-17, 10-21, A-2
keywords used with, A-2
rwconverter
generating a SQL script for batch registration, C-1
rwconverter command, 10-16, 10-21, A-3
keywords used with, A-4
rwproxy, 3-6
rwrun command, 10-17, 10-21, 10-23, A-2
keywords used with, A-2
rwrun.jar, 6-5
rwserver, 2-5
install, 2-2
server, 2-3
rwserver command, A-6
keywords used with, A-6
rwserverconf.dtd, 3-2, 3-4, 4-4, 4-6
cache element, 3-7
cluster element, 3-23
compatible element, 3-6
connection element, 3-21
destination element, 3-13
engine element, 3-8
identifier element, 3-26
job element, 3-14
jobStatusRepository element, 3-18
log element, 3-17
notification element, 3-15
orbClient element, 3-22
persistFile element, 3-25
pluginParam element, 3-27
queue element, 3-24
security element, 3-11
server element, 3-5
trace element, 3-19
rwserver.template, 3-3
rwservlet, 1-5, 2-3, 14-1
rwservlet command, A-4
keywords used with, A-5
rwservlet.properties, 3-3, 14-1
cookieexpire, 3-35
db_sys_diffauth, 3-31
db_sys_sameauth, 3-31
dbauth, 3-31

diagbodytags, 3-35
diagheadtags, 3-35
diagnostic, 3-31
encryptionkey, 3-35
errortemplate, 3-33
help, 3-36
image_url, 3-34
KeyMapFile, 3-30
reload_keymap, 3-30
server, 3-34
server_in_process, 3-33
single sign-on, 3-36
sslport, 3-35
sysauth, 3-31
tracefile, 3-32
tracemode, 3-32
traceopts, 3-32

S

save_rdf command keyword, A-62
schedule command keyword, 8-10, A-62
scheduled jobs queue, 13-5
scripts
rw_server.sql, 14-17
srw_test.sql, 11-7
srwAPIdrop.sql, 11-2
srwAPIgrant.sql, 11-2
srwAPIins.sql, 11-2
Secure Sockets Layer, 1-5
security configuration, 5-4
security configuration element, 3-11, 5-3
class attribute, 3-12
id attribute, 3-12
security element
class attribute, 5-4
id attribute, 5-3
securityId attribute, 3-15
serve parameter, events, 11-4
server
in_process, 3-33
in-process, 1-2, 1-5
rwserver command, A-6
server access controls, 5-14
server cluster, 3-34
server clusters, 3-23
server command keyword, 2-3, 2-5, A-63
server configuration element, 3-5
server, in-process, 14-1
server, rwservlet.properties, 3-34
server, showjobs parameter, 14-13
server_in_process, 3-33
server_name.conf, 3-2
SERVEROUT, 11-7
servlet, 1-1, 1-5, 2-3, 3-3, 7-3
adding custom help, 3-36
rwservlet, 14-1, A-4
rwservlet.properties, 14-1
URL syntax, 8-2
session cookie, 7-2
showenv command keyword, A-64
showjobs command keyword, 1-2, 11-5, 14-13, A-65
server parameter, 14-13
statusformat parameter, 14-14
showmap command keyword, A-65
showmyjobs command keyword, A-66
shutdown command keyword, 2-5, A-66
single sign-on, 5-2, 7-1 to 7-7
feature, 5-2
process, 7-4
rwservlet.properties, 3-36
single-byte, 12-13, 12-14
sitename command keyword, A-67
SMTP, 3-16, 9-8, 9-9
sorting sequence, of language, 12-8
source command keyword, A-68
specifying a report request
for the URL engine, 8-10
from a Web browser, 8-9
publishing a report portlet, 8-7
scheduling to run automatically, 8-10
SQL*PLUS, 11-7
src attribute, include, 9-15
srcType attribute
attach, 9-12
body, 9-11
SRW_PARAMETER, 11-2
SRW_PARAMLIST, 11-2, 11-5, 11-10
srw_test.sql, 11-7

SRW.ADD_DEFINITION, 10-18, 10-24
SRW.ADD_PARAMETER, 11-3
srwAPIdrop.sql, 11-2
srwAPIgrant.sql, 11-2
srwAPIins.sql, 11-2
SRW.APPLY_DEFINITION, 10-3, 10-16, 10-18
SRW.CANCEL_REPORT, 11-7
SRW.CLEAR_PARAMETER_LIST, 11-4
SRW.JOB_IDENT, 11-5
SRW-Package, 11-2
SRW.REMOVE_PARAMETER, 11-3
SRW.REPORT_STATUS, 11-5
SRW.RUN_REPORT, 8-6
SRW.START_DEBUGGING, 11-7
SRW.STATUS_RECORD, 11-5
SRW.STOP_DEBUGGING, 11-7
SSL, 1-5, 3-35
sslport, rwservlet.properties, 3-35
SSO, 3-36
SSO, see single sign-on
ssoconn command keyword, A-69
status record, 11-6
statusfolder command keyword, A-71
statusformat command keyword, A-70
statusformat, showjobs parameter, 14-14
statuspage command keyword, A-72
stype command keyword, A-73
subject attribute, mail, 9-9
subject command keyword, A-73
symbol equivalents, 12-8
syntax, of commands, A-6
syntax, reports URL, 8-1
sysauth, rwservlet.properties, 3-31

T

taglib, 14-9
targets.xml, 13-1, 13-14
templates, rwserver.template, 3-3
text display, multilingual, 12-14
text reading order, 12-1
tnsnames.ora, 3-6
to attribute, mail, 9-8
tolerance command keyword, 1-7, 8-12, 8-13, 14-7, 14-8, A-74
trace configuration element, 3-19
  traceFile attribute, 3-20
  traceMode attribute, 3-20
  traceOpts attribute, 3-20
trace performance monitoring, 14-11 to 14-13
trace_all, 3-21
trace_app, 3-21
trace_brk, 3-21
trace_dbg, 3-21
trace_dst, 3-21
trace_err, 3-21
trace_inf, 3-21
trace_log, 3-21
trace_pls, 3-21
trace_prf, 3-21
trace_sql, 3-21
trace_sta, 3-21
trace_tms, 3-21
trace_wrn, 3-21
traceFile attribute, 3-20
tracefile command keyword, A-75
tracefile servlet parameter, 3-32
traceMode attribute, 3-20
tracemode command keyword, A-76
tracemode servlet parameter, 3-32
traceOpts attribute, 3-20
traceopts command keyword, A-76
traceopts servlet parameter, 3-32
tracing, 10-17
translating applications, 12-16
translation
  PL/SQL blocks, 12-16
  TranslationHub tool, 12-16
triggers, database, 11-8
true type big fonts, 12-14
type attribute, 3-28

U

uifont.ali, 12-8
unicode, 12-13 to 12-15
URL engine
  configuring, 3-36
  elements, 3-36
  sending a request to, 8-10

URL job requests, 8-1 to 8-15
URL syntax, 8-9
URL, runtime syntax, 8-1
urlparameter command keyword, A-78
USER_NLS_LANG, 12-9, 12-13, B-5
userid command keyword, A-79
userid parameter, events, 11-4, 11-8
UTF8, 12-14

V

variables, NLS environment, 12-2 to 12-10
variables, using with XML attributes, 9-2
version attribute, 3-6

W

Web listener, 1-5, 7-3
Web Object Cache, 14-6, 14-9
webserver_debug command keyword, A-80
webserver_docroot command keyword, A-80
webserver_port command keyword, A-81
Windows registry, 12-6
writing direction, of language, 12-8

X

XML
  adding a new query, 10-8
  adding formatting exceptions, 10-5
  adding hyperlinks, 10-7
  adding program units, 10-7
  applying, 10-16 to 10-22
  applying at runtime, 10-16
  applying customizations, 10-3
  applying definition in PL/SQL, 10-18
  applying multiple definitions, 10-17
  applying one definition, 10-16
  applying via PL/SQL, 10-18
  batch modifications, 10-21
  changing format masks, 10-5
  changing styles, 10-4
  creating cross-product groups, 10-12
  creating customizations, 10-3, 10-3 to 10-9
  creating data models, 10-9 to 10-16
  creating formulas, 10-13
  creating group hierarchies, 10-11
  creating matrix groups, 10-12
  creating multiple data sources, 10-9
  creating parameters, 10-14
  creating placeholders, 10-13
  creating summaries, 10-13
  customization tracing options, 10-23
  debugging, 10-24
  debugging customizations, 10-22 to 10-24
  distribution.dtd, 9-2, 9-33
  interpreting, 10-2
  linking data sources, 10-10
  opening in Reports Builder, 10-23
  parser error messages, 10-22
  report customizations, 10-1 to 10-24
  reports.dtd, 10-2
  required customization tags, 10-4
  running by itself, 10-21
  targets.xml, 13-1, 13-14
  using distribution XML file, 9-32
XML attributes, using variables with, 9-2
XML, advanced distribution, 9-1 to 9-33
XSL, distribution.xsl, 9-33